A feasibility study on the use of agent-based image recognition on a desktop computer for the purpose of quality control in a production environment

Bertram Peter Haskins
N.Dip., B.Tech.

Study submitted in accordance with the requirements for the degree

MAGISTER TECHNOLOGIAE: INFORMATION TECHNOLOGY

in the

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

of the

FACULTY OF ENGINEERING, INFORMATION AND COMMUNICATION TECHNOLOGY

at the

CENTRAL UNIVERSITY OF TECHNOLOGY, FREE STATE

2006

Supervisor: Mr P Veldtsman
Co-Supervisor: Mr CH Wessels

Declaration

I, **BERTRAM PETER HASKINS**, identity number 8110115041088, and student number **20002696** do hereby declare that this research project which has been submitted to the Central University of Technology, Free State for the Degree **MAGISTER TECHNOLOGIAE: INFORMATION TECHNOLOGY**, is my own independent work; and complies with the Code of Academic Integrity, as well as other relevant policies, procedures, rules and regulations of the Central University of Technology, Free State; and has not been submitted before by any other person in fulfilment (or partial fulfilment) of the requirements for the attainment of any qualification.

_____ _____

SIGNATURE OF STUDENT DATE

Acknowledgements

The author would like to thank the following people for all their help and understanding:

- *Pieter Veldtsman and Casper Wessels;* my supervisors. Without their help, I would never have navigated through these unknown waters.
- *Professor GD Jordaan;* for adding that little bit of extra polish.
- *My parents.* Without their support and motivation, I might have given up before ever completing the study.
- *My friends;* for putting up with my anti-social behaviour these past few months.
- *The Good Lord*; for giving me the strength to pull through.

Preface

This study is the culmination of a lot of hard work and sleepless nights, but seeing the completed project at the end of the day makes it all worthwhile.

"Nothing ever comes to one, that is worth having, except as a result of hard work."

Booker T. Washington; teacher and political rights activist in America in the late 1800's and early 1900's.

Summary

A multi-threaded, multi-agent image recognition software application called RecMaster has been developed specifically for the purpose of quality control in a production environment. This entails using the system as a monitor to identify invalid objects moving on a conveyor belt and to pass on the relevant information to an attached device, such as a robotic arm, which will remove the invalid object.

The main purpose of developing this system was to prove that a desktop computer could run an image recognition system efficiently, without the need for high-end, high-cost, specialised computer hardware. The programme operates by assigning each agent a task in the recognition process and then waiting for resources to become available. Tasks related to edge detection, colour inversion, image binarisation and perimeter determination were assigned to individual agents.

Each agent is loaded onto its own processing thread, with some of the agents delegating their subtasks to other processing threads. This enables the application to utilise the available system resources more efficiently.

The application is very limited in its scope, as it requires a uniform image background as well as little to no variance in camera zoom levels and object to lens distance. This study focused solely on the development of the application software, and not on the setting up of the actual imaging hardware. The imaging device, on which the system was tested, was a web cam capable of a 640 x 480 resolution. As such, all image capture and processing was done on images with a horizontal resolution of 640 pixels and a vertical resolution of 480 pixels, so as not to distort image quality.

The application locates objects on an image feed - which can be in the format of a still image, a video file or a camera feed - and compares these objects to a model of the object that was created previously. The coordinates of the object are calculated and translated into coordinates on the conveyor system. These coordinates are then passed on to an external recipient, such as a robotic arm, via a serial link.

The system has been applied to the model of a DVD, and tested against a variety of similar and dissimilar objects to determine its accuracy. The tests were run on both an AMD- and Intel-based desktop computer system, with the results indicating that both systems are capable of efficiently running the application. On average, the AMD-based system tended to be 81% faster at matching objects in still images, and 100% faster at matching objects in moving images.

The system made matches within an average time frame of 250 ms, making the process fast enough to be used on an actual conveyor system. On still images, the results showed an 87% success rate for the AMD-based system, and 73% for Intel. For moving images, however, both systems showed a 100% success rate.

Opsomming

'n Multi-draad, multi-agent voorwerperkennings-sagtewareprogram, naamlik RecMaster, is ontwikkel vir die spesifieke doel om kwaliteitbeheer in 'n produksie-omgewing toe te pas. Dit behels dat die stelsel as 'n monitor gebruik word om ontoepaslike voorwerpe wat op 'n vervoerband beweeg, uit te ken en die inligting na die relevante aangehegte toestel te stuur, byvoorbeeld 'n robotiese arm, wat die foutiewe voorwerp dan sal verwyder.

Die hoofdoel met betrekking tot die ontwikkeling van die stelsel was om te bewys dat 'n tafelrekenaar 'n voorwerperkenningsisteem effektief kan hanteer, en dus die aankoop van duur, gespesialiseerde rekenaarhardeware kan uitskakel. Die program funksioneer deur aan elke agent 'n taak in die erkenningsproses toe te ken en dan te wag vir bronne om beskikbaar te raak. Take wat verband hou met randopsporing, kleurinversie, beeldbinerisering en omtrekbepaling is aan individuele agente toegeken.

Elke agent word op sy eie prosesseringsdraad gelaai, waarna sommige van die agente hul subtake delegeer aan ander prosesseringsdrade. Die tegniek stel die program in staat om die beskikbare stelselbronne beter te benut.

Die toepassing is baie beperk in sy omvang, aangesien dit 'n eenvormige beeldagtergrond benodig, sowel as ‘n geringe tot geen afwyking in die vergrotingsvlak van die kamera en die afstand van lens na voorwerp. Hierdie studie het slegs gefokus op die ontwikkeling van die sagtewareprogram, en nie op die opstel van die optiese hardeware nie. Die optiese toestel waarop die stelsel getoets is, was 'n webkamera wat teen ‘n 640 x 480 resolusie kan werk. Alle foto's is dus geneem en verwerk teen 'n horisontale resolusie van 640 pieksels en 'n vertikale resolusie van 480 pieksels ten einde te verseker dat die kwaliteit van die beeld nie verlaag word nie.

Die program spoor voorwerpe op in ‘n beeldinvoer, wat die vorm van ‘n foto-, videoleêr- of lopende kamera-invoer kan wees. Hierdie voorwerpe word dan met ‘n voorafgemaakte model van die voorwerp vergelyk. Die koördinate van die objek word vasgestel en na koördinate op die vervoerband omgeskakel. Hierdie koördinate word dan deur middel van ‘n seriekoppeling na ‘n eksterne ontvanger soos ‘n robotiese arm aangestuur.

Die stelsel is op ‘n model van ‘n DVD en ‘n verskeidenheid soortgelyke en nie-soortgelyke voorwerpe toegepas ten einde die akkuraatheid daarvan te toets. Die toetse is op beide AMD- en Intel-gebaseerde tafelrekenaarstelsels gedoen. Die resultate het getoon dat beide die stelsels die sisteem effektief kan behartig. Die AMD-gebaseerde stelsel was gemiddeld 81% vinniger in die proses om voorwerpe en ‘n model bymekaar te bring in foto’s, en 100% vinniger in die proses om voorwerpe en ‘n model bymekaar te bring in bewegende beelde.

Die model-voorwerpkoppelings wat die stelsel gemaak het, is binne ‘n gemiddelde tydsduur van 250 ms gedoen. Dit maak die proses vinnig genoeg om op ‘n werklike vervoerband gebruik te word. Op foto’s het die AMD-gebaseerde stelsel 87% sukses behaal, en die Intel-gebaseerde stelsel 73%. Beide stelsels het 100% sukses behaal op bewegende beelde.

Table of Contents

List of Figures

Chapter 2

Chapter 3

Chapter 4

Appendix A

List of Tables

Chapter 2

Chapter 3

Chapter 4

Chapter 5

Appendix A

Appendix C

List of Abbreviations

3D	Three-dimensional
AMD	American Micro-Devices
AVI	Audio-Video Interchange
CD	Compact Disc
CD-R	CD-Recordable
CCD	Charge-coupled Device
cm	centimetre(s)
cm/s	centimetres per second
CPU	Central Processing Unit
DVD	Digital Versatile Disc
DVD-R	DVD-Recordable
IC	Integrated Circuit
JPEG	Joint Photographic Experts Group
m	metre(s)
m/s	metres per second
MAS	Multi-Agent System
MB	Megabyte
ms	milliseconds
s	second(s)
SI	Still Image
TV	Television

Chapter 1

1. INTRODUCTION

Over the years, desktop computing applications have evolved from simple data capturing tasks to the complicated desktop applications we have today. Processing power has increased exponentially, and along with it the complexity of applications that can be run. Applications that were previously in the domain of super and mini computers are now moving down to the desktop microcomputer domain. Yesterday's cutting edge has become today's utility blade.

1.1 Current trends

Fabrication plants formerly needed expensive high-end computing power to keep the fabrication process running smoothly. Most of this processing power went towards operating robotic limbs and other complicated fabrication machinery. The inspection of fabrication quality was normally a hands-on process, with rows of workers manually inspecting each part. But with stronger, more affordable computing power available, a shift was made towards computer-controlled quality inspection. Although the image processing done in quality control will always be a very processor-intensive operation, the rapid progress of modern technology has now made it possible to run these systems on a desktop environment.

1.2 The purpose of this study

The purpose of this study was to develop such a quality control image recognition system and have it run on a desktop system. To make the study a bit more interesting and to utilise the power of the desktop system more fully, some of the application's tasks were threaded, thus allowing them to run

simultaneously on a single-processor desktop system, as they would in a multi-processor system. Different processing tasks were also allocated to pieces of software code called agents. Each of these independent pieces of code completes its task in isolation before transferring its results to the next agent in line. That means that each of these pieces of code work together to accomplish the image processing task. The theory behind these technologies is discussed to a limited extent in the second chapter of this study.

1.3 Study Overview

The main part of this study consisted of the actual development of an agent-based, multi-threaded image recognition application called RecMaster, as detailed in Chapter 3, Appendix A and Appendix B (on the included disc). The scope of the study was very narrow, as it only deals with the recognition of objects at a fixed distance from the capture device in relatively constant light conditions. The optimal placement of imaging devices did not form part of this study, as it is more hardware-related. The RecMaster application's acceptance or rejection of an object is based on a model that was previously created by the application user. The coordinates of any object found is then output via a serial link to the intended recipient device or application. The included version of RecMaster was developed in Visual Studio 2003 and does not include serial link capability; a recompiled Visual Studio 2005 version (not included) does, however, make provision for variable output via serial link.

The results of benchmarking the system on both an AMD- and Intel processor-based system are available in Chapter 4, as well as in Chapter 5, which deals with the conclusions drawn from this study.

1.4 The RecMaster system

The main reason behind this study was to aid in the trial of a conveyor-based system under development at the Faculty of Engineering, Information and Communication Technology of the Central University of Technology, Free State. This system will incorporate a vision-based quality control system to determine whether items running on a conveyor from a rapid-prototyping device have been correctly manufactured or not. If an imperfection is detected on the manufactured item, coordinates need to be passed to a robotic arm, which will remove the defective item. The RecMaster system has, however, not been tested with an attached device such a robotic arm, as this does not fall within the scope of the development process.

1.5 Study hypothesis

The aim of this study is to prove the following:

Modern desktop computing systems are able to reliably perform recognition-based quality control functions at an acceptable rate in an industrial environment, using a multi-threaded, agent-based image recognition system using a digital camera, where:

- *An acceptable rate is defined as matching objects to predefined models within the period of time that the object is within complete view of the associated digital camera system, and*
- *Reliable performance is considered as providing consistent, accurate results.*

1.6 Satisfying the hypothesis

The hypothesis will be satisfied if the following criteria are met:

1. The RecMaster system is able to perform image recognition by matching an object to a previously created model at least 75 % of the time.
2. The RecMaster system is able to perform matching on still images within a time frame of at least 500 milliseconds.
3. The RecMaster system is able to perform matching on a video or camera feed while the object is still within the viewfinder at least 75 % of the time.
4. The RecMaster system is able to perform these matches consistently, i.e. creating matches at least 75 % of the time.

1.7 Overview of the remaining chapters

Chapter 2 will shed some light on the underlying agent and machine vision technologies, and will also provide a brief insight into what threading signifies.

Chapter 3 discusses the underlying logic behind the development of the RecMaster image recognition programme. The chapter focuses on the processes involved in building up a model reference as well as how the model is used to perform recognition. Code snippets as well as diagrams are included to aid in the understanding of these processes.

Chapter 4 consists of the results obtained from using the RecMaster programme on different input feeds to perform recognition. The system has been tested on two different PC architectures in order to see whether systems from different manufacturers will be able to properly run the RecMaster programme.

Finally, Chapter 5 contains the conclusions drawn from this study as well as any thoughts on further system development.

Chapter 2

2. AGENT AND VISION TECHNOLOGIES

This chapter is a review of the basic approaches used in agent-based, threaded and computer vision applications, as well as instances where they have been successfully used in conjunction with one another. By no means does it presume to lay out all of the technical nuances of these various technologies and concepts, but rather acts as a general introduction in order for the reader to understand RecMaster's underlying technologies.

2.1 Agents

2.1.1 A general description

Computer programmes normally do what the user tells them to, performing a set of predefined algorithms when the user elects to do so. Agents, however, are programmes that are initialised once and can then run without any further user interference. They are computer programmes capable of flexible, autonomous action - and may support a wide range of different behaviours, defined as rules or based on machine learning algorithms such as neural networks.

Sycara [71] defines an agent as a set of conflicting tasks where only one can be active at any given time (a task being a high-level behavioural sequence, as opposed to the low-level actions performed directly with actuators). Another more formal definition of agents is: "Anything that can be viewed as *perceiving* its environment through *sensors* and *acting* upon that environment through *effectors*" [56].

2.1.2 Agents in practice

Agents are frequently used when lots of small programmes or algorithms need to be run at the same time. These programmes tend to share the same basic code, with slight variations to give them individual functions, but this need not be the case. Agents form membership in a unit on a cooperative basis, with each agent contributing some knowledge, but not enough to solve the problem. They communicate with one another via messages on a shared blackboard [35]. The blackboard is simply a central repository for all shared information. The 'chalk' is controlled by a facilitator, allowing agents to write to the blackboard in the order that best suits the problem. Such a blackboard session typically begins by a problem being written onto the blackboard. As individual agents realise they can contribute to the solution with their own expertise, they approach the facilitator and are allowed to add new information to the blackboard. In some intelligent agent models, the agents negotiate deals amongst themselves to ensure that the most effective agents contribute the solution.

Agents have been used to present weather reports on mobile phones, drive trucks, monitor environments to support life on other planets and perform many other sophisticated tasks [15]. Agents are especially useful as monitors – for instance, the Australian Bureau of Meteorology [21] makes use of agents in conjunction with people to monitor even the slightest changes in weather. Some medical equipment also makes use of agents to monitor and assist in the real-time application of Medical Protocols in a distributed hospital environment with computer-based medical records [2]. Thus, agents can be used to assist or even replace people in many different types of applications.

Agents are most useful in open systems. An open system is one in which the structure of the system itself is capable of dynamic change [71]. The characteristics of such a system are that its components are not known in advance, i.e. they can change over time. The best-known example of such an

open system is the Internet. Agents are all over the Internet, in the form of search engine spiders. These agents locate sites and store their indexes in a database for users to browse [53]. Thus, a common trait in all kinds of agents is that they can act or perform specific tasks in the place of someone or something else [50]. This makes agents a valuable tool in our ever-advancing technological society, as we are always striving to find ways to reduce our own workload and make processes more efficient. Agents will enable us to make use of our computing resources more efficiently, thus freeing up time that would normally be spent on menial tasks and putting it to better use.

2.1.3 The agent's environment

An (autonomous) agent is a system that is integrated into an environment to such an extent that it acts as part of it, sensing what will happen in the future and acting on this in pursuit of its own agenda to affect future events [28]. In particular, an agent receives information about its environment via sensors and has the ability to change this environment through actions [67]. If the environment is changed, the agent may no longer be viable – for instance, if an agent can only interact using network protocols, but there are none in use on the system, then the agent cannot function.

The environment can be defined as whatever provides input and receives output. If receiving input is likened to sensing and producing output to acting, then every programme is an agent. Thus, if we want to arrive at a useful contrast between agent and programme, some of the notions of environment, sensing and acting must be restricted [28].

2.1.4 Programme or agent?

A normal software programme runs once and then goes to sleep until it is called again; either by the user or by an explicit command built into the system. In other words, the programme does not have temporal continuity [36]. A programme that is classified as agent normally runs all the time, or rests until it senses the need to be active again. Agents are sometimes defined as software that takes action without user intervention and operates concurrently while the user is either idle or taking other actions [50] - in other words, there is no need for the user to activate or control it after its initial activation. Most software programmes have some form of input through which they sense and interact with their environment. This input is then acted upon to create output. When this output affects what the programme senses as its input, then such a programme might be classified as an agent. Thus, all software agents are programmes, but not all programmes are agents [28].

The flowchart in Figure 2 - 1 demonstrates how a traditional programme would sequentially process its assigned tasks. Even though a needed resource has already been created, the processes have to wait until they are programmatically called to act upon the resource.

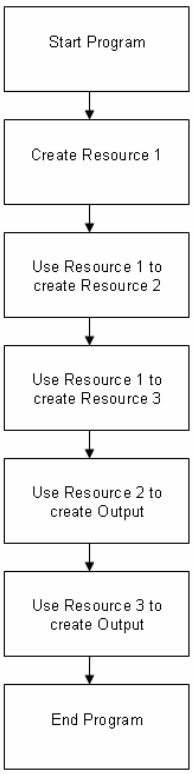

Figure 2 - 1 Normal program flow

If the programme is changed to an agent-based system, it would operate more efficiently. All the agents are created after the programme is initiated; all they need to do now is to wait for their required resources to become available and process them immediately as they do. This process is demonstrated in Figure 2 - 2.

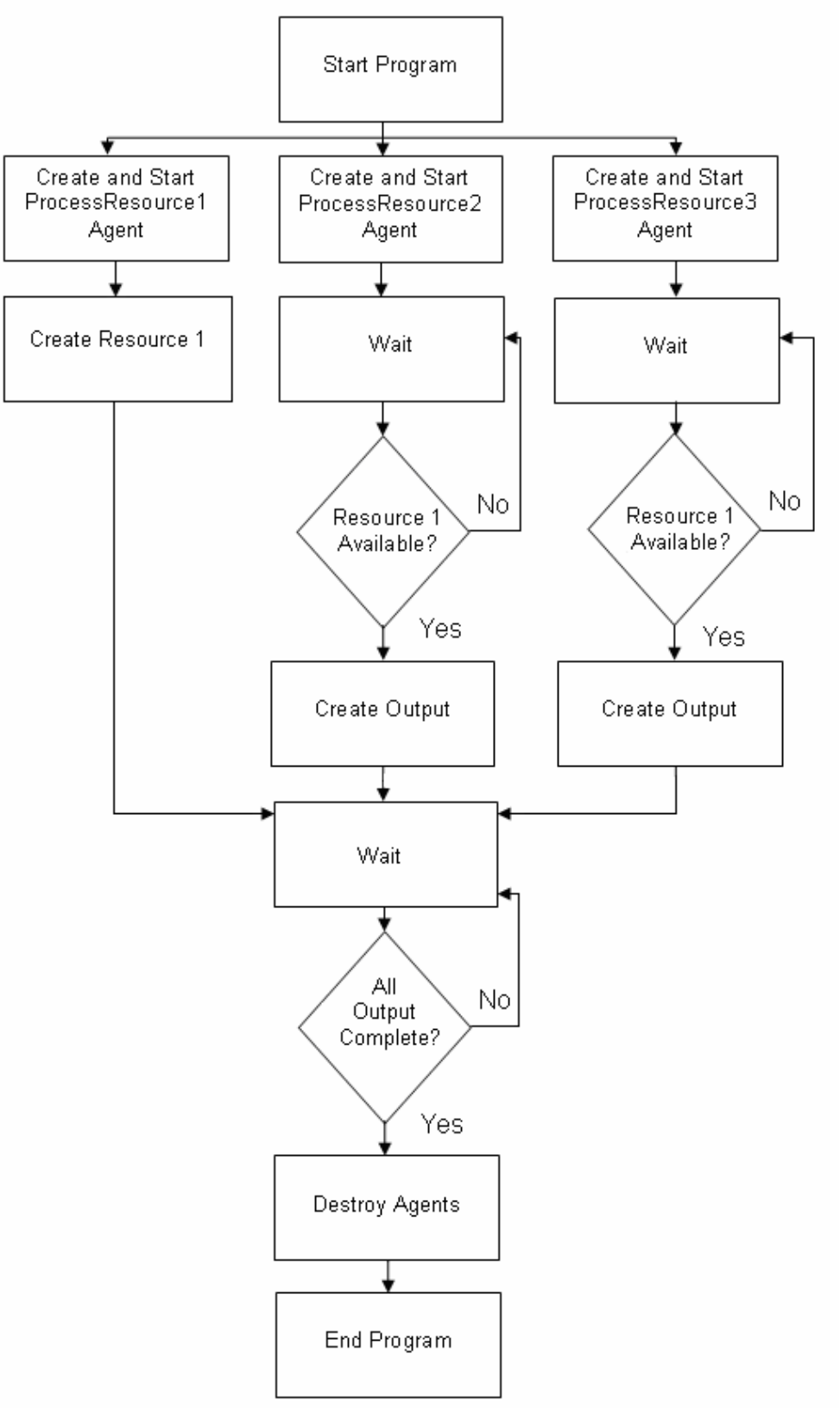

Figure 2 - 2 Program flow when agents are used

2.1.5 Multi-agent systems

It has been stated that agents are indeed forms of programmes, so - if conventional programming logic prevails - we can conclude that they may have subroutines, which - in turn - might be agents in their own right. This kind of system is called a Multi-Agent System (MAS). Table 2 - 1 describes the characteristics a MAS requires, as well as how the RecMaster system fits these characteristics.

Table 2 - 1 MAS and RecMaster characteristics

MAS	RecMaster
An environment	*Images loaded into the RecMaster system*
Objects and agents (the agents being the only ones to act)	*Objects: images* *Agents: All the agents defined in the RecMaster system.*
Relations between all the entities	*Agents act on objects and share results.*
A set of operations that can be performed by the entities	*Each agent has its own assigned task.*
Changes that take place in the environment over time, and also due to the agents' actions [27]	*Each agent acts on the image object and changes it before transferring it to another agent.*

A MAS usually consists of a population of autonomous agents that cooperate with one another to reach common objectives. These systems offer modularity, which means that, if a problem domain is complex, large or unpredictable, it can be better addressed by splitting it up into a number of functionally specific modules. This subdivision allows each agent to use the most appropriate solution for solving its particular problem [71]. This normally works using a layered architecture, in which each layer acts independently. The layers may be implemented horizontally or vertically [55]. In a horizontal implementation, each layer senses the environment (raw sensor data) and acts on it. It does not need

to confer with any other layer to perform its task. This is typically called a discrete multi-agent system. A multi-agent system is *discrete* if it is independent, and if the agendas of the agents bear no relation to one another [26]. In a vertical implementation, the layers are stacked. Agents still act autonomously, but only the bottom-most layer deals with the actual raw sensor data as its input. This layer creates an output, which is sensed by the next layer and used as input. So it continues up to the top layer, with each layer using the previous layer's output as input. This implementation is typically called fully connected.

The developed system is implemented in such a manner, with the bottom-most agent acting upon the raw input image and then passing a processed image to the next agent, and so on and so forth. The website www.calresco.org defines "fully connected" as every agent interfacing with every other agent in the system [51].

Planning the steps to be performed in a single agent system is quite straightforward, and consists simply of constructing a sequence of actions that take into consideration goals, capabilities and environmental constraints. Planning a MAS is more complicated in that the actions other agents take can have an effect on an individual agent, as each individual agent would have to wait for another agent to compute information it requires to continue [44].

Multi-Agent Systems are usually used to solve problems that are too large for a centralised agent to solve [70], as it may be too resource-intensive or too risky to have everything done by a single agent. MAS may also be used to connect multiple existing legacy systems [41]. Since it may prove inordinately expensive to completely recode an old software programme to make it compatible with new technology, it may be more cost-effective to encapsulate the existing programme in an agent wrapper and have the agent interact with the rest of the system. This method can be used to have several legacy programmes interact with each other. MASs have the ability to use the computational resources of the host

hardware system more efficiently. They are more reliable, as another agent may be able to take over the work of a failed agent. Programmes that use MAS architecture can also be extended and maintained more easily, as there is less coding to be revised than in a normal subroutine-based programme. Due to their inherent flexibility these agents can be reused in another system, thus saving time, money and resources.

Designing such a MAS is not easy, since you will have to deal with all the problems of a traditional distributed programme, as well as the dynamic and sometimes unpredictable behaviour of individual agents. In order to even attempt such an endeavour you will need a well-planned design, as well as an application that lends itself perfectly to MAS architecture, or the entire exercise will be a waste of time and resources.

2.1.6 Disadvantages of using agents

It will be evident from the discussion of many of the functions and nuances of agent architecture thus far that agent-based applications provide numerous advantages over normal top-down applications - in fact, they even serve to improve the performance of conventional legacy software. However, like any other approach this technique is not without flaws. Thus, agents also have a few disadvantages, of which Sycara [71] lists the following:

1. It is difficult to formulate, describe, decompose and allocate problems and study results among a group of intelligent agents.
2. Enabling agents to communicate and interact is not always easy.
3. Getting agents to act coherently when making decisions or taking action may prove difficult and time-consuming.
4. Individual agents need to be enabled to represent and reason about the actions, plans and knowledge of other agents, and to coordinate with them.

5. Agents need to recognise and reconcile disparate viewpoints and conflicting intentions amongst themselves in order to coordinate their actions.

2.1.7 An organisational framework for agents

Agents make use of organisations to provide a framework for their interactions. This framework facilitates the definition of roles, behaviour expectations and authority relations. Organisations are generally, conceptualised in terms of their structure, namely the pattern of information and control relations that exists among agents, as well as the distribution of problem-solving capabilities [71]. Using a structure provides every agent in a MAS with information on how the group functions, solves problems and distributes tasks among its members. In a more open-ended environment, such as the Internet, there is no definite way of imposing structure. Agents enter and exit systems on the Internet without user knowledge or intervention. Agents that operate in an open environment sometimes request the use of another agent, called a middle agent, to help gather the information or resources they need [24]. Middle agents are capable of either gathering information on their own or delegating their tasks to other middle agents, until the original agent's request has been fulfilled. To determine where agents fit into the organisational structure they may be classified on the basis of some general traits (listed in Table 2 – 2), taken from a sampling of popular literature [19] [57] [59]:

Table 2 - 2 General agent traits

Trait	Description
Autonomy	*The degree to which agents can act on their own.*
Cooperation	*The ability to communicate with other agents.*
Hybrid	*This is a combination of some of the aforementioned traits.*
Mobility	*The ability to move around in an environment.*
Role	*This refers to where the agents are used.*
Reactivity	*The ability to react appropriately to their environment.*

Assigning individual tasks to agents before execution has definite advantages. When agents are informed of their task beforehand, less communication is needed between individual agents during run-time thus freeing up valuable computing resources. In a desktop application environment in which all tasks are known beforehand, this method of task distribution will provide the optimum usage of resources. The reason for this is that, unlike applications that run in a dynamic online environment, desktop applications normally have a predetermined set of static tasks to perform. This would allow the programmer to permanently assign a task to each agent in order to make better use of the limited desktop resources and ensure that no resources are wasted as agents try to dynamically decide, amongst themselves, which one is better suited to the task. Another advantage of pre-programmed agents is that they reduce the chance of individuals wasting time by performing the same task, thus creating the possibility of solution inconsistency.

There is, however, a major drawback to the static assignment of tasks to agents beforehand: if too many static assignments are made, the programme will not be readily adaptable to a dynamic environment. However, if the environment is well mapped out and predictable, static assignment of agents is preferable. Dynamic allocation of agents takes place according to the Producer/Consumer model [10].

An agent that receives a task to perform will first determine whether it can break the task down into subtasks that can be performed concurrently. If so, this agent acts as the Producer. The Producer announces that there are tasks to be performed. Individual agents (Consumers) then bid on the tasks, which are assigned on a best-bid basis. This dynamic architecture makes good use of the available processing power, as there is always another agent to pick up the slack if the tasks become too much. Although additional resources need to be set aside for the agent communication process, this is a small price to pay for the greater efficiency and expandability they provide.

The option of static implementation was chosen for the development of the RecMaster system. The reason for this decision is that, firstly, since the system is not yet so huge that dynamic allocation would provide a significant performance boost, it may - in fact - be detrimental, causing extra overhead. Secondly, it adds reliability, as the developer always knows which part of the image processing each agent will handle.

The next section will deal with the processes involved in Machine Vision and Pattern Recognition.

2.2 Machine Vision and Pattern Recognition

2.2.1 A general description

Although Machine Vision is a well-researched part of the field of Artificial Intelligence, entirely satisfactory solutions have only been found for a very small number of problems. Machine vision focuses on providing computers with the functions typical of human vision. However, most current computer vision systems are vastly different from biological vision systems and, in most ways, inferior to such systems. One definition of Computer Vision is “to make useful decisions about real physical objects and scenes based on sensed images” [68].

In the field of industrial automation alone, its applications include guidance to enable robots to correctly pick up and place manufactured parts, as used in a system called Visionscape Express, developed by the MASS Group [52]. Other uses include non-destructive quality and integrity inspection, and on-line measurements. The use of machine vision technology is growing very rapidly, spurred by manufacturers' need for increasingly fine control over the quality of manufactured parts.

Machine vision technology uses an imaging system and a computer to analyse an image and to make decisions based on that analysis. The most basic types of machine vision applications are those used for inspection [72] [64] and control. In inspection applications, the machine vision optics and imaging system enable the processor to "see" objects precisely, and thus make valid decisions regarding the identification of parts to be approved or rejected. Such an inspection application was developed for this study. In control applications, sophisticated optics and software are used to direct the manufacturing process or even the movement of some form of robotic apparatus - such as that used in the KARES system, which consists of a wheelchair-mounted robotic arm [47]. It is really hard to draw a fine, distinguishing line between inspection and control applications, since inspection applications will still provide some means of control when they direct a robotic arm to remove a rejected part, while a control application will need to constantly inspect its work to ensure that it is machining the parts correctly. Inspection and control applications therefore run the same basic processes, but viewed from different perspectives.

Typically, applications utilise a video or still camera placed above or to the side of the inspection point. The camera captures an image of a part and sends it to a vision processor. The camera captures an image of a part and sends it to a vision processor. From there on, the process of pattern recognition takes over.

2.2.2 Pattern recognition

Pattern Recognition is an information reduction process: the assignment of visual or logical patterns to classes based on the features of these patterns and their relationships [45]. It deals with the classification of objects in pattern categories. Each category is unique, based on specific features. The application attempts to classify each object of the image in a category upon comparison of the individual features with the features of the general category [30].

The basic premise of pattern recognition is that there is an unknown object or set of objects that needs to be analysed by the programme. During analysis these objects need to be broken down into familiar constructs to aid the eventual recognition of the object via its sum parts.

According to popular literature [45] [75], Pattern Recognition can be broken down into eight generally identifiable parts. These eight parts, as well as which sections they are discussed in, are listed in Table 2 – 3.

Table 2 - 3 Pattern Recognition breakdown

	Concept or process	Discussed in section
1	*Data Collection*	2.2.2.1
2	*Registration*	2.2.2.2
3	*Preprocessing*	2.2.2.3
4	*Segmentation*	2.2.2.4
5	*Normalisation*	2.2.2.5
6	*Feature extraction and edge detection*	2.2.2.6
7	*Recognition*	2.2.2.7
8	*Training*	2.2.2.8

2.2.2.1 Data collection

In order to initiate pattern recognition, we need a set of measurements to build up a pattern vector. Mathematically, a pattern can be represented as a multi-dimensional vector, with the components in the vector corresponding to the elements in the pattern [40]. Depending on the needs of the application, this may take place either two- or three-dimensionally. Either technique used will generate data, which will need to be stored for analysis. These storage techniques are discussed in section 2.2.2.1.3.

2.2.2.1.1 Two-dimensional techniques

Two-dimensional techniques normally rely on a video signal captured by either a TV or a CCD camera. A TV camera works by focusing an image onto a photoconductive target [16]. The target is then scanned line by line by an electron beam, which produces an electrical current as the beam passes over the target. The strength of the current is directly proportional to the intensity of the light striking the photoconductive target. This current is then translated into a video signal.

A CCD camera works on the same premise as a TV camera, but provides the solution as a single IC device. The video signal output by a CCD camera also has less geometric distortion and a more linear form of video output. Care should be taken to ensure that the data collected by the equipment is of the highest quality available. This is achieved by ensuring that pictures are taken at the highest resolution available, that the camera is always in focus and that there is optimal and predictable lighting in the area to be photographed. These measures need to be taken to limit additional noise on the raw image to the minimum.

The video signals obtained from either the TV or CCD camera are then fed into a device called a frame grabber, which digitises the signal. This process makes the sample signal easily accessible by storing its individual frames in computer memory or as a file. This allows the video signal to be processed numerically by the computer.

This study uses the two-dimensional approach. Any single camera (or other attached video capture device), video file or single image will provide appropriate input. The only catch is that the images taken from any of these diverse sources all need to have the same object to lens distance and a constant zoom setting, as the system does not compensate for size differences caused by variance in distance or level of zoom. This study focuses solely on the development of the system itself; the correct placement of imaging devices to take advantage of environmental conditions does not fall within the scope of this study.

2.2.2.1.2 Three-dimensional depth imaging

Three-dimensional techniques add extra information to the captured image in the form of depth or perspective. The simplest and most convenient way of representing and storing the depth measurements taken from a scene is a depth map. According to Shirai [69], a depth map is a two-dimensional array where the x and y distance information corresponds with the rows and columns of the array as in an ordinary image, and the corresponding depth readings (z values) are stored in the array's elements (pixels). Depth maps are like greyscale images except that, instead of intensity information, they contain information about the z-axis.

Several methods can be used to acquire the 3D depth image. The first method is known as laser ranging [1]. This method works by directing a laser beam at the object to be measured. Any light reflecting off the object is measured according to the time taken for the reflected light to bounce back towards the receiver. This

time duration is then used to calculate depth. These kinds of systems are normally used for longer distances (e.g. calculating the distance from the Earth's surface to the moon), rather than short distances.

Another method of acquiring a 3D depth image is called Structured Light [65]. In this method, a pattern of light (e.g. a grid) is projected onto the object in question. The shape of the object's surface is then deduced from any distortions of the pattern on the object's surface. If the camera and projector geometry are precisely known, then depth can be deduced by a process of triangulation.

The next method is called the Moiré Fringe Method [34], and is actually a derivative of the Structured Light method. It also makes use of a projected pattern - in this case specifically a grid pattern - but employs a second reference grid pattern that is situated in front of the image acquisition device. An image is then formed on the plane of the reference grid pattern, which causes interference. These interferences form Moiré Fringe contour patterns, which appear as dark and light stripes [69]. Although this method is capable of producing very accurate depth data, it is computationally expensive and large angles may sometimes be missed when too many fringe patterns are formed.

A less widely used method is known as Shape from Shading [39]. In this method, a single fixed camera is used to capture 2 or more images of a specific object. Each of the images captured will have different lighting conditions. Although some depth information may be obtained from the changes in brightness on the object's surface, this method is more suited to two-dimensional image acquisition since it's a single camera implementation. For more reliable 3D image acquisition, a second camera is needed. This method is known the Passive Stereoscopic Method, but may still not produce totally reliable depth maps. The Active Stereoscopic Method, on the other hand, makes use of a strong light source (normally a laser beam) and is employed in industrial

applications that provide a very controlled environment, thus producing very reliable depth maps.

2.2.2.1.3 Storage considerations

A lot of storage space will be required for all the collected data used to create a pattern recognition system. Image processing tends to eat up hard-drive space rapidly, since storage space needs to be reserved for the video being streamed and the images currently in use, as well as the store of images used in training the recognition model. In addition to all this storage space, there is also a need for secondary or back-up storage. In a desktop system, this will most probably take the form of CD-R or DVD-R. The amount of storage space needed can be reduced by compressing the images. Depending on whether you make use of a lossless or lossy algorithm [58], you may have to sacrifice some of the smaller details in the image. Rather take fewer data samples than more samples of lesser quality - even though more samples might give a better data range, it may not be entirely accurate. It is therefore preferable to take a few high-quality samples to obtain a smaller data range, but one that is far more representative of the object being modelled.

The RecMaster system, which was developed for the purpose of this study (as outlined in Chapter 1), contains a few different items that will influence the required capacity of the storage medium. The first item is the actual model information files. These files are generally small, as they are only text files; however, the model directory can grow in size as EdgeGraphs are added to the model. For each image or frame used to train the model, an EdgeGraph is added to the model directory. This EdgeGraph has the same resolution as the original file, and is stored in JPEG (lossy) format.

The system also provides users with the option of acquiring image captures from a video file or camera (device) source. These images are also stored in JPEG

(lossy) format and have a resolution of 640 x 480 pixels, so they take up very little storage space.

The system components discussed so far would not really take up too much space in permanent or temporary (memory) storage. However, the system also makes provision for video capture from a camera (device) source. Video capture is a very processor- and memory-intensive operation; it also eats up hard drive storage at a very rapid rate, as it is done in an uncompressed .avi format. This means that a 30-second video file can use up as much as 200MB, not including the swap space used on the hard drive. The files can, however, be compressed with a third-party tool, which will significantly reduce the amount of storage space needed.

Thus, depending on how the system is used, it may take up very little storage space or a whole lot of space. As with any other application that handles video or audio files, it is always prudent to check the amount of storage space available before attempting any processing. As always, backups are essential - both for preserving the data and for conserving precious hard drive space.

2.2.2.2 Registration

The registration step can be regarded as the alignment of the data with the internal model of the system. Before any further processing of our image can take place, the system needs to determine exactly which parts of the image are of interest. This is facilitated by the RecMaster system having at least some prior knowledge of the environment in which the data capturing is done. When exercising quality control on a conveyor belt, it would be beneficial to know the exact size of the environment our device is focusing on – i.e. the length of the X- and Y-axis. It may also be necessary to know the speed and direction in which the conveyor belt is moving in relation to our device image. The system that was

developed facilitates the input of these variables, thus providing the means to match on-screen coordinates with their real-world counterparts.

2.2.2.3 Preprocessing

No matter how much we try to control the environment in which data captures are made, or even at how high a quality the camera settings are, there will always still be unacceptable levels of noise on the picture to some extent. Noise in an image means that there are many rapid transitions (over a short distance) in intensity from high to low and back again or vice versa, as faulty pixels are encountered [31]. Although noise can be anything that hinders an image recognition programme in performing its task, it may be caused by either the acquisition process or by the way in which the image is transmitted to the intended application – however, it is generally quite easily removed by running the data (image) through a filter. A good example for image recognition would be to run either a median filter - which removes point noise - or an edge sharpening filter, which makes edges more easily detectable. The RecMaster system is implemented without either of these two functions since they would only add another layer of processing, which might give a slight boost to recognition accuracy, but would certainly cause deterioration in overall system performance.

For any vision system to work effectively, the image must first be processed to remove any noise, improve the image's contrast or brightness, remove blurring caused by camera movement and correct geometrical distortions caused by the lens. These techniques ensure that all images upon which the vision system operate are properly normalised, thus reducing the risk of deviances affecting the recognition process. The option of adjusting brightness or contrast is available in the RecMaster system whenever still images are processed. This allows the user to normalise the images slightly, which makes the results more reliable.

During image processing, an array or matrix of pixels is normally used as input to produce another improved array of pixels. Ballard and Brown [6] suggest that image processing methods may be broadly divided into two classes as described in Table 2 – 4.

Table 2 - 4 Image processing methods

Method	Description
Real Space methods	*These methods directly process the input pixel array. This can be done by working on the image in a byte by byte manner, with each byte containing either a red, green or blue value in a 24-bit image. Another way is to analyze the pixels themselves, i.e. including their colour values, as single entities and then detract some meaning from them. The size of the input array depends on the size of the image. A 640 x 480 resolution image consists of an array of 640 horizontal and 480 vertical pixels. Both the byte and pixel handling techniques were used in the implementation of the RecMaster system.*
Fourier space methods	*These methods first derive a new representation of the input data by performing a Fourier Transform, which is then processed. When this is complete, an inverse Fourier transform is performed on the resulting data to yield the final output image.*

The process of image smoothing attempts to remove any noise that could be detrimental to the image recognition process, and may be performed on either the real space image or its Fourier Transform equivalent.

The Fourier method is a sampling method that is used to gain in-depth information about an image. Holota and Nemecek [37] define the Fourier method as converting an image under inspection into reciprocal space (Fourier space) –

i.e. expressing the image in terms of spatial frequencies. Any image has lines; brightness along these lines can be measured at equally spaced distances (also known as spatial frequency values). Each of these frequency values is referred to as a frequency component [6]. Since images consist of two-dimensional arrays of pixel measurements, this information can also be described as a two-dimensional grid of spatial frequencies. This whole process of converting an image into its frequency components is called the Fourier Transform, and yields what is known as the Fourier space description of the image. When we want to transform this Fourier space description back into its real (spatial) form we use a method called the inverse Fourier Transform, which basically just reverses the process.

Fourier frequency grids are very handy for interpreting the contents of an image. Large values at a high frequency mean that the data is changing rapidly over a short distance, e.g. a page of text, while large values at a low frequency mean that there are relatively few changes, making large-scale features more important - e.g. a large, simple object, such as a cube, in the centre of an image. Generally a Gaussian filter is run on the image before a Fourier Transform is performed; this smoothes over most of the salt-and-pepper (spot) noise on the image.

There are a number of techniques that may be used to reduce the high frequencies in a Fourier Transform:

1. Ideal Low-pass Filters
2. Low-Pass Butterworth Filters
3. Real Space Smoothing Methods

A discussion of these methods does not fall within the context of this study.

2.2.2.4 Segmentation

Segmentation is the partitioning of an image into parts that are coherent on the basis of a certain criterion [23]. This part of the process is actually not so well defined, and may take place alongside the registration and pre-processing subprocesses. During the process of segmentation the input data is split into meaningful subparts, which may be used for classification. Segmentation information is only passed along to the rest of the processes after the total amount of data has been processed. This may either be done in the form of tags or labels, added to the existing data (image), or the separate segmented parts may be passed along individually. It was decided to pass the information along individually for the RecMaster programme - either as a processed image or as recorded measurements.

2.2.2.5 Normalisation

In any range of data there will always be variances. These variances need to be cancelled out in order for the pattern recognition to succeed. A good example would be the recognition of an image of a teacup. In one image the teacup's ear is pointing left, while in another it is pointing right. It's still the same teacup, but in the 'eyes' of the system its dimensions will be different. Normalisation is the process through which the images are reconciled with each other in order to facilitate the image recognition process. In the case of the teacup example, the image just needs to be rotated. However, the process used in the RecMaster system does not need to rotate the image, as it actually measures the perimeter and area of the object and determines whether it fits into a specific bounding box. These variables would stay the same, no matter from which direction the image is viewed. Another example [45] is used in handwriting recognition: our handwriting is not always straight or upwards, but somehow slanted to the left or right. Estimating the slant and reversing it can achieve normalised characters.

2.2.2.6 Feature extraction and edge detection

The main purpose of feature extraction is to reduce the number of features to a set of a few significant ones while maintaining the classification rate [60]. During this process, the information that is deemed most relevant to the classification process is extracted from the raw data. If the feature extraction scheme was well designed, it should only select those distinct features that allow for classification and ignore those caused by human, technical or environmental variance. This limits the amount of memory and storage space needed during the pattern recognition process. It also speeds up the classification process by reducing estimation errors. Something to keep in mind during the feature extraction process is the aspect of scale or metric. In some applications, an aspect of varying distance might be involved in the image recognition input data. The system needs to perform a linear transformation on the data to ensure compatibility. In the RecMaster system distance and angle from the camera to the objects must always remain constant, as the system does not do any processing to correct variances in these fields.

One good method of reducing the number of features in an image is to apply edge detection [14]. For any vision system to work properly, the object to be focused on needs to be positively differentiated from its surroundings. The underlying principle of edge detection is simple enough, as edges can normally be classified as the boundary between two dissimilar regions in an image [14]. Tadrous [73] offers another definition of an edge detector, namely that it aims to identify changes in the image intensity that corresponds to the visual perception of an edge. These edges are normally fairly cheap to compute, and provide strong visual clues to aid in the recognition process. One drawback of the Edge Detection process is that any noise on the image will be clearly shown after the edge detection algorithm has been run; it is therefore recommended that noise reduction be implemented on an image before any edge detection algorithms are executed. Using a threshold is a good way to remove some unnecessary noise.

This works by setting a specific value. Depending on how the threshold is implemented, any generated pixel values that are higher or lower than the threshold will be ignored. Thresholds were used in the RecMaster system to remove unnecessary noise during the edge detection process.

Most edge detection techniques can be broken up into 2 phases [14]:

1. Finding pixels in an image where edges are likely to occur, by looking for discontinuities in gradients. These candidate points for edges are normally called edge points.
2. Linking these edge points in some way to produce descriptions of edges in terms of lines, curves, etc.

These edge points may be regarded as a point in an image where a discontinuity in gradient occurs across some line [14]. Discontinuities may be classified as one of 3 types of the types listed in Table 2 – 5.

Table 2 - 5 Gradient discontinuity types

Discontinuity Type	Description
Gradient Discontinuity	*The gradient of the pixel values changes across a line.*
Jump Discontinuity	*The pixel values change suddenly across some line.*
Bar Discontinuity	*The pixel values increase rapidly and then decrease again across some line, or decreases and then increases again.*

The gradient can be defined as a vector, the components of which measure how rapidly pixel values are changing in terms of distance in the x and y directions. The presence of a gradient discontinuity may be detected by calculating the change in gradient at the specific pixel location. This may be done by first determining the gradient magnitude measure, and then the gradient direction.

These techniques normally involve applying masks to the image on a per-pixel basis, beginning at the top left-hand corner. A general value is then calculated along the x or y axis by using the mask coefficients in a weighted sum of the value of the specific pixel and its neighbours. These masks are generally referred to as convolution masks. Iossifidis, Krathanassi and Rokos [43] describe convolution as an operation that expresses the relation between the input and the output of a linear, time-invariant system. A convolution mask is usually much smaller than the actual image. The mask is therefore moved over the image, manipulating a square of pixels at a time.

There are many different implementations of these masks, each providing basically the same result. Selection of the correct implementation depends on the needs of the specific application.

There are many different edge detection algorithms, each using its own method or mask. These techniques can be divided into two different types as described in Table 2 – 6.

Table 2 - 6 Main edge detection techniques

Edge Detection Technique	Description
First Order / Derivative [62]	*These operators are based on the 1st derivative of the intensity, which provides the intensity gradient of the original data. This information is used to seek out peaks in the intensity gradient. A First-order technique was used in the RecMaster system.*
Second Order / Derivative [62]	*Some other edge-detection operators are based on the 2nd derivative of the intensity, which is obtained by processing the 1st-order results. This is essentially the rate of change in intensity gradient and is best at detecting lines, but implements more processor overhead.*

A few First-order edge detection techniques are presented in Table 2 – 7.

Table 2 - 7 First-order edge detection techniques

Edge Detector	Description
Sobel [32]	*The Sobel operator is typically used on greyscale images to locate the approximate absolute gradient at each point. This is done by performing a 2-D spatial gradient measurement on an image. Sobel uses two 3x3 convolution masks. One mask is used to estimate the x-direction gradient, while the other is used to estimate the gradient in the y-direction. The Sobel masks are shown in Figure 2 – 3.*
Robert's Cross [22]	*This algorithm performs a two-dimensional spatial gradient measurement on an image. It works on the assumption that regions of high spatial frequency often correspond with edges. It is used mostly on greyscale images, along with the masks shown in Figure 2 – 4.*
Prewitt [46] [62]	*Prewitt is an edge detection method that calculates the maximum response of a set of convolution kernels to find the local edge orientation for each pixel. The values for the output orientation image lie between 1 and 8, depending on which of the 8 kernels produced the maximum response.* *This edge detection method is also called edge template matching since a set of edge templates is matched to the image, each representing an edge in a certain orientation. The edge magnitude and orientation of a pixel are then determined by the template that matches the local area of the pixel the best.*

Canny edge detector [17]	*The Canny operator works in multiple stages. The first step is to perform a Gaussian smoothing on the image. Any edges in the image are represented by ridges. The algorithm then tracks along the top of the ridges and sets pixels that are not on the top of these ridges, to zero. This gives a thin line, representing the sought-after edge.*

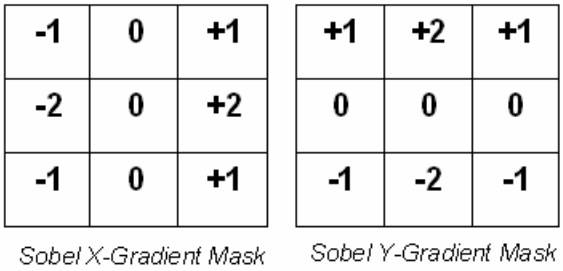

Figure 2 - 3 Sobel masks

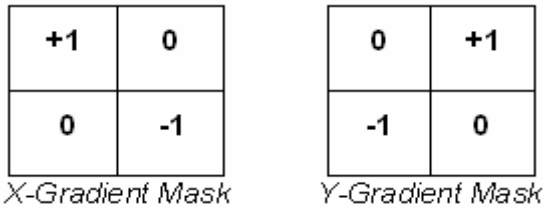

Figure 2 - 4 Robert's Cross masks

Various methods of edge detection were experimented with for the RecMaster system, including column-by-column detection of colour variance as well as a few convolution masks of the programmer's own design. These masks ranged in size as well as implementation (vertical, horizontal, both and tracing the first edge found); the results returned, although actually comparable to those of the other algorithms, were noisy, took too long to finish processing or returned too many edges by not thresholding correctly. Based on the above-mentioned process of trial of error it was eventually decided to use the Sobel algorithm, as it produced the edges best suited to the study in a short enough duration of time. The Sobel algorithm itself also integrated easily into the RecMaster system.

Laplacian of Gaussian [32] is an example of a second-order technique. The Laplacian is a 2-D isotropic measure of the 2nd spatial derivative of an image. It is often used for edge detection since it highlights regions of rapid intensity change in an image, i.e. transitions between background and object. The Laplacian is generally applied to an image once it has been smoothed over by an effect such as a Gaussian filter.

There are also a few other image manipulation operators that are commonly used in conjunction with edge detectors; some of them are described in Table 2 - 8.

Table 2 - 8 Image manipulation operators

Operator	Description
Gaussian smoothing [33]	*This image manipulation smoothes over noisy images by making use of a Two-Dimensional convolution mask.*
Dilation [31]	*Dilation acts on a binary image to gradually enlarge the boundaries of foreground pixel regions. This causes areas of foreground pixels to expand, while holes within those regions become smaller.*
Erosion [32]	*Erosion acts on a binary image to erode away the boundaries of foreground pixel regions. This causes areas of foreground pixels to shrink in size, and any holes inside the specified object to become larger.*

2.2.2.7 Recognition: Classification and post-processing

All processes up to this point have been preparing the data to be used as input for the classification process. During this process, individual segments are compared to predefined classes or models and assigned a value that signifies the likelihood of a match. During the Post-processing phase, a correlation is drawn between individual segments in order to determine the correct classification. In other words, this is the phase in which the actual recognition takes place, as the input model is compared to the stored model.

Recognition is the main and final aim of any vision system. Horn [38] names a few uses for recognition:

1. To move around and safely avoid objects
2. To pick up and place various objects
3. To inspect objects

One way of recognising simple objects is by means of invariants. Bob Bailey describes invariance as the generation of shape features functioning independently of parameters that cannot be controlled in an image [5]. They have the advantage of providing a simple means of comparison, as well as providing both position and orientation information; however, they may prove too simplistic for some applications, as they do not provide a unique means of identification.

Another recognition method makes use of models. These techniques work by first creating a model of the object to be recognised. The model can be created by teaching the software by example. The process of recognition then becomes an Artificial Intelligence matching problem, as the software needs to try and find a match between the stored model and the scene being analysed. There are three broad methods of model-based image recognition (see Table 2 -9).

Table 2 - 9 Methods of model-based image recognition

Method	Description
The Interpretation Tree Search Method	*This implementation makes use of a binary tree, with each node on the tree representing a model primitive, such as a specific edge, surface or texture. The recognition process then takes the form of a search through the tree nodes to find matching primitives [12].*
The Relaxation Labelling Method	*This implementation takes each of the model's primitives and tries to match it to primitives identified in the scene. Each of these primitives is then given a probability (percentage) to indicate the likelihood of a match [20].*
The Graph Searching Method	*Primitives of both the object model and the scene to be recognised are represented as two separate relational graphs. These graphs consist of a set of nodes, which represent object or scene features, and are connected by lines called links or edges. These nodes connect in ways that indicate the relationship between the features they represent. Recognition then becomes a task of matching these 2 graphs to each other [7].*

A proprietary method, called Simple Variable Matching, was used in order to enable the RecMaster system to find a match. Simple Variable Matching basically works in the same manner as the Relaxation Labelling Method, in that certain variables (primitives) are defined for a model. When recognition takes place, the program generates new variables (primitives) for the object on the image and then compares these variables to those stored in the model. The difference between Relaxation Labelling and the Simple Variable Matching method is that the pre-defined primitives, used in simple variable matching, define a range of possible values for the specific primitive. When the new primitive is generated it is simply compared to the range of values stored in the

model. If the new value falls within the range, it triggers a match to that primitive. Thus, there is no percentage-orientated match to each primitive, but rather a true or a false match. If more than 80% of these variables (primitives) match, then the object as a whole is matched to the model.

2.2.2.8 Training

This is actually a process performed separately from the rest of the recognition process; it normally takes place beforehand. It involves the selection of certain features, mostly by a user, and teaching the system to recognise these features. This ensures that, during the classification process, the system has a pattern with which to compare its extracted segments. The training process can be divided into two subtasks: deciding which features to use in describing the concept and deciding how to combine those features [11]. At a practical level, the algorithms employed in the training process need to be easily scalable to domains with many irrelevant features. This means that we need to train the system to a point at which the training examples reach a desired level of accuracy, called the sample complexity. These kinds of algorithms are called induction algorithms. The website LCSWeb defines an induction algorithm as a function that uses an existing classifier to create one or more new classifiers [9]. Induction is based on specific facts (examples) instead of general axioms, as is deduction - in other words, the inductive inference tries to obtain a complete and correct description of a given phenomenon from specific observations of it [4]. The most frequent application of inductive learning is concept learning, which aims to find symbolic descriptions and express them in high-level terms that are understandable by people.

This raises the subject of bias, which Mitchell and Utgoff define as anything influencing the way in which the induction is made [54]. Why bias? The fact that the symbolic descriptions are understandable by people makes it possible for them to add their input to the process, based on their own preferences.

Depending on the user's level of expertise and experience, this input may either boost the system performance or have a detrimental effect. However, all these algorithms have different approaches when it comes to deciding which features to focus on. One example, called the nearest-neighbour method, retrieves the nearest match training model and then classifies test instances by comparing all available attributes. The problem with this approach, however, is that irrelevant training examples severely slow down the learning rate - in fact, the number of training examples needed to reach a given level of accuracy grows exponentially with the number of irrelevant attributes [42].

Another induction method attempts to explicitly select relevant features and reject irrelevant ones by focusing on only a small subset of features. This allows the algorithms to significantly reduce the number of samples under consideration.

One big problem with all of these methods is how to determine which features are relevant to the recognition process, and which are irrelevant. As already stated, the higher the number of irrelevant features present in our training examples, the more examples we will need to accurately train our model. Thus, selecting only the most relevant features is of critical importance. However, relevant feature selection in the RecMaster system will not present as big a problem as in the case of facial recognition or scenery recognition from aerial photographs, for instance. Working on objects running off a conveyor belt from a fixed angle, with fixed lighting, will make the selection of relevant features much more predictable. Blum and Langley [11] suggest that the process of feature selection has four basic issues that need to be dealt with:

1. The starting point for the search must be determined.
2. It must be decided how the search will be organised.
3. The strategy, that will be used to evaluate the strengths of alternative features, must be decided on.

4. It must be determined when the search needs to be halted, either successfully or unsuccessfully.

It is believed that these 4 issues will easily fit into the framework of the proposed system, and will greatly assist in the selection of relevant features. The Recmaster system fits into this framework as follows:

1. The search begins simultaneously at the top and left of the image.
2. The search works simultaneously from top to bottom and left to right, terminating when a perimeter pixel is found.
3. The system always evaluates perimeter, area, the bounding box's X-axis length and the bounding box's Y-axis length. The number of edges on the EdgeGraph is only evaluated if the first 4 values do not provide an adequate match.
4. The search is halted unsuccessfully if the perimeter does not match the minimum search parameters, if only 2 or less of the first 4 values match or if less than 4 values match in total. The search is halted successfully if 4 of the 5 values match.

2.2.3 Obstacles to pattern recognition in image processing

Pattern recognition in images is very resource-intensive, since most pixel-based techniques treat each pixel as a separate random variable. Incorporating prior knowledge (training) reduces this randomness of pixels, but how to incorporate this prior knowledge into the system is not always certain. This prior knowledge can range from object shape, texture and colour to how to best optimise the recognition process. However, most image recognition programmes ignore the prior information of neighbouring pixel correlation, concentrating on the direct extraction of features using distance or error measurements instead. During the process of segmentation and recognition, a large number of dimensions can be

generated very rapidly, which may overwhelm the system - especially when algorithms are being applied on a per pixel basis.

Something else to keep in mind while working on image recognition is: What are the assumptions about the image? In other words, can it be assumed with relative certainty that the specifically sought after object is indeed in the image. With this assumption made, can one use a refined, deformable model to search for it or does one not make any assumptions at all? When making no assumption will one attempt to extract any information possible from the image, hoping it will match up with something in memory?

When one does indeed make an assumption, then the technique used is known as the top-down approach [13]. This approach assumes that the sought after object is actually in the image, so it makes use of a specific deformable model in its interpretation process, thus drastically cutting down on resource usage.

However, when no assumption is made about whether or not the object is in the image, then the technique is known as the bottom-up approach to image interpretation. This involves analysing an image by applying filters and algorithms to obtain basic information that may be used to draw a comparison between information in an on-line knowledge store. This method was not very successful in implementation, as it had to work under the assumption of a small world with a complete world model in which only a limited number of objects was allowed, which could be constrained by tight descriptions [23].

However, the best interpretation method consists of a combination of these two models. This entails doing the basic image recognition in conjunction with assumptions gained from prior knowledge; the information obtained through this process is then compared with the sum total of information stored in the knowledge base in order to make a correct interpretation. The process implemented in the RecMaster system works in this manner, performing image

recognition on an image or frame by looking for a specific minimum-sized object. When one is found, it is matched to the saved model.

The next section will describe how agents can be used to increase the performance of an image processing application through preset task delegation.

2.3 Agent-based computer vision

As discussed earlier, most agent-based problem-solving techniques involve the subdivision of an overall task into smaller subtasks. There are several ways of accomplishing this in actual practice, as described by Rana and Rosin [63]:

One method involves dividing an image into separate subsections and then assigning individual agents to these sections for processing. Once the entire section has been processed the information is not shared with the other agents, but is rather passed along to another system component for evaluation and presentation. This method does not make full use of the benefit of agent-based programming, as no underlying agent communication is involved. Thus, the processing could just as easily have been done using a conventional top-down design approach.

Another method makes use of the inherent hierarchical nature of agents. This method divides the task up into layers. Agents at the bottom-most layer of the hierarchy will work on individual pixels; the next level of agents will govern regions of the image and will use the output from the lower layer as its input. Agents that govern regions may have overlapping fields in their control environment, as this helps to ensure continuity between the regions and may assist in spotting trends. There may be any number of layers, depending on the task at hand, but mostly the top layer will be concerned with taking the

cumulative input from the lower layers and drawing useful conclusions to present to the user.

Yet another method works on the basis of assigning a specific low-level processing task to each agent. These tasks will encompass many of the activities one would normally associate with image processing, such as:

1. Colour detection
2. Hue detection
3. Brightness thresholding
4. Edge detection
5. Pattern recognition

Any one of these agents may be applied to the image at different levels of resolution. In essence, each agent in this kind of implementation basically acts as a filter. This type of image processing could also have been implemented just as easily in a basic top-down design approach, as there is no communication between the individual filter agents. The user may decide which agents are needed for each type of image, but an approach more suited to the agent architecture would be to implement them in a mini-hierarchy. This would involve a single higher-level agent governing the use of all these lower-layer agents – adding at least some level of autonomy and agent communication to this method.

The agents were implemented in a basic top-down approach in the RecMaster system, with each agent acting as a filter, but passing the processed image to the next upon completion of processing. Thus, each agent relies on the previous one for successful performance. The individual agents however communicate via a controller agent, called the Whiteboard agent. Whenever one agent is finished processing the shared object (image), controlled by the Whiteboard, the Whiteboard will grant access to the next agent in line.

There are also different opinions on how the image interpretation process should take place when used in conjunction with an agent-based system. The specific control processes used to maintain control over the image processing task have traditionally been the source of bottlenecks. Bosch, Bovenkamp, Dijkstra and Reiber [13] suggest that a vision agent should have at least the following groups of abilities (see Table 2 – 10).

Table 2 - 10 Vision agent abilities

Agent Ability	Description
Image processing	*This entails object detection, object adjustments and hypostudy testing.*
Communication	*This encompasses all forms of communication between individual agents or groups of agents.*
Bookkeeping	*This involves the storage, retrieval and removal of image objects.*
Conflict Resolution	*Agents should be able to resolve any differences in interpretation amongst themselves.*

The agents in the RecMaster system all perform some form of image processing task, and they communicate their findings to one another. According to Bosch, Bovenkamp, Dijkstra and Reiber [13], the first control issue that needs to be dealt with is: what should an agent do, given its knowledge state and the abilities it possesses? They have chosen to implement a model in which these tasks are dealt with at what is called the system control level. This level is very abstract, and is hard-wired into their system's architecture. What this model basically entails is that each agent in their architecture works according to a specific cycle of events, e.g. the first step would consist of reading input, then it elaborates on its current working memory state, decides what to do next, fires specific rules related to the situation and finally produces output. After the output is produced, the whole sequence is repeated again. Although this may sound like a very rigid and specific task list to be used in an autonomous system, it is a very efficient

approach since individual agents still work on their own tasks and then communicate their findings to the next agent so that the cycle can continue. Even in an agent-based system, some measure of control is necessary to ensure that the processing spectrum for an individual agent is not too vague for it to fully and successfully accomplish its task.

This section is concluded by giving short descriptions of applications that have successfully married the concepts of agents and image recognition (see Table 2 - 11).

Table 2 - 11 Applications that combine agents with image recognition

Application	Description
iJADE Surveillant	*iJade is a truly intelligent multi-resolution neuro-agent based automatic surveillance system* [48].
OURVS	*OURVS is an agent-based vision system for controlling robots competing in the RoboCup Soccer Tournament* [18].
VIBRA	*VIBRA is a Multi-Agent Architecture for visually guided robotic tasks. It integrates visual perception, planning, reaction and execution in order to solve complex tasks* [8].
MORE	*The objective of this system is to recognise multiple objects in a single image of a real-world scene, including complex occlusions* [25].

The next section is a brief discussion on threading.

2.4 Threading

The desktop computer industry has been dominated of late by chipsets manufactured by two major companies. One of these companies is called Intel. Intel has been a pioneering force in desktop computing from 1979 [61], when it introduced its x86 instruction set. Many of the current chipset architectural

technologies and advances were first available in Intel chips. Another company named American Micro-Devices (AMD) started off playing catch-up to Intel, but has - in recent years - matched Intel's performance and even surpassed the giant in some ways. A lesser known fact is that AMD has been in contention with Intel for a very long time. AMD started off making clones of the Intel chipsets, some of which - such as the 386DX-40 CPU - topped Intel's 486SX chip in terms of speed, performance and cost [75]. Two of the areas in which AMD excel have made their chips very attractive, the first being that their chips can handle more instructions per clock cycle, and the other - which is of great interest to consumers - being the affordability of their chips.

This study will not focus on each and every chipset nuance, but rather give a description of a technology that has become more and more prevalent in recent years, and has been incorporated into the RecMaster system that was developed. This technology is called multi-threading. Although multi-threading is incorporated differently in the Intel and AMD chipsets, it can be abstractly referenced with the same high-level code in both cases, without the programmer having to explicitly manipulate low-level code for each of the different implementations.

But what is multi-threading? In the past, programme performance was improved by splitting a programme into multiple streams called threads. These threads were then queued and sent to multiple processors. The new Intel and AMD technologies enable programmes to run multiple streams or threads on a single processor. This facilitates a higher level of parallelism, which results in improved processor performance and better utilisation of system resources. [3]

The reason for this performance increase is that, since a lot of current desktop software actually needs to multitask for optimal performance, the programmers have already built threading into the software. However, these programmes could never perform at their best, as multi-processor systems are expensive and

mostly out of the average consumer's price range. Multi-threading offers a solution by allowing a single-processor system to behave more like a multi-processor system.

These threads are actually perfect for implementing an agent-based system, or even just a system in which more than one image needs to be processed simultaneously. Whereas the system would previously have needed to wait for one line of processing to complete before another begins, it can now depend on the operating system and processor to schedule the different processes to be performed in a round-robin fashion, thus dramatically improving recognition.

The combination of multi-agent systems, efficient image processing algorithms and threading make a desktop image recognition system much more viable than it would have been a few years ago. It brings affordable, high-end desktop image processing to the consumer market.

The next chapter discusses the development of the RecMaster system.

Chapter 3

3. PROGRAMME DEVELOPMENT

The culmination of this study was the development of an image recognition system called RecMaster. This chapter outlines key areas of the development process. The design and code behind any of the features discussed in this chapter are available in Appendix A and Appendix B (available in the Windows Start Menu after program installation). All coding of the RecMaster system was done in Microsoft Visual C# 2003.

3.1 Edge detection

In any image recognition system the processes used during the image processing phase are the most important, as there would be no system without these processes. The development process was therefore initiated by seeking the best way to process images. The process that was used was selected on the basis of literature study and the attendance of a workshop on image recognition, presented by Dr. Herbert Frey [29] from the Fachhochschule Ulm (University of Applied Sciences) in Germany.

The first step was to find the edges on any image the system was presented with. Some time was spent developing systems for edge detection. These systems were mostly based on convolution masks of differing sizes and weights, but some of them functioned by analysing an image column-by-column and row-by-row in order to find jumps in colour values. These systems all worked, to some extent. Some were fast, but did not provide adequate edges; others were slower, but returned viable edges. Some actually worked as required, providing fast performance and viable edges, but the problem was that a means of setting thresholds had not been provided. These systems returned an excess of edge data, and running a second dilation filter (see 2.2.2.6 Feature extraction and

edge detection) on the image would add unnecessary overhead. In the end, through literature studies the Sobel [50] edge detection algorithm was identified as the most suitable. This algorithm is based on a convolution mask that finds the median value of a 3 x 3 matrix of pixels and then sets the centre pixel to this value, depending on its relation to the threshold value. The mask is applied to an image from left to right and top to bottom. A snippet of the source code is shown in Figure 3 - 1. A fully explained version of the entire source code is available as Appendix B in the Windows Start Menu, after program installation.

```
int stride = bmData.Stride;
System.IntPtr Scan0 = bmData.Scan0;
System.IntPtr Scan02 = bmData2.Scan0;
unsafe
{
        byte * p = (byte *)(void *)Scan0;
        byte * p2 = (byte *)(void *)Scan02;
        int Offset = stride - picInUse.Width*3;
        int Width = picInUse.Width * 3;
        int Pixel = 0;
        int PixelMax = 0;
        p += stride;
        p2 += stride;
        for(int y=1;y<picInUse.Height-1;++y)
        {
                p += 3;
                p2 += 3;
                for(int x=3; x < Width-3; ++x )
                {
                        PixelMax = Math.Abs((p2 - stride + 3)[0] - (p2+stride-
                                3)[0]);
                        Pixel = Math.Abs((p2 + stride + 3)[0] - (p2 - stride –
                                3)[0]);
                        if (Pixel>PixelMax)
                           PixelMax = Pixel;
                        Pixel = Math.Abs((p2 - stride)[0] - (p2 + stride)[0]);
                        if (Pixel>PixelMax)
                           PixelMax = Pixel;
                           Pixel = Math.Abs((p2+3)[0] - (p2 - 3)[0]);
                        if (Pixel>PixelMax)
                           PixelMax = Pixel;
                        If (PixelMax < Threshold)
                           PixelMax = 0;
                         p[0] = (byte) PixelMax;
                        ++ p;
                        ++ p2;
                }
        p += 3 + Offset;
        p2 += 3 + Offset;
        }
}
```

Figure 3 - 1 Edge detection code snippet

3.2 Colour inversion

The edges returned by the Sobel algorithm are sufficient for gaining information about the object on the image. The only problem with the algorithm is that it returns an image that consists of a dark background, with the edges indicated in the original colour of the image. To make the objects on the image stand out a bit more, it was necessary to invert the colours of the image. This would make the dark background light and the coloured edges dark. The process behind this is quite simple. The images used in this system are all 24-bit images, meaning that 3 bytes are used to represent each pixel. Each of these bytes respectively represents the pixel's Red, Green or Blue value. Each byte of the image was analysed individually, with every 3 bytes representing a pixel. The highest value that can be represented in a byte is 255, so the byte's inverse value was determined by subtracting the current byte value from 255. The source code in Figure 3 - 2 demonstrates the colour inversion process.

```
int stride = bmData.Stride;
System.IntPtr Scan0 = bmData.Scan0;
unsafe
{
        byte * p = (byte *)(void *)Scan0;
        int Offset = stride - picInUse.Width * 3;
        int Width = picInUse.Width * 3;
        for(int y = 0;y < picInUse.Height ;++y)
        {
                for(int x = 0; x < Width; ++x )
                {
                        p[0] = (byte)(255-p[0]);
                        ++p;
                }
                p += Offset;
        }
}
```

Figure 3 - 2 Colour inversion code snippet

3.3 Converting the image to binary

This study ignores any colour values present in an object, as they would add extra overhead during the recognition phase and make it more difficult to trace the perimeter in the next step of the process. It was therefore decided to convert the image to binary. This means that any light colours will be converted to white, and any dark colours will be converted to black. This is done by going through the image pixel by pixel, and adding up the 3 byte values of each pixel. This byte total is then divided by 3, and compared to a threshold value. If the value is smaller than the threshold all 3 bytes are set to zero, giving the pixel a black colour. If the value is higher than or equal to the threshold all 3 bytes are set to 255, giving the pixel a white colour. A snippet of the binary conversion algorithm is shown in Figure 3 - 3.

```
int stride = bmData.Stride;
System.IntPtr Scan0 = bmData.Scan0;
unsafe
{
        byte * p = (byte *)(void *)Scan0;
        int Offset = stride - picInUse.Width*3;
        byte red, green, blue;
        byte binary;
        for(int y=0;y < picInUse.Height;++y)
        {
                for(int x=0; x < picInUse.Width; ++x )
                {
                        blue = p[0];
                        green = p[1];
                        red = p[2];
                        binary = (byte) ((blue + green + red) * 0.333);
                        if (binary < Threshold)
                           p[0] = p[1] = p[2] = 0;
                        else
                           p[0] = p[1] = p[2] = 255;
                        p += 3;
                }
                p += Offset;
        }
}
```

Figure 3 - 3 Binary conversion code snippet

3.4 Measuring the object

Because the scope of this project is very limited, the algorithms implemented are designed to deal with a uniform background environment with an object in the foreground, just as on a conveyor belt. This was by no means ever intended to be a general-purpose image recognition system. Based on the assumption that we are working on a uniform background with an object in the foreground, the filters that were run on the image thus far should leave us with a white background and our object's edges indicated in black. The next step would be to actually take measurements of the object. These measurements could be stored during the training phase and used as comparison during the recognition phase. The first noticeable difference between this filter and the implementation of the previous filters lies in the way the image is processed. The other filters work by manipulating the image line-by-line, byte-by-byte. This filter, however, works on a per pixel basis, treating the image more as a two-dimensional array. Originally the filter was implemented by working with the bytes, as this would be a more low-level and ultimately timesaving approach (for the processor). However, the recursive method employed in this filter lends itself better to the grid-based technique. At the moment, this filter takes the longest of all to complete all its functions.

The first task was to determine in which direction the programme would be searching on the image; i.e. how it would find the object the fastest. To this end, 6 threads were created; 3 of them running over the image from top to bottom with regular spaces between them, and 3 of them running over the image from left to right, also with regular spaces between them. These threads all have the same purpose, namely to find the first black pixel. The first one to find it will change the value of a shared variable, so that the rest will all stop their search as well. It will also log the coordinates of the pixel's location. An agent could have been created to perform this task, but trial-and-error proved that running 6 functions on independent threads proved more resource friendly.

This pixel marks the beginning of our search for the object's perimeter. Since no dilation algorithms were run on the image, the edges indicated may be a few pixels thick. This algorithm's first task is therefore to change this so that only the outermost layer of object pixels remains. As it does so, it also counts the number of pixels in the perimeter. The process of finding the right algorithm in order to correctly determine the perimeter took up a considerable amount of time. Eventually, the following solution was found.

The first step in determining the perimeter is to find the pixel following the first pixel. To keep the search organised, the mask shown in Figure 3 – 4 is used. Searches are always conducted in an anti-clockwise direction using the numbers 1 – 8 in Figure 3 – 4 as a reference.

2	1	8
3	First/Current Pixel	7
4	5	6

Figure 3 - 4 Pixel map

For the first pixel, the search for the next pixel begins at value 2 and moves anti-clockwise to 1. The mask value, on which the next pixel is found, is saved and the coordinates of the current pixel are set to the coordinates of the found pixel. This is used to remember the current direction. The value of the perimeter counter is also incremented from 1. This same mask is then applied for the next pixel, but the value of First Pixel is now substituted with that of Current Pixel.

The direction that was saved during the search for the second pixel is now used to guide the search for the next one in line. In other words, if the second pixel was found on value 3, then the search for the next pixel will begin at value 3 and move anti-clockwise to value 2. For each new pixel found, the perimeter value is incremented. This process will continue until the entire perimeter has been mapped out. During this process, the highest and lowest x-axis and y-axis values of the object will also be saved.

After the perimeter has been mapped out, the image is scanned again to find the surface area of the object. This is done by running through the image from the lowest x-axis value of the object, to the highest. For every x-axis value, the image is run through along its y-axis. The first and last perimeter pixel's coordinates are recorded and subtracted from one another, giving the number of pixels inside the object along the y-axis. The surface area of the object is obtained by adding up all these y-axis difference values.

The highest and lowest x- and y-axis values represent the object's bounding box. A bounding box is an imaginary 4-sided box that can be drawn around the object to a perfect fit. Subtracting the highest and lowest x-axis values from each other yields the x-axis length of the bounding box, and subtracting the highest and lowest y-axis values from each other yields the y-axis height of the bounding box. The algorithms involved in creating these values are too long to show here; please consult Appendix B (available after program installation) for a complete source code listing.

All these calculations leave us with four measured values:

1. Perimeter
2. Area
3. Length of bounding box X-axis
4. Height of bounding box Y-axis

3.5 Creating the EdgeGraph

The previous algorithm provided us with 4 values that could be stored and used in comparison to perform recognition. However, sometimes these 4 values are not enough to provide a positive match. To this end one more value was added, which would provide the original number of edges on the object before only the perimeter was extracted. These edges are counted along the centre of the object, so a bounding box is needed for accurate measurement.

This algorithm analyses the pixels along the object's central y-axis to determine which pixels are black and which ones are white. This information is used to create a binary string of the central y-axis, with 1's representing black and 0's representing white. The string is then analysed to determine the number of times that strings of 1's are present. Each of these strings of 1's represents an edge, and is counted. This information is also used to draw an EdgeGraph.

The EdgeGraph is basically a graph that depicts the spatial relationship between the edges of the object. It uses the 0's of the string to increase the spaces on the graph, and the 1's to indicate how high a specific edge must be drawn.

Figure 3 – 5 shows the outlines of a circular object with the central Y-axis indicated in red and the corresponding EdgeGraph drawn from this Y-axis. The lines in blue indicate the correspondence of the first half of the EdgeGraph to the edges of the circular object.

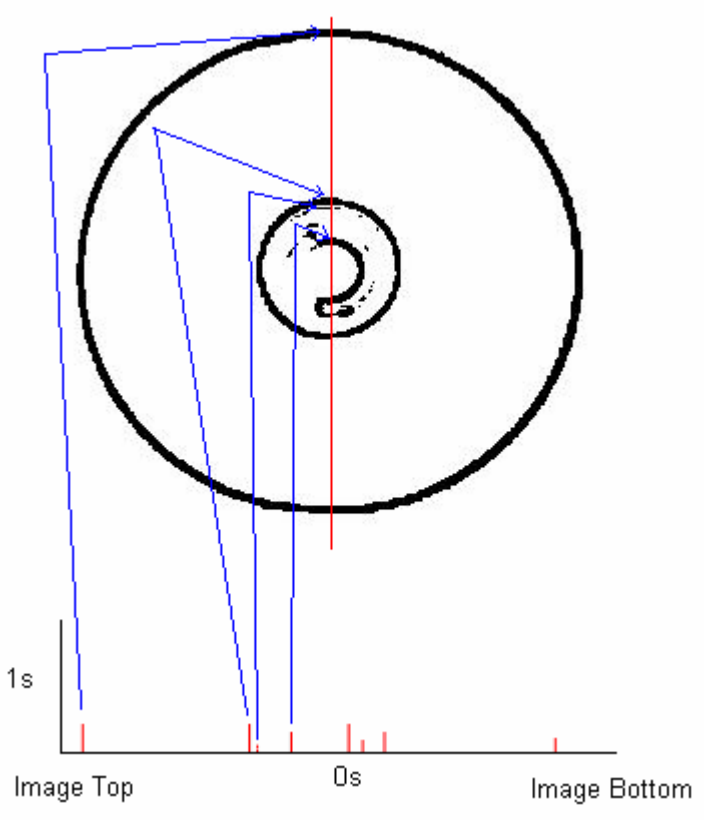

Figure 3 - 5 EdgeGraph example

3.6 The blackboard

The aforementioned processes of edge detection, colour inversion, image binarisation, object measurement and EdgeGraph creation are performed by individual agents. Each agent is responsible for handling one step in the overall process. These agents use a shared blackboard system to communicate with one another. The blackboard system consists of an object of a class named Whiteboard. Each of the agents has a connection to this Whiteboard object, which allows them to see the same variables. Some of these variables indicate the current state in the processing sequence so that only a certain agent is allowed to operate on an image at a given time, depending on the value of the variables. The variables are incremented or decremented by individual agents as processing occurs. This creates a very simple form of communication

between the agents, allowing them to inform other agents when they are finished with their part of the processing. This process is shown in Figure 3 – 6.

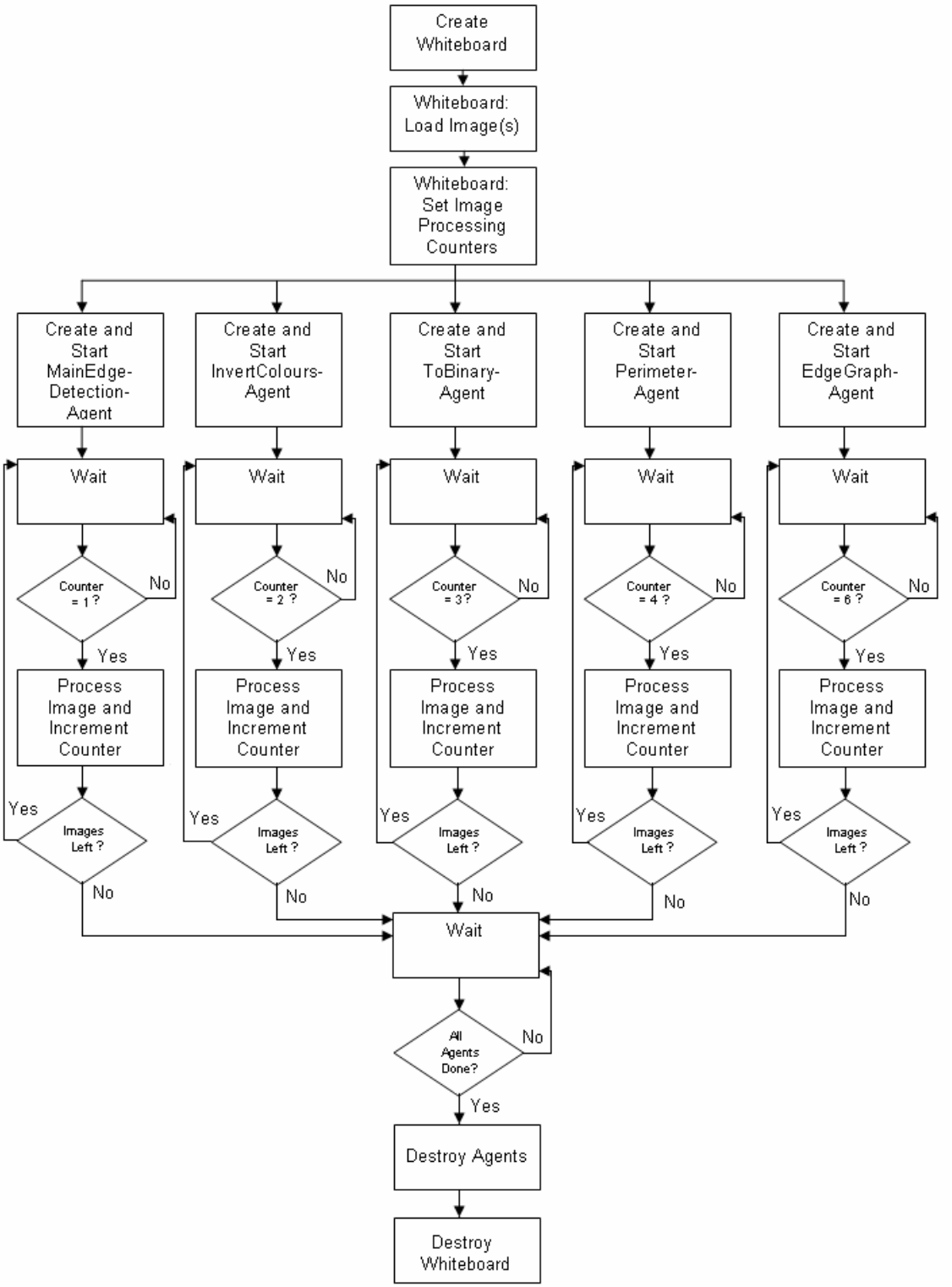

Figure 3 - 6 Agent Creation and Communication

3.7 Creating the agents

The blackboard system provides individual agents with a means of letting others know when they have completed their processing tasks. However, the programmer still needed to figure out how the agents were actually going to be created and destroyed, as well as how to incorporate the different algorithms into them. To this end, 5 agents were created to bear the brunt of the image processing work. The agents and their functions are listed in Table 3 – 1.

Table 3 - 1 Program agents

Agent	Task
MainEdgeDetectionAgent	*Edge Detection.*
InvertColoursAgent	*Invert Image Colours.*
ToBinaryAgent	*Create a binary (two-colour) image.*
PerimeterAgent	*Calculate the perimeter, area and bounding box of the object.*
EdgeGraphAgent	*Draw the EdgeGraph and determine the number of median edges.*

Initially the agents were activated whenever a new subset of the program was initiated, e.g. the Recognition phase of the programme. All the agents were therefore created and loaded onto individual threads, where they would then just sit and wait for something to happen. Although the waiting phase is not resource-heavy, it does consume resources to some extent. Consequently it was finally decided that the agents and threads would be created at the beginning of each phase of the programme, e.g. Recognition, but would only be set to their waiting state whenever a specific part of a phase (e.g. Video Mode) was activated. Whenever the specific activity ends, the threads are stopped and the agents are taken out of their waiting state. If the agents are needed again, their waiting states are restarted and the threads activated again. Figure 3 - 7 is an example of how agents are set to a waiting state, activated and then set to rest after their function has been completed.

```
// Creates the shared Whiteboard object. Then initializes all of the
// agents and loads them onto individual threads.
whiteboard = new Whiteboard(bitmap);
MEDagent = new MainEdgeDetectionAgent(640,480,whiteboard);
MEDthread = new Thread(new ThreadStart(MEDagent.StartWait)); MEDthread.Start();
ICagent = new InvertColoursAgent(640,480,whiteboard);
ICthread = new Thread(new ThreadStart(ICagent.StartWait)); ICthread.Start();
TBagent = new ToBinaryAgent(640,480,whiteboard);
TBthread = new Thread(new ThreadStart(TBagent.StartWait)); TBthread.Start();
Pagent = new PerimeterAgent(whiteboard,5,5,635,475);
Pthread = new Thread(new ThreadStart(Pagent.StartWait)); Pthread.Start();
EGagent = new EdgeGraphAgent(whiteboard,5,435,5,635);
EGthread = new Thread(new ThreadStart(EGagent.StartWait)); EGthread.Start();
// Tells the first agent to start processing.
Whiteboard.Increment();
StillRecognitionForCamVideo();
// Ends the agents' waiting process and stops the agents.
whiteboard.RunStartWait = 1;
MEDthread.Join();
Cthread.Join();
TBthread.Join();
Pthread.Join();
EGthread.Join();
```

Figure 3 - 7 Whiteboard agent initialization code snippet

In the end, 2 additional classes were also created to help present data to the user. These classes (as listed in Table 3 – 2) are only used in the Update model phase, and only when the user wants to see more information about an image processed in Still Image Mode.

Table 3 - 2 Additional RecMaster classes

Class	Task
RemoveOutsideBackground	*Sets the area outside the object's bounding box to white.*
Overlay	*Traces the edges located by the MainEdgeDetectionAgent and the perimeter located by the PerimeterAgent onto a greyscale version of the original image.*

The RemoveOutsideBackground class works simply by running through all of the pixels of an image and then setting all the values outside the object's bounding box (as located by the PerimeterAgent) to white.

The Overlay class processes a copy of the original image by first converting it to greyscale and then tracing over the greyscale image with the data created by the

MainEdgeDetectionAgent and the PerimeterAgent. All tracing is done by setting the intended pixels to red.

Greyscaling the image is a simple matter of adding up the Red, Green and Blue values of each pixel. This new value is then divided by 3 to determine the average. This average is then assigned to the Red, Green and Blue values of the pixel, overwriting the original values and recreating the colour of the specific image as a median colour. This is done for every pixel on the image. A snippet of the process is shown in Figure 3 - 8.

```
int stride = bmData.Stride;
System.IntPtr Scan0 = bmData.Scan0;
unsafe
{
        byte * p = (byte *)(void *)Scan0;
        int Offset = stride - picInUse.Width*3;
        byte red, green, blue;
        for(int y=0;y<picInUse.Height;++y)
        {
                for(int x=0; x < picInUse.Width; ++x )
                {
                        blue = p[0];
                        green = p[1];
                        red = p[2];
                        p[0] = p[1] = p[2] = (byte) ((blue + green + red) * 0.333);
                        p += 3;
                }
                p += Offset;
        }
}
```

Figure 3 - 8 Greyscaling code snippet

3.8 Creating the model

The next step in the development process was to create a way for the values to be recorded from different sources, and then to save the values to a file. To this end, several options were created for the system to gather the required information. There are four different modes in which the information can be gathered:

1. Using a single still image
2. Using multiple still images, all being processed by different threads
3. Using frames from a video file
4. Using frames from a camera or other device feed

The processes behind these different modes are discussed in Appendixes A and B. The processing of these various modes makes it possible to build up a model by recording the following values:

1. Perimeter
2. Area
3. Length of the bounding box X-axis
4. Height of the bounding box Y-axis
5. Number of edges on the EdgeGraph

The measurements of each new processed image or frame are used to adjust and fine-tune the range of values for the model. In the model creation and updating process, all 5 values always need to be calculated. The only problem is that - as this project's scope is so limited - it is necessary to always use the same two-dimensional view of the object, or else the model would become bloated and useless. Separate models therefore need to be created for the top, sides and bottom of an object.

3.9 Performing recognition

The creation of a model gave the programme a frame of reference within which to perform recognition. The next step was to use the model to find the object in an image. This is also done in three different modes, using the exact same technique that was used in creating the model. These three modes are:

1. Using a single still image
2. Using frames from a video file
3. Using frames from a camera or other device feed

The first 4 values are always calculated and compared to those stored in the model. Each of these individual values needs to fall within the range of values stored in the model file for that specific aspect of the model. If the measured value falls within the specified range, then this value triggers a match to the aspect. If only three of these four comparative values are matched then the fifth value (number of edges on the EdgeGraph) is also calculated and compared to try and find a total of four individual aspect matches. If less than four of the aspects trigger a match, then the object is not recognized as fitting the model. The fault tolerance of the model depends entirely on the quality and accuracy of the images used to build the model. The greater number of accurate images used in the model's construction, the greater the chances become of an accurate match to an aspect. The reverse is also true: if the model is fed inaccurate date, then the matching process will also become inaccurate. The process is explained in greater detail in Appendixes A and B.

The recognition phase makes it possible to use more than one model at a time, so that different views of an object can be taken into account.

3.10 Fleshing out the programme

With the main functions of the programme completed, it was decided to add extra functionality to the programme in the form of still image capture, as well as video capture from a camera (device) feed. These functions are explained in Appendixes A and B.

3.11 Conveying real-world measurements

If the programme was to be useful in a quality control environment, some way had to be found to map on-screen coordinates to real-world coordinates on the conveyor belt. To achieve this, a section was added in which real-world x-axis and y-axis measurements (in millimetres) of the device feed view could be entered into the system, along with the speed and direction of the conveyor belt on which the system is operating.

The speed and direction are very important. Since it can take up to 300 milliseconds for recognition to be completed in extreme cases, the object might not be exactly where it was originally when the recognition process began. To accommodate this, a calculation needs to be done to determine the object's present position on the conveyor. This is important, since the eventual end goal of the system is to provide an external robotic arm with the coordinates of an object on the conveyor, so that it can be picked up. At the moment, the values are calculated but not passed to an external source, as this is not within the scope of the study. (Another version of the RecMaster system - compiled in Visual Studio 2005 - has been partially completed, and does allow for the passing of variables via a serial link.)

This chapter explained the theory behind most of the system's main issues. More detailed technical specifications regarding the design and explanations of the source code can be found in Appendixes A and B (available in the Start Menu after program installation). The next chapter discusses the results of benchmarking the RecMaster system.

Chapter 4

4. BENCHMARKS

The completed RecMaster system needed to be evaluated for performance. This chapter presents the findings of benchmarking the system on both an AMD and an Intel system. These two different architectures were used in order to determine which system would best suit the RecMaster programme, and to prove that both of them are, in fact, capable of running the RecMaster system. The specifications of the AMD- and Intel-based systems used in this study are indicated in Table 4 -1.

Table 4 - 1 Specifications of test systems

Component	AMD	Intel
Processor	*Athlon64 3200+*	*Pentium IV 3.2 GHz*
Memory	*1 GB RAM*	*1 GB RAM*
Hard Drive	*120 GB Hard Drive*	*120 GB Hard Drive*
Video Card	*GeForce 6800GS 256 MB*	*GeForce 6800GS 256 MB*

The data used in this benchmarking study has been obtained from a logging system written into the RecMaster system itself. All logged events are written to a text file, and then exported to Microsoft Excel for comparative study. Listed in Table 4 - 2 is a list of descriptors that are included in the logging process, together with possible values:

Table 4 - 2 Logged values

Field Name	Possible Values
Operating Mode	*Still, Video or Camera*
Sub-operating Mode	*Whole, Area, Single Image or Continuous*
Filename	*Path of selected file, void if in camera mode*
Current Date	*Format: mm/dd/yyyy e.g. 04/24/2006*
Current Time	*Format: hh:mm:ss AM/PM e.g. 9:19:46 AM*
Object Found	*Y(es) or N(o)*
Overall Match Found	*Y(es) or N(o)*
Recognition Time	*Time in milliseconds from process beginning to end.*
Informative Message	*Informs the user of the percentage match found, and to which model.*
Object Perimeter Value	*Value in number of pixels*
Object Area Value	*Value in number of pixels*

Object Bounding Box X-axis Length	*Value in number of pixels*
Object Bounding Box Y-axis Length	*Value in number of pixels*
Number of Edges on the Object's Edgegraph	*Value in number of pixels (if any)*
Match Found on Perimeter	*Y(es) or N(o)*
Match Found on Area	*Y(es) or N(o)*
Match Found on Bounding Box X-Axis	*Y(es) or N(o)*
Match Found on Bounding Box Y-Axis	*Y(es) or N(o)*
Match Found on Number of Edges	*Y(es) or N(o)*
Time Stamp of Video File (Video Mode only)	*Displays the current playing time of the video file when the frame was processed.*

4.1 Still images

4.1.1 Creating the model

The first step in benchmarking the RecMaster system was to create a model to use during the recognition process. The chosen object from which a model was created was a blank DVD, of which an example is shown in Figure 4.1. The model was thus also named DVD. All of the images and videos used in benchmarking the system or creating models can be found in RecMaster's install directory. To create the model, fifteen 640 x 480 pixel images of a DVD were taken with a camera attached to the system. These images were then analysed using whole image mode in the Update model process. The model's specifications are shown in Table 4 – 3.

Table 4 - 3 Still image mode model specifications

Match Field	Lowest Value	Highest Value
Perimeter	*713*	*764*
Area	*48102*	*53611*
Bounding Box X-Axis Length	*251*	*264*
Bounding Box Y-Axis Length	*242*	*257*
Edges on EdgeGraph	*6*	*8*

The RecMaster system saves time by only inspecting objects with a perimeter that is at least 75% of the length of the lowest Perimeter Range value of the chosen model. In this case the value was calculated as follows:

713 * 75 / 100 = 534.75

If the perimeter of any object found by the RecMaster system does not have a value of at least 535, then the object will not be investigated further. The reasoning behind the 75% cut-off value is that if the object's perimeter is too far below the required perimeter match value, the chance of succeeding in matching the object using only the remaining match values is too slight. Thus, the cut-off value conserves valuable processing resources.

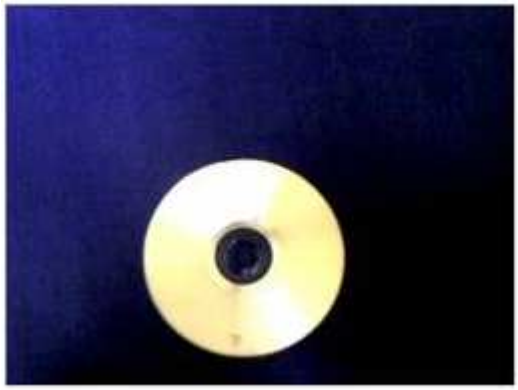

Figure 4 - 1 DVD model image

4.1.2 Choosing comparative images

In order to test the accuracy of the model, it needs to be determined whether the system can recognise a DVD object in an image. To this end, a list of images was selected in which the system was to search for DVD objects:

Table 4 - 4 Test image types

Image Group	Example
15 images of a DVD	*Figure 4 - 2*
15 images of a DVD Spindle	*Figure 4 - 3*
15 images of a Blender Top	*Figure 4 - 4*
15 images of a Grey Bottle Cap	*Figure 4 - 5*
15 images of a Canned Fruit Bottle Cap	*Figure 4 - 6*
15 images of a Playing Card	*Figure 4 - 7*
15 images of a Remote Control	*Figure 4 - 8*

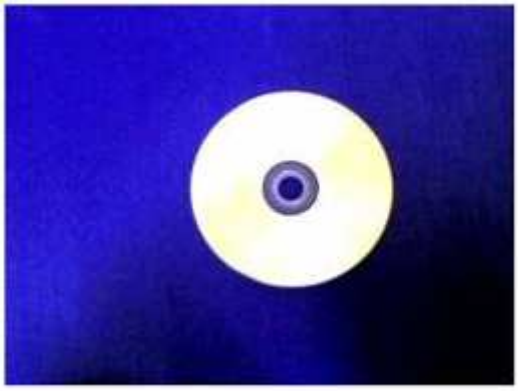

Figure 4 - 2 Example of DVD test image

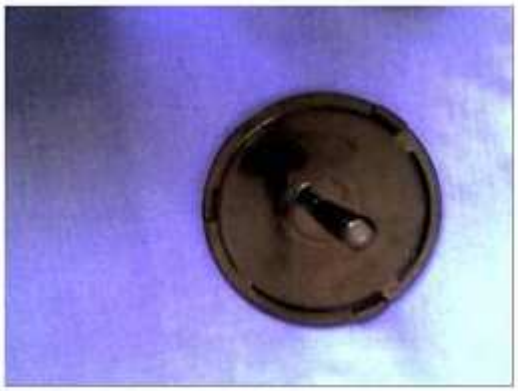

Figure 4 - 3 Example of DVD Spindle test image

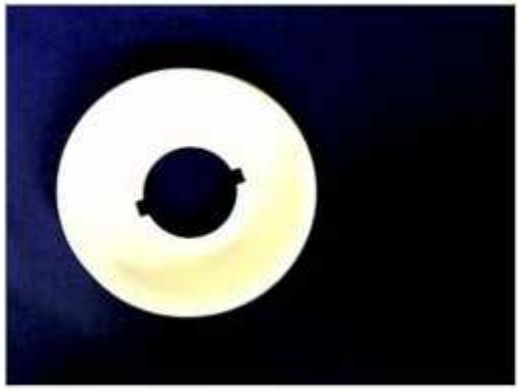

Figure 4 - 4 Example of Blender test image

Figure 4 - 5 Example of a Grey Bottle Cap test image

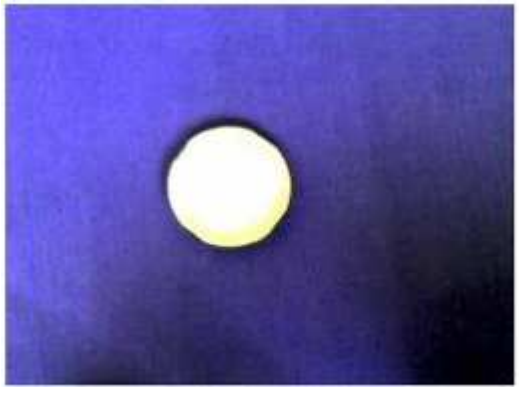

Figure 4 - 6 Example of a Canned Fruit Bottle Cap test image

Figure 4 - 7 Example of a Playing Card test image

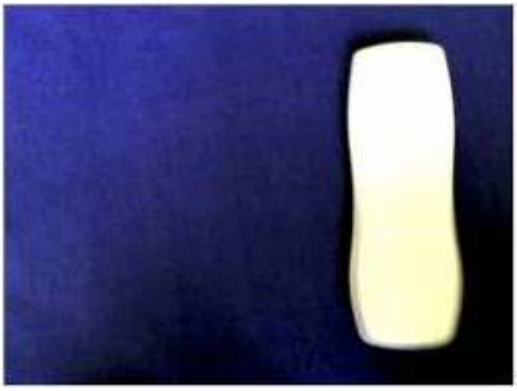

Figure 4 - 8 Example of Remote Control test image

Groups of 15 images were chosen, because it gives the benchmarks a larger scope than a simple decimal sampling of ten images. It was not deemed necessary to use more than 15 images, as the consistency of the results shown in Appendix C seemed to indicate that the system would yield similar results no matter how large the test sampling.

The groups of images were not selected randomly. The first set of images, which contains a DVD object, is used to determine whether the model will actually recognise a DVD object.

The sets of images containing DVD Spindles and Blender Tops were chosen because these objects have the same general size and shape as a DVD, so they are used to test whether the model will be able to distinguish between a DVD and a closely related object.

The sets of images containing the Grey and Canned Fruit Bottle Caps were chosen because these objects have the same general shape as a DVD, but a different size.

The last sets of images containing the Playing Cards and Remote Controls were chosen because their shape and size do not correspond to those of a DVD.

All the objects were photographed using the same web cam, against a uniform blue background. This is important, as the system has been tuned to function on a uniform background with the object in the foreground. The background colour of the images may vary slightly, since the web cam automatically compensates for differing light conditions.

The results obtained by running each of these groups of images through the Perform Recognition process in Still Image mode of the RecMaster system, are indicated below.

4.1.3 Objects found in images

The first step in determining whether the system is performing to expectation was to ascertain in how many of the images an object was actually found.

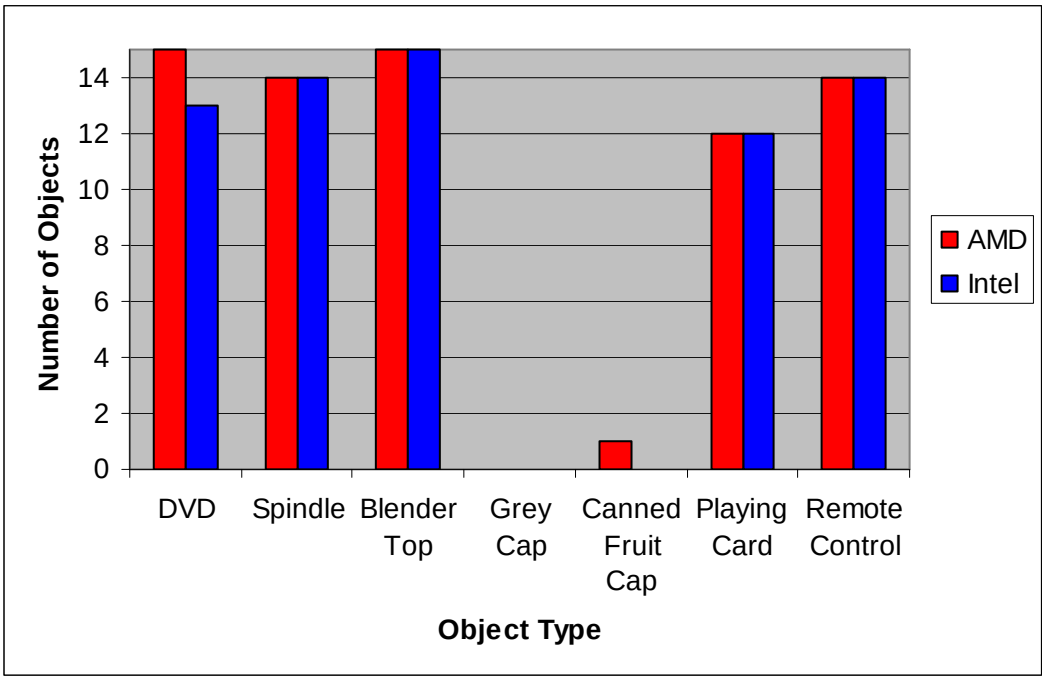

Figure 4 - 9 Objects Found graph

Figure 4 - 9 indicates the results shown in Table 4 – 5.

Table 4 - 5 Results of the Objects Found graph

Object Name	Images	Objects found		Success Rate (%)	
		AMD	Intel	AMD	Intel
DVD	15	15	13	100	87
DVD Spindle	15	14	14	93	93
Blender Top	15	15	15	100	100
Grey Cap	15	0	0	0	0
Canned Fruit Cap	15	1	0	7	0
Playing Card	15	12	12	80	80
Remote Control	15	14	14	93	93

All results are rounded off to the nearest whole value. The findings on the AMD and the Intel system were relatively consistent, with differing results only on the DVD and Canned Fruit Cap images. On the DVD images the AMD system had a 13% higher success rate than the Intel-based system, while on the Canned Fruit Cap images the AMD system had a 7% higher success rate at finding objects in images. These results seem to indicate that the AMD system is more effective in running the Perform Recognition process, but a final conclusion will only be reached in this regard once more results have been obtained. Another aspect of Figure 4.9 that requires attention is the low success rate obtained with the Grey Cap and Canned Fruit Cap images. The individual benchmarking results for the Grey Cap (4.1.5.4) and the Canned Fruit Bottle Cap (4.1.5.5) shed some light on these poor results.

4.1.4 Matches found on objects

The test in section 4.1.3 indicated that the system is indeed capable of finding objects on images - but how accurate are these findings, and can the objects actually be matched to a model? The next set of results seems to confirm that the system does have these capabilities.

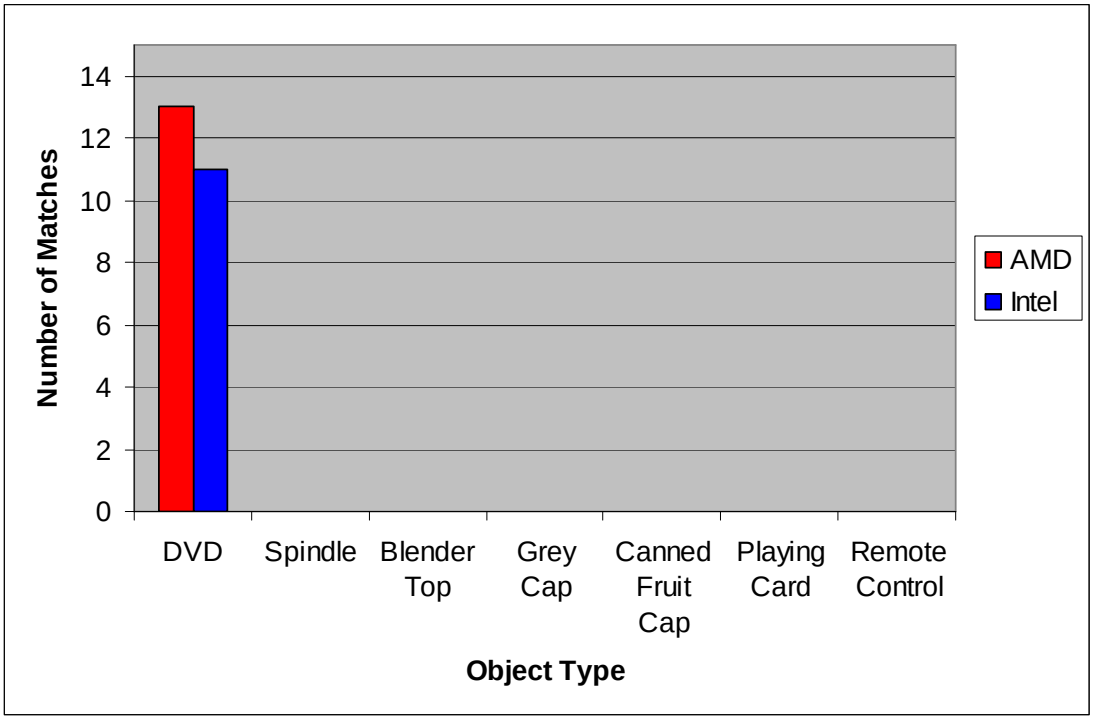

Figure 4 - 10 Matches Found graph

Figure 4 - 10 indicates the results shown in Table 4 – 6.

Table 4 - 6 Results of the Matches Found graph

Object Name	Objects Found		Matches Made		Success Rate (%)		Total Success Rate (% out of 15)	
	AMD	Intel	AMD	Intel	AMD	Intel	AMD	Intel
DVD	15	13	13	11	87	85	87	73
DVD Spindle	14	14	0	0	0	0	0	0
Blender Top	15	15	0	0	0	0	0	0
Grey Cap	0	0	0	0	0	0	0	0
Canned Fruit Cap	1	0	0	0	0	0	0	0
Playing Card	12	12	0	0	0	0	0	0
Remote Control	14	14	0	0	0	0	0	0

All results are rounded off to the nearest whole value. The findings indicate that matches were only made to objects found in the images that actually contain

DVD objects. This proves that the RecMaster system was not fooled by the Spindles or the Blender Tops, which have similar sizes and shapes to that of a DVD. None of the other objects could fool the system either. It is interesting to note that the AMD-based system had an 87% success rate, both for the number of matches made to objects found and the overall match percentage of the number of images. The Intel-based system had an 85% success rate for the number of matches made to objects found, which is only 2% less than the AMD-based system; however, it obtained a 73% success rate for the overall match percentage of the number of images. This is 14% less than the AMD-based system. Why would this difference exist if the same images were used for testing both systems? Figure 4 - 11 is a representation of the processing threads traversing an image as they search for a perimeter pixel.

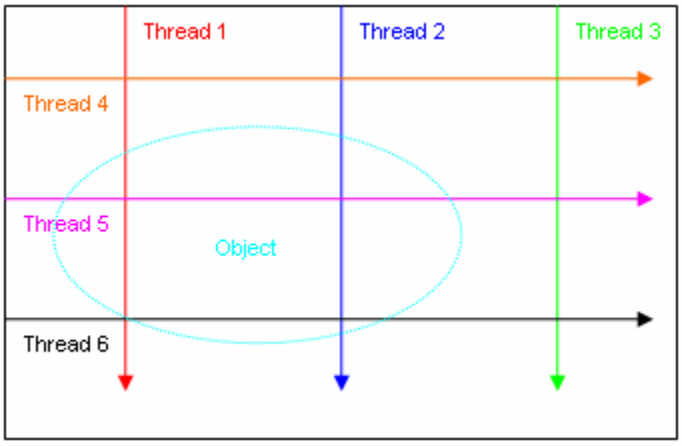

Figure 4 - 11 Thread movements on an image

Threads 1 – 6 run simultaneously. Whenever one of them locates a perimeter pixel, all the other threads stop processing. Since the AMD and Intel systems do not operate at exactly the same speed and inherently do not schedule their threads in the same way, it may occur that one system locates a different start pixel. Even though noise has been reduced as much as possible, it may occur that a chosen start pixel is noise-related, and not located on the targeted object. This would then yield an unlocated object.

Figures 4.9 and 4.10 prove that the RecMaster system is capable of distinguishing different objects with the aid of the created model. This proves that the system is capable of performing the required image recognition task – at least in still image mode. It also seems to indicate that the AMD-based system is inherently better at both finding objects and matching them to the chosen model.

Next, the results of each of the different image types were studied to determine how the system arrived at its findings, as well as how rapidly these findings were extracted, i.e. how fast the RecMaster system's Perform Recognition process performance is in Still Image mode.

4.1.5 Individual image types

The first thing the system does when determining a match is to investigate the perimeter of the located object. If the perimeter is below the calculated cut-off value, which in this case is 535, processing on the image will be stopped and the system will specify that no image has been found – this conserves valuable processing resources. If the perimeter is above or equal to the cut-off value, the rest of the object's values will be calculated and matched to the model. As soon as all aspects have been calculated, the system will determine how long this entire process has taken.

4.1.5.1 DVD

Figure 4 - 12 DVD Perimeter graph

Figure 4 - 12 clearly indicates that the system did not find a perimeter above 535 for Images 10 and 11 on the Intel-based system. This corresponds to the results on Figure 4 - 9, which indicated that the AMD-based system found 15 objects, and the Intel-based system only 13.

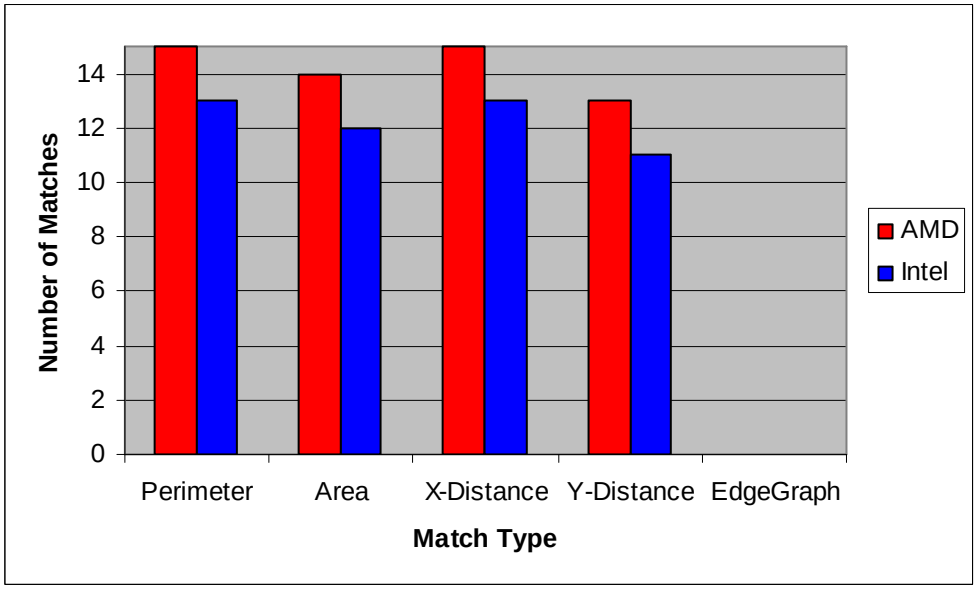

Figure 4 - 13 DVD Property Matches graph

Figure 4 - 13 indicates that both the AMD- and Intel-based systems only relied on the first 4 object characteristics to secure a match to the DVD model. No matches were made on the EdgeGraph value - in fact, the data indicates that the EdgeGraph value was only calculated for the second DVD image, and not for any other images in the entire test; nevertheless, no matches were obtained from it. It seems that perimeter is the match value that is used most often. The RecMaster system relied on perimeter to try and form a match in 15 of the objects on the AMD system and 13 of the objects on the Intel system, which corresponds directly to the number of objects found.

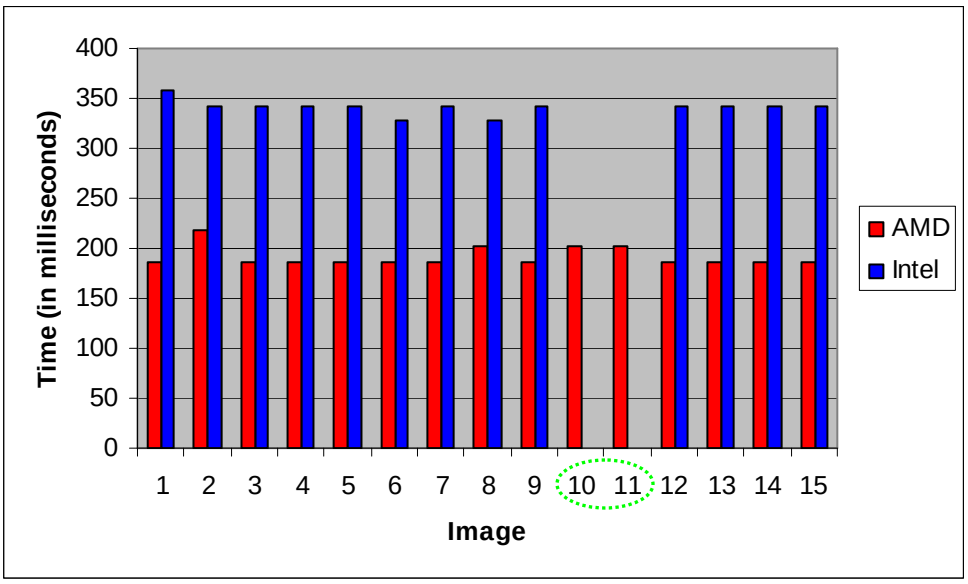

Figure 4 - 14 DVD Recognition Speed graph

Figure 4 - 14 displays the time that it took (in milliseconds) for each object to be fully processed, whether it ended up as a match to the DVD model or not. The graph again clearly shows that no objects were found by the Intel-based system in Images 10 and 11. It also shows that, for all the images, the AMD-based system was much faster in completing the recognition process.

4.1.5.2 DVD spindle

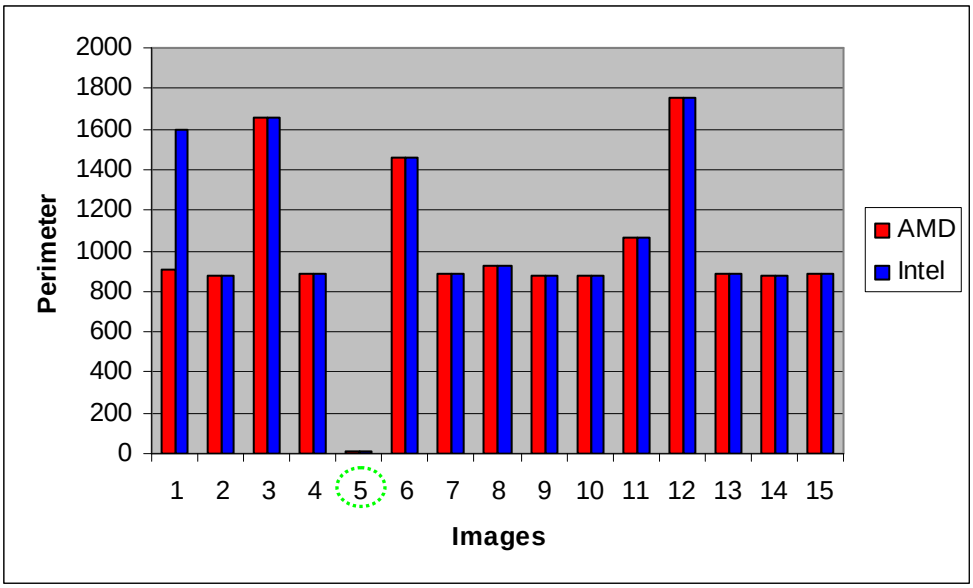

Figure 4 - 15 DVD Spindle Perimeter graph

Figure 4 - 15 reflects the results shown in figure 4 - 9, which indicated that there was only one DVD spindle image on both systems in which an object could not be found. This seems to be Image 5, which is the only image with a perimeter value lying below the cut-off value of 535.

Analysis of the data has shown that none of the objects found have any values matching those of the DVD model. This implies that, even though the size and shape of the object was similar to that of a DVD, the RecMaster system was not fooled.

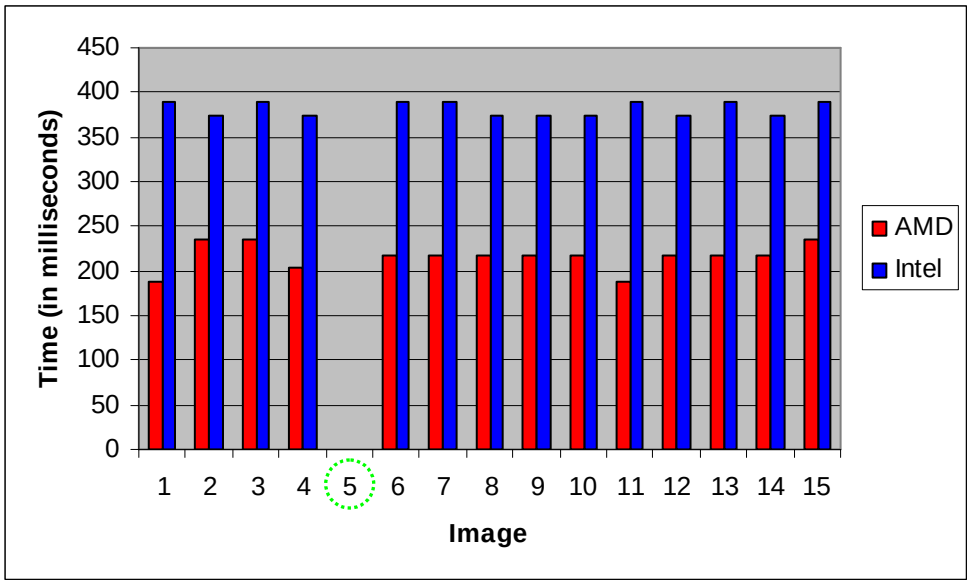

Figure 4 - 16 DVD Spindle Recognition Speed graph

Figure 4 - 16 again clearly shows that neither system found any objects in Image 5. It also shows that, for all the objects, the AMD-based system was much faster in completing the recognition process.

4.1.5.3 Blender top

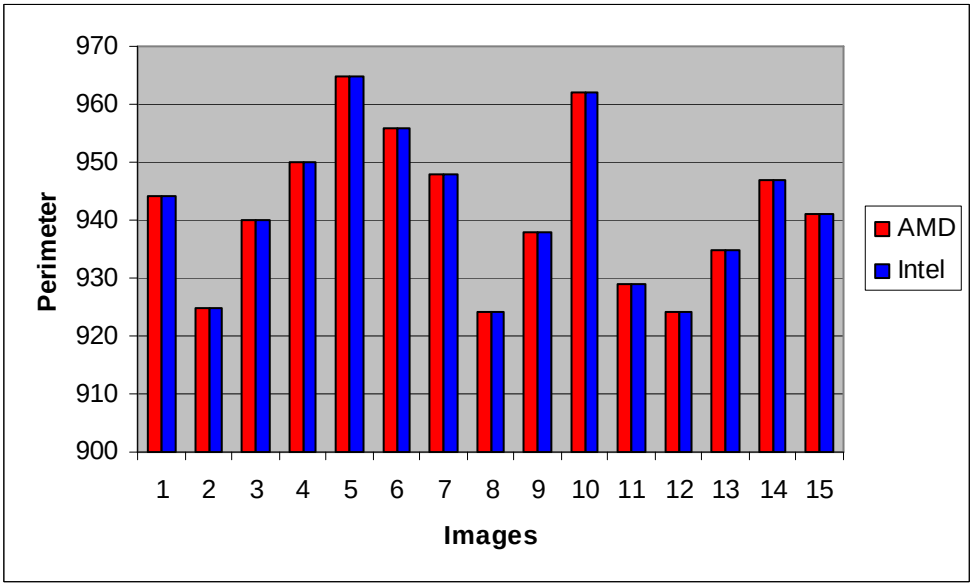

Figure 4 - 17 Blender Top Perimeter graph

Figure 4 - 17 clearly shows that objects were found in each of the images by both systems. It also shows that there was a variance of only 40 pixels between the lowest and highest perimeters, which makes the results very consistent.

Analysis of the data has shown that none of the objects found have any values matching those of the DVD model. This implies that, even though the size and shape of the object is similar to that of a DVD, the RecMaster system was not fooled.

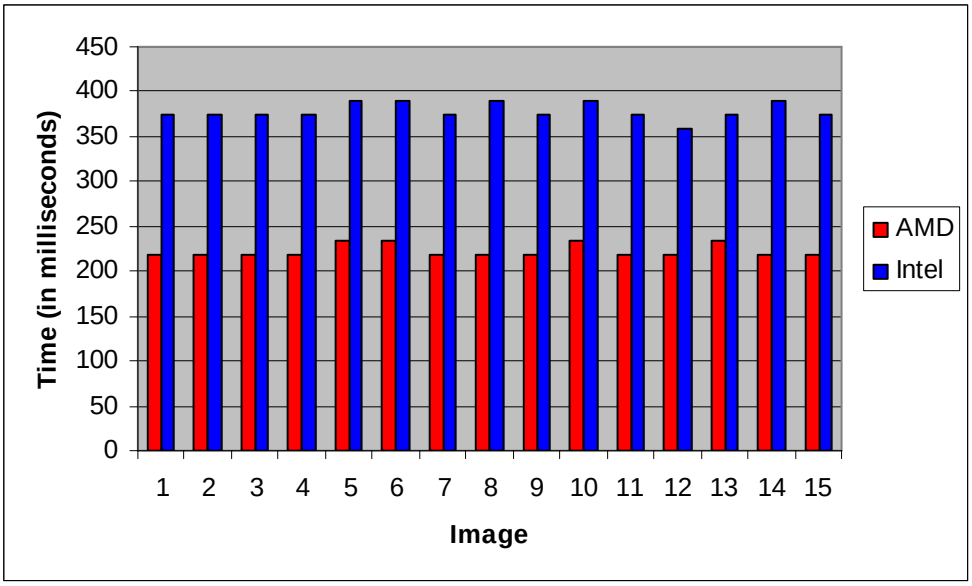

Figure 4 - 18 Blender Top Recognition Speed graph

Figure 4 - 18 again confirms that an object was found in each of the images by both systems. It also follows the same trend as in the previous groups of images, in which the AMD system outperforms the Intel system on all counts.

4.1.5.4 Grey cap

Figure 4 - 19 explains the results of Figure 4 - 9, which indicates that no Grey Cap objects were found for either system. The system did, in fact, find evidence

of objects on all but Image 14 for AMD and Intel, but all the perimeters of these objects were below 535, which is the cut-off value. Consequently, the RecMaster system immediately marked all these images as containing no objects.

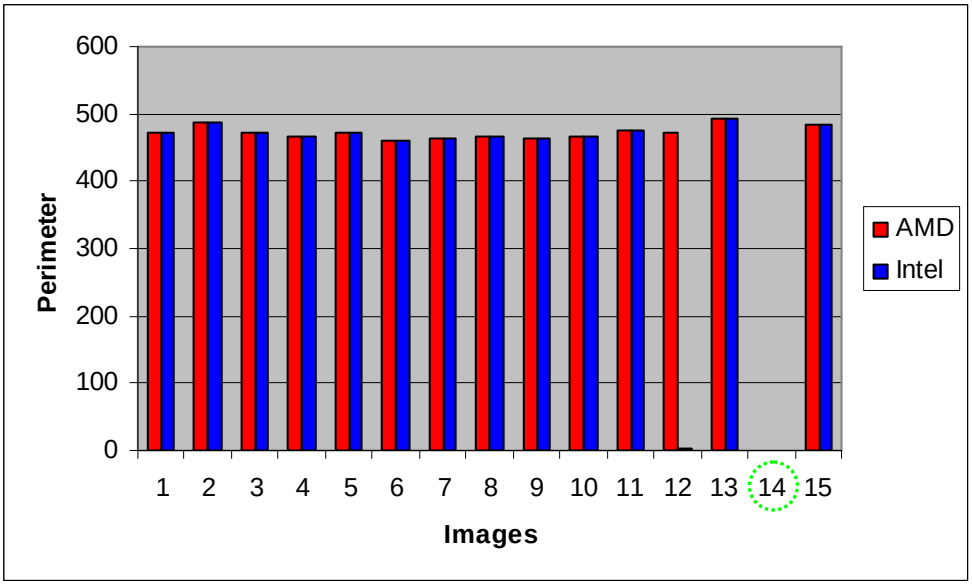

Figure 4 - 19 Grey Cap Perimeter graph

4.1.5.5 Canned fruit cap

Figure 4 - 20 explains the results of Figure 4 - 9, which indicated that no Canned Fruit Cap objects were found on the Intel system, and only 1 was found on the AMD system. Again, the system did - in fact - find evidence of objects on all but Image 8 for both systems, but all the perimeters of these objects - except for Image 1 on the AMD system - were below 535, which is the cut-off value. Consequently, the RecMaster system immediately marked all these images as containing no objects. The object found in Image 1 took 171 milliseconds to process.

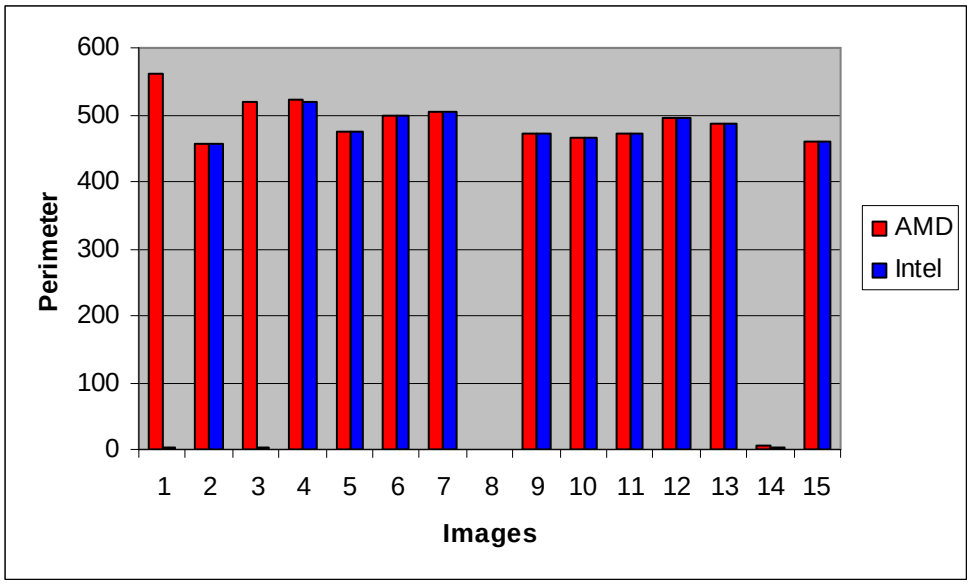

Figure 4 - 20 Canned Fruit Cap Perimeter graph

4.1.5.6 Playing card

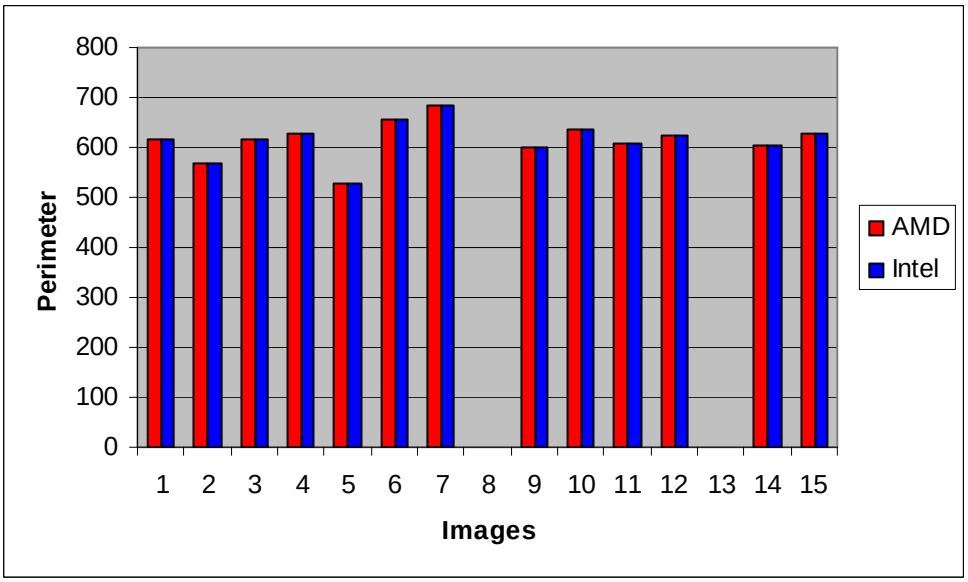

Figure 4 - 21 Playing Card Perimeter graph

Figure 4 - 21 indicates that objects were found in all the images, except for Images 8 and 13. An object was found in Image 5, but its perimeter only had a value of 529, which is below the cut-off value. Thus, only 12 objects in total were found in the 15 playing card images.

None of the characteristics of the playing card objects triggered any matches on the DVD model.

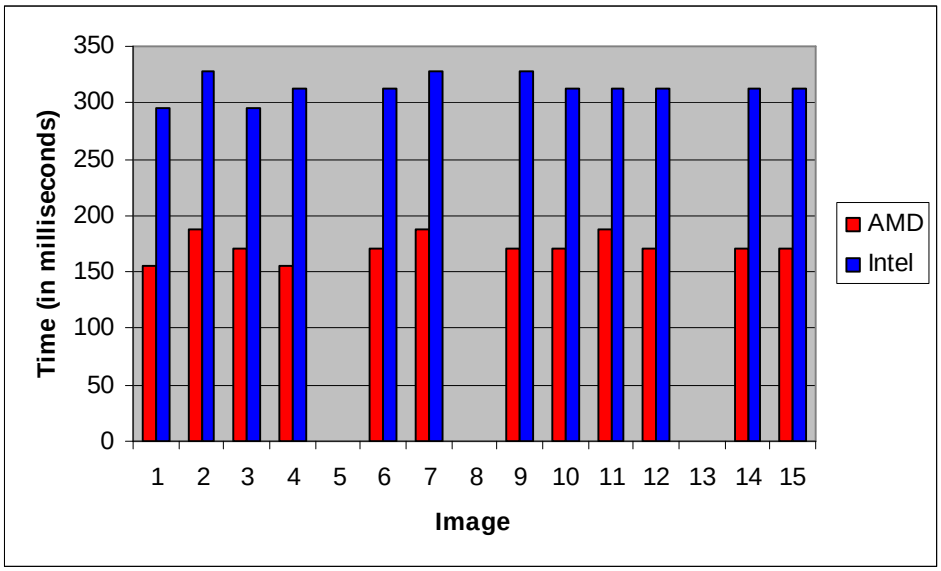

Figure 4 - 22 Playing Card Recognition Speed graph

Figure 4 - 22 clearly supports the findings of Figures 4 - 9 and 4 - 21, which indicate that only 12 objects were found in the 15 images. Again, the results show that the AMD system performed much faster than the Intel system.

4.1.5.7 Remote control

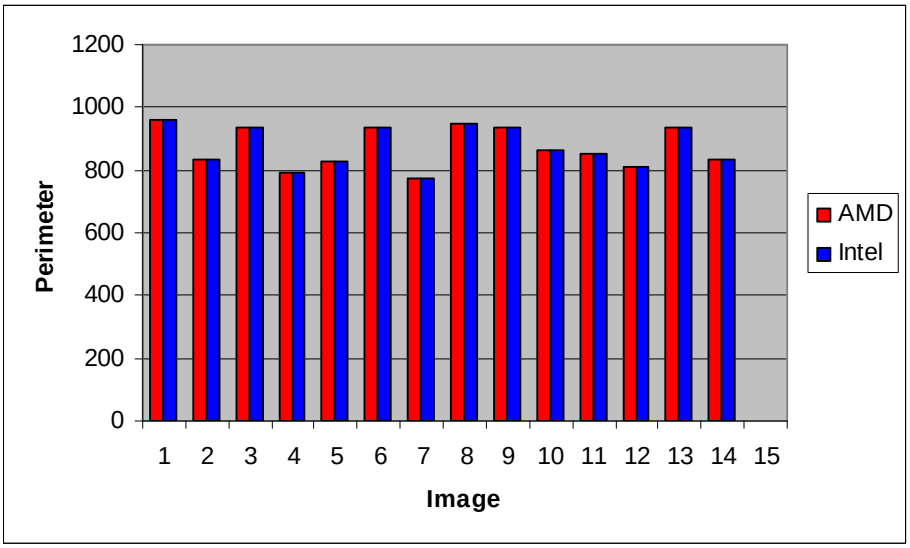

Figure 4 - 23 Remote Control Perimeter graph

Figure 4 - 23 indicates that objects were found in all the images except for Image 15, thus supporting the results shown in Figure 4 - 9.

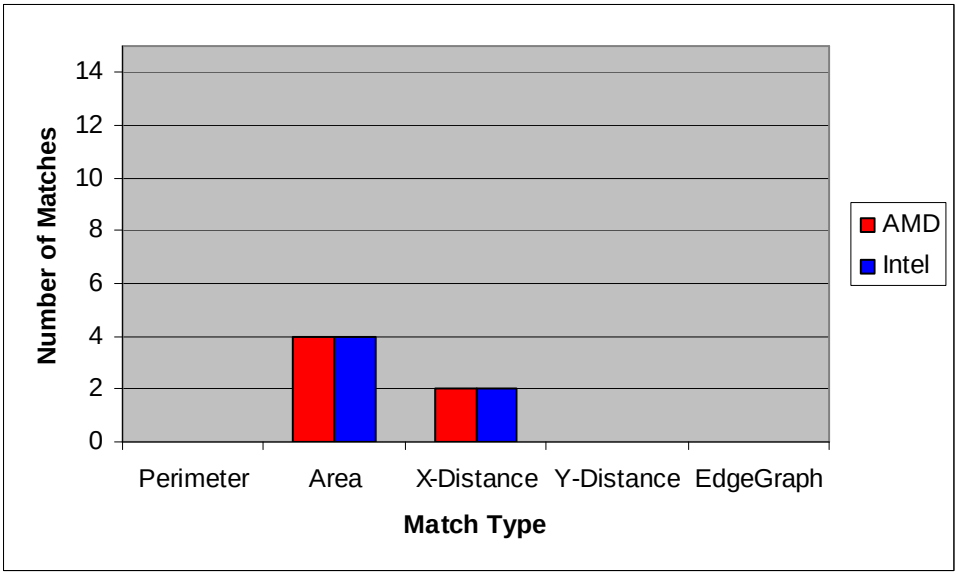

Figure 4 - 24 Remote Control Property Matches graph

As Figure 4 - 24 indicates, the Remote Control objects did indeed trigger some matches to the DVD model's characteristics. Matching Area characteristics were indicated in 4 objects, and matching X-axis Distance characteristics in another 2.

However, none of these characteristics were adequate to trigger a complete match of the DVD model. Thus, the model's integrity remains intact.

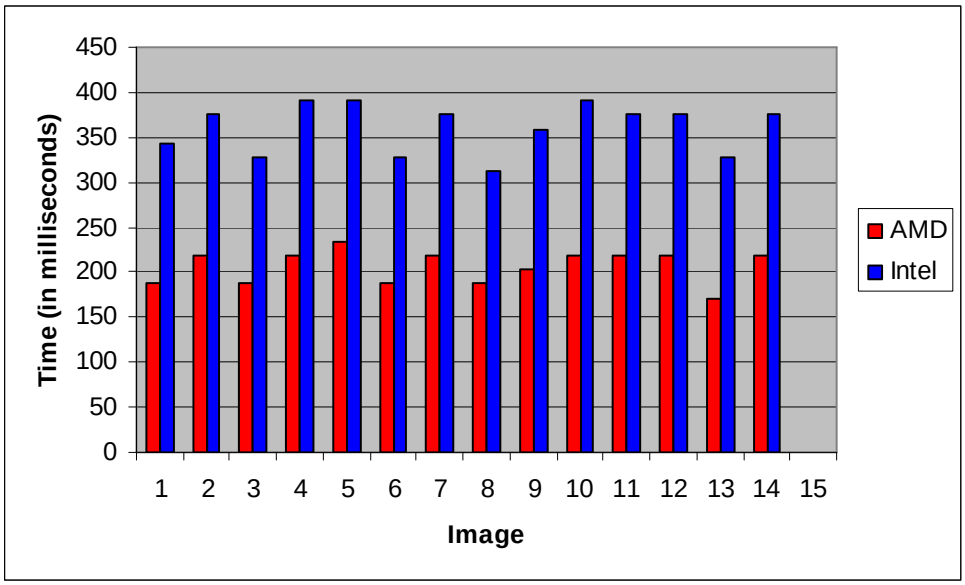

Figure 4 - 25 Remote Control Recognition Speed graph

Figure 4 - 25 clearly supports the findings of Figures 4 - 9 and 4 - 23, which indicate that only 14 objects were found in the 15 images. Again, the results show that the AMD system performed much faster than the Intel system.

4.1.6 Total recognition time

All the results have indicated that the RecMaster system is indeed capable of correctly performing recognition using a predetermined model. The speeds at which these recognition processes are performed differ from one image type to the next, as well as between the 2 different system architectures.

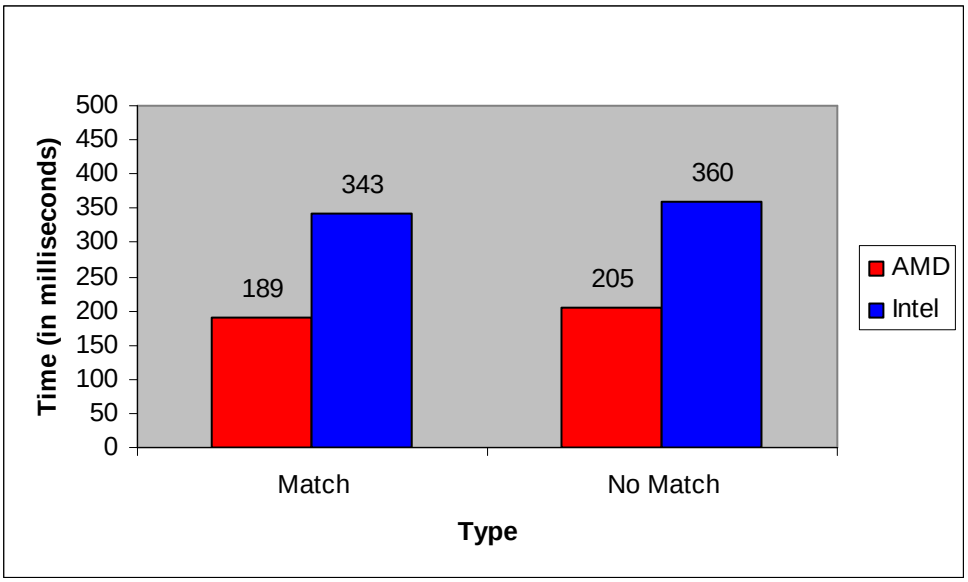

Figure 4 - 26 Average SI Recognition Speed graph

Figure 4 - 26 shows the average speed of both systems with regard to processing an object. The average speed is calculated on the basis of all the images processed in this benchmark, and is calculated both for objects that were matched, as well as for those that were not matched. The following facts have been derived from Figure 4 – 26:

1. On the AMD system, it takes 16 milliseconds longer on average for the system to process an unmatched object than it does to process a matched one.
2. On the Intel system, it takes 17 milliseconds longer on average for the system to process an unmatched object than it does to process a matched one.
3. The closeness of these two differences signifies that the time difference between the processing of matched and unmatched objects is relatively constant.
4. The Intel system takes about 81% longer on average to find an object match than the AMD system does.

5. The Intel system takes about 76% longer on average to find an unmatched object than the AMD system does.

These results seem to indicate that the AMD-based system is much more suitable for running the RecMaster system than the Intel-based system. In defence of the Intel system, however, it must be conceded that the RecMaster system was developed and tested on an AMD system, so it may just be slightly more tweaked towards AMD architecture. Another influencing factor may be different background processes running on either of the two systems. Although all unnecessary processes were halted before conducting the benchmarks, one can never be entirely sure that this possibility was ruled out.

4.2 Video Mode

4.2.1 Creating the model

All the benchmarks that were conducted in single whole image mode have confirmed that the system is capable of correctly performing image recognition on a chosen model. The next step is to determine whether the system can duplicate these results on a continuous input feed, i.e. a camera feed or a video file.

The image processing technique used on the continuous input feed is the exact same as the one used on single images. The system works by capturing frames from the given input feed, and then processing them as still images. Because it has already been proved that the system works in still image mode, the tests done on the continuous feed are not intended to prove this again. Rather, these benchmarks will operate on the assumption that the system does indeed work and, as such, will only endeavour to prove that the system can duplicate the previous results on a continuous feed. To do this, it only needs to be proved that

the system can indeed recognise a DVD in a continuous feed, and will ignore any objects that do not fit the model.

To this end, the tests have been simplified somewhat. The system will be provided with 5 input feeds, each consisting of a video file. The details of these video files are listed in Table 4 – 7.

Table 4 - 7 Test video types

Video Name	Object(s) on Video	Times Object(s) Appears
DVD	*DVD*	*15*
Blender	*Blender Top*	*15*
Canned Fruit Cap	*Canned Fruit Cap*	*15*
Playing Card	*Playing Card*	*15*
3-object Combo	*DVD, Blender Top and Playing Card*	*Each object appears 5 times = 15*

The reasons for the four chosen object types are listed below:

1. The DVD is to make sure that the model can find the intended object.
2. The Blender Top is to make sure that the model can still distinguish the DVD from an object of similar shape and size.
3. The Canned Fruit Bottle Cap is to make sure that the model can still distinguish the DVD from an object of similar shape.
4. The Playing Card acts as a control, since it has a totally different shape and size than that of the DVD.

The 5 feeds were again benchmarked on both an AMD and an Intel system. To guarantee consistency of the results, it had to be ensured that both systems would receive the same input. To this end, it was decided to use a video feed instead of a direct camera feed. During the course of developing the RecMaster system, the results from processing a camera feed and a video feed were deemed comparable. The results should therefore be a good indication of the system's capability.

Since the height of the camera in relation to the objects is not the same as for the still images, a new DVD model - named DVD_Continuous - had to be created for the continuous feed. This model was built up by running the RecMaster system in Camera Update Mode. In this mode, the system was set to only accept the specifications when the object's perimeter was between 500 and 650 pixels – these numbers were arrived at through a process of trial and error. The new model's specifications are liste in Table 4 – 8.

Table 4 - 8 Video mode model specifications

Match Field	Lowest Value	Highest Value
Perimeter	*542*	*617*
Area	*22595*	*33598*
Bounding Box X-Axis Length	*135*	*216*
Bounding Box Y-Axis Length	*186*	*193*
Edges on EdgeGraph	*2*	*5*

This means that the minimum perimeter cut-off value that will allow an object to qualify is calculated as follows:

542 x 75 / 100 = 407

Anything found on a captured frame with a perimeter above or equal to 407, will qualify as an object.

4.2.2 DVD

The first step in determining whether the model could be used to identify objects in continuous mode, is to test it on a video file that is known to actually contain DVD objects. To this end, a video file of 15 DVD's was created, and run on both the AMD and Intel machines in Continuous Recognition Mode.

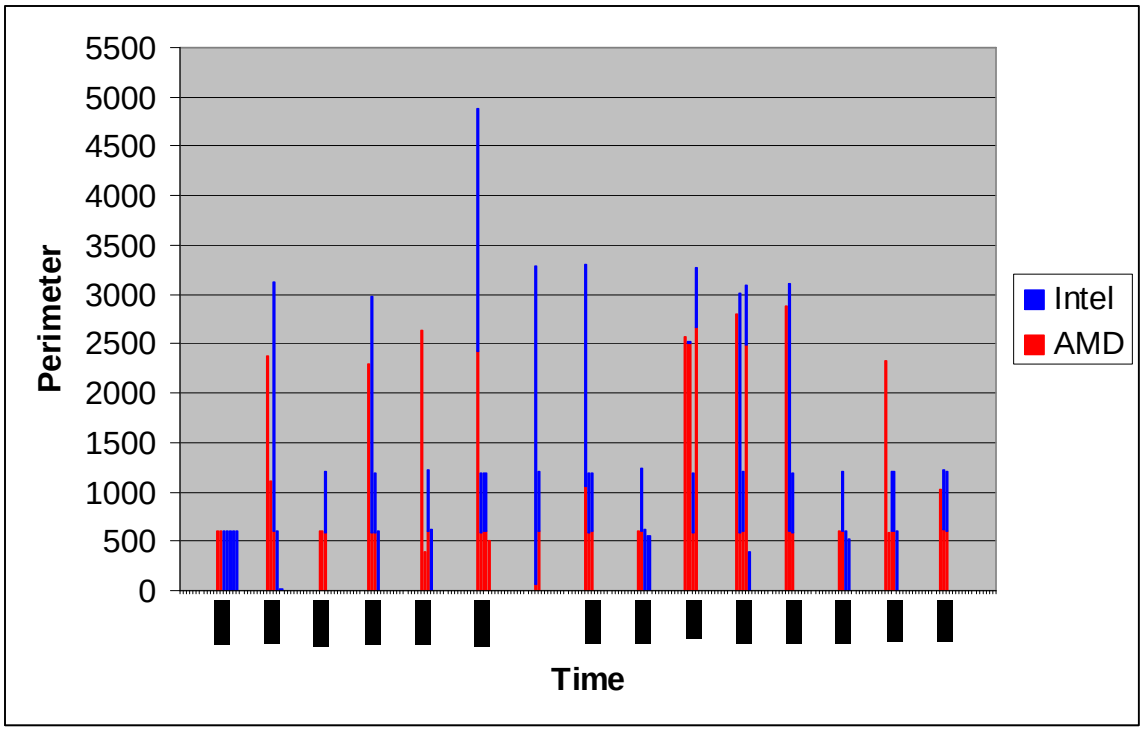

Figure 4 - 27 DVD Perimeter graph

Figure 4 - 27's X-axis indicates the flow of time for the video file, and the Y-axis indicates the perimeter value found during the specific time period. The figure clearly indicates 15 distinct perimeter groups - both for AMD and Intel - with all of them showing at least one perimeter above 407; this implies that an object was found for each of the 15 DVD objects in the video file.

These results are supported by Figure 4 - 28. This figure consists of a separate graph for AMD and Intel. A line on either of these 2 figures indicates a match. Both graphs have 15 lines, indicating 15 matches. This gives both the AMD and Intel solutions a 100 % success rate on matching the model to the DVD video.

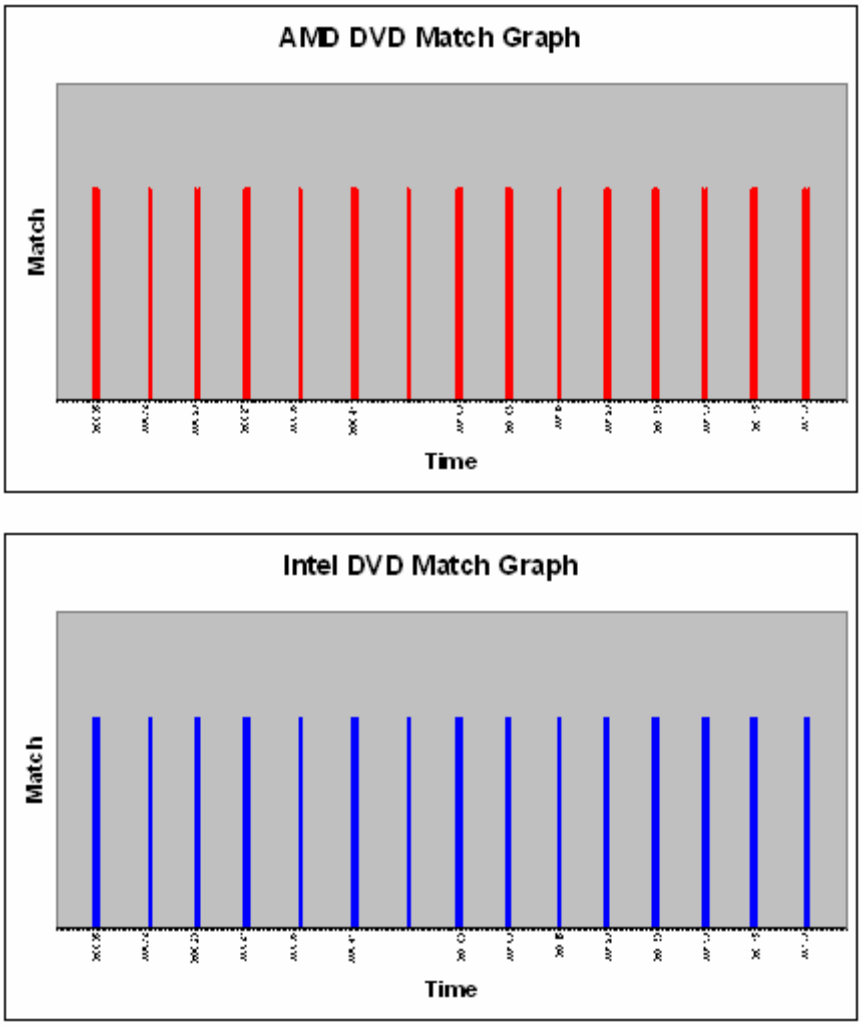

Figure 4 - 28 DVD Match graphs

4.2.3 Blender top

The next step was to determine whether the model could still distinguish between objects of a similar size and shape in continuous mode.

Figure 4 - 29 consists of two graphs - one for AMD and one for Intel. These graphs have a line for each object found on the Blender Top Video. Again, it is noted that the figures each have 15 distinct groupings of lines, meaning that both of the solutions achieved a 100% success rate at finding objects. However, no

matches were made, which implies that the system also has a 100% success rate in applying the model to the Blender Top video.

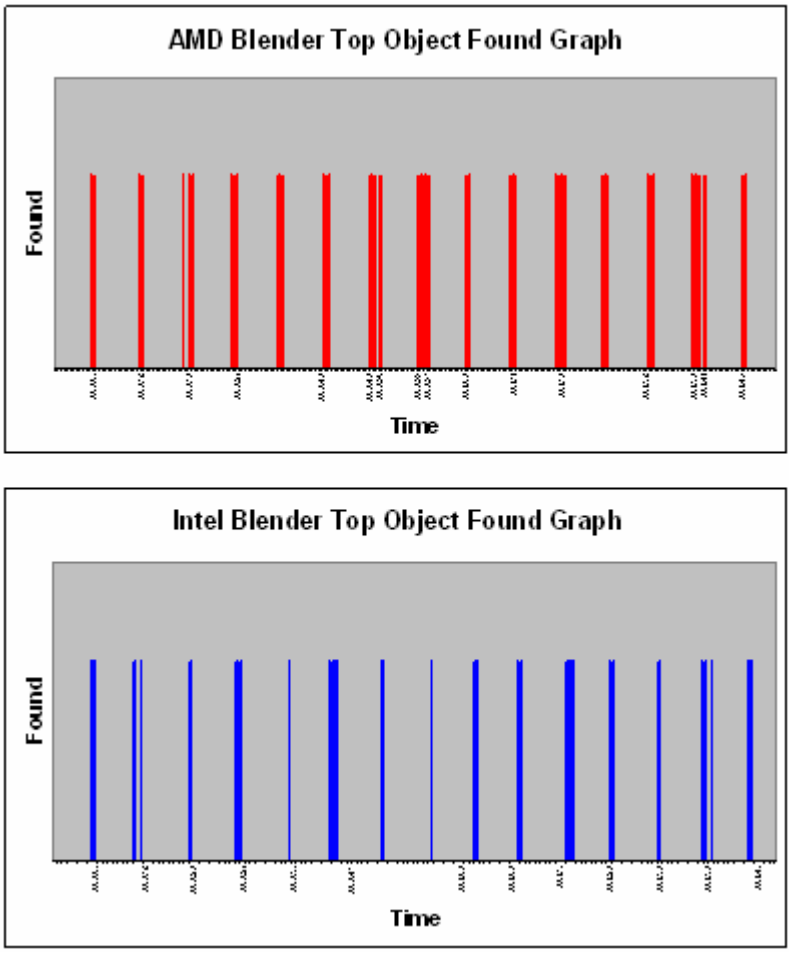

Figure 4 - 29 Blender Top Object Found Graphs

4.2.4 Canned fruit bottle cap

Next, it had to be determined whether the model could still distinguish between objects of a similar shape in continuous mode.

In Figure 4 - 30, which again plots perimeter against video time, we clearly see 14 groupings and then a separate 15th grouping for AMD and Intel; however, this still means that both the systems did find all 15 Canned Fruit Bottle Cap images. Only 3 of the Canned Fruit Bottle Caps triggered the AMD system to create a

perimeter higher than 407 (Figure 4 – 31b), while the same video triggered the Intel solution on 8 out of the 15 images (Figure 4 – 31a). Nevertheless, none of these triggers really constitute an error on the part of the system as no matches were made to the model, thus keeping the system's 100% match record intact. However, these triggers do show that the Intel-based system wasted more processing resources on objects that shouldn't have been processed further, although this could also mean that the Intel system was better at extracting perimeter information.

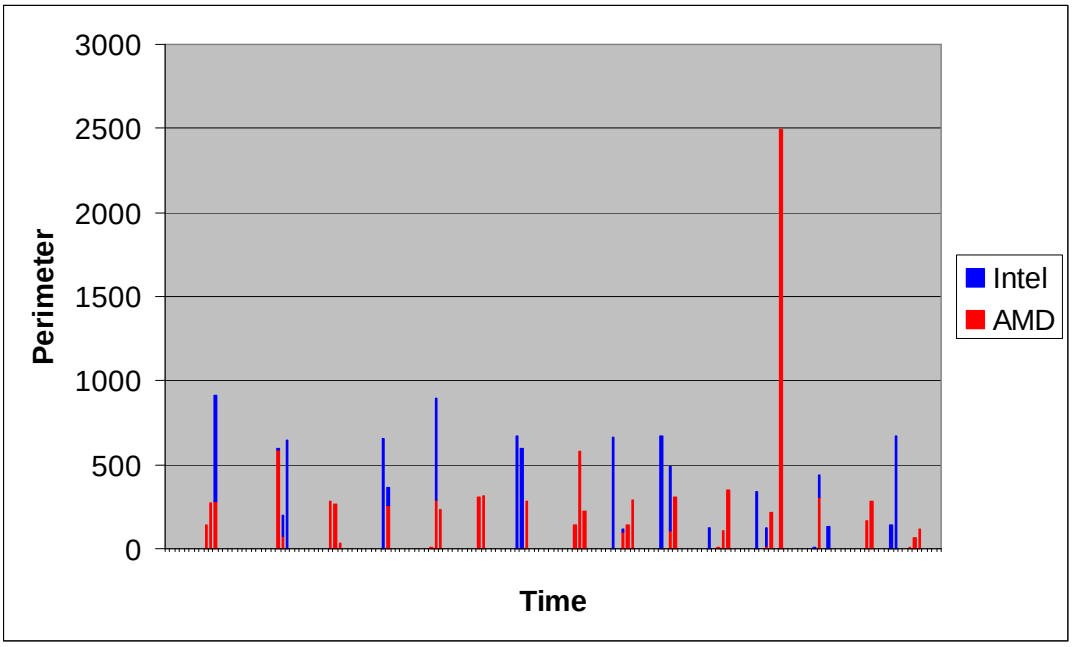

Figure 4 - 30 Canned Fruit Bottle Cap Perimeter graph

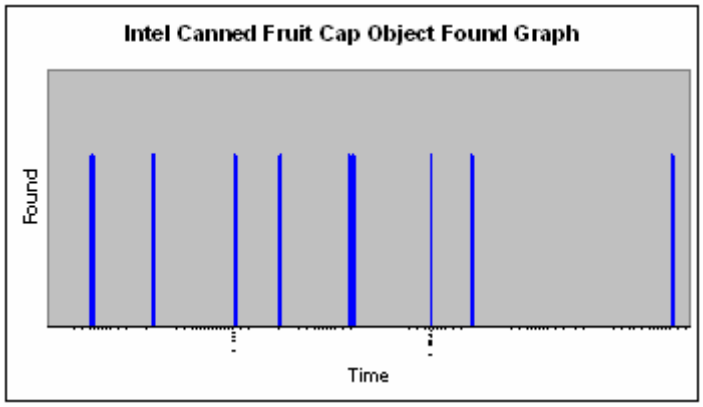

Figure 4 - 31a Canned Fruit Bottle Cap Found graphs

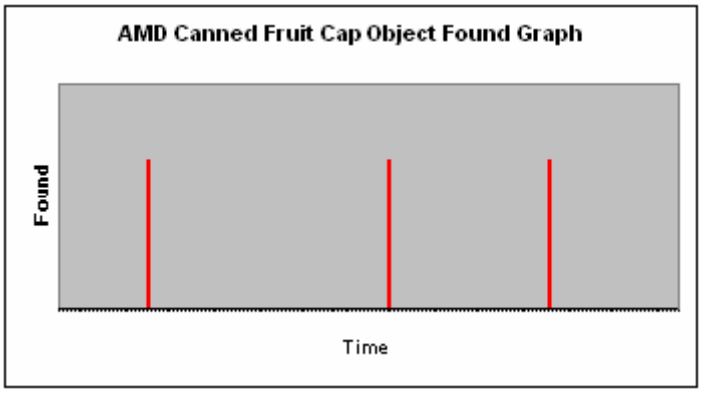

Figure 4 - 31b Canned Fruit Bottle Cap Found graphs

4.2.5 Playing card

To determine whether the model could still distinguish between objects of a dissimilar shape and size in continuous mode, the system was provided with a video file of 15 playing cards against which to try and match the model.

Figure 4 - 32 shows that the Intel system ended up with 15 groupings of perimeters, and the AMD system only with 14. In each of these groupings, both systems had at least one perimeter above 407. This means that the Intel system found 15 out of 15 objects, while the AMD system only found 14 - making the Intel system about 7% more effective at finding objects on the Playing Card video. However, these results are actually cancelled out due to the fact that no matches were found to the model on either system, which implies that - even though the AMD system was less reliable - it still did not make a recognition error. This also means that the RecMaster system retains a 100% success rate at matching the model to objects.

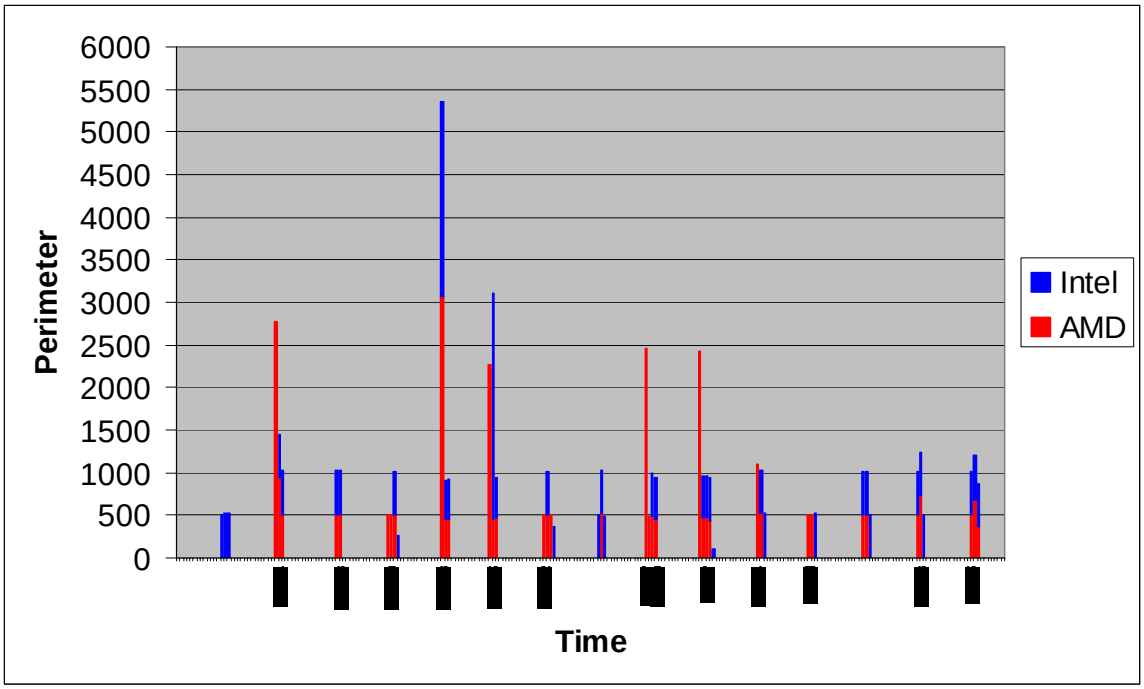

Figure 4 - 32 Playing Card Perimeter graph

4.2.6 DVD\blender top\playing card

The last step was to determine whether the model could distinguish between different kinds of objects in one video. To this end a video was created, consisting of 5 DVD objects intermixed with Blender Top and Playing Card objects. For this test, the Canned Fruit Bottle Cap objects were left out, since the tests done in both Still Image and Video Mode have shown that the system generally does not process them due to the fact that their perimeters are too low. For the sake of keeping the data set smaller, it was therefore opted to exclude the Canned Fruit Bottle Cap objects from this test.

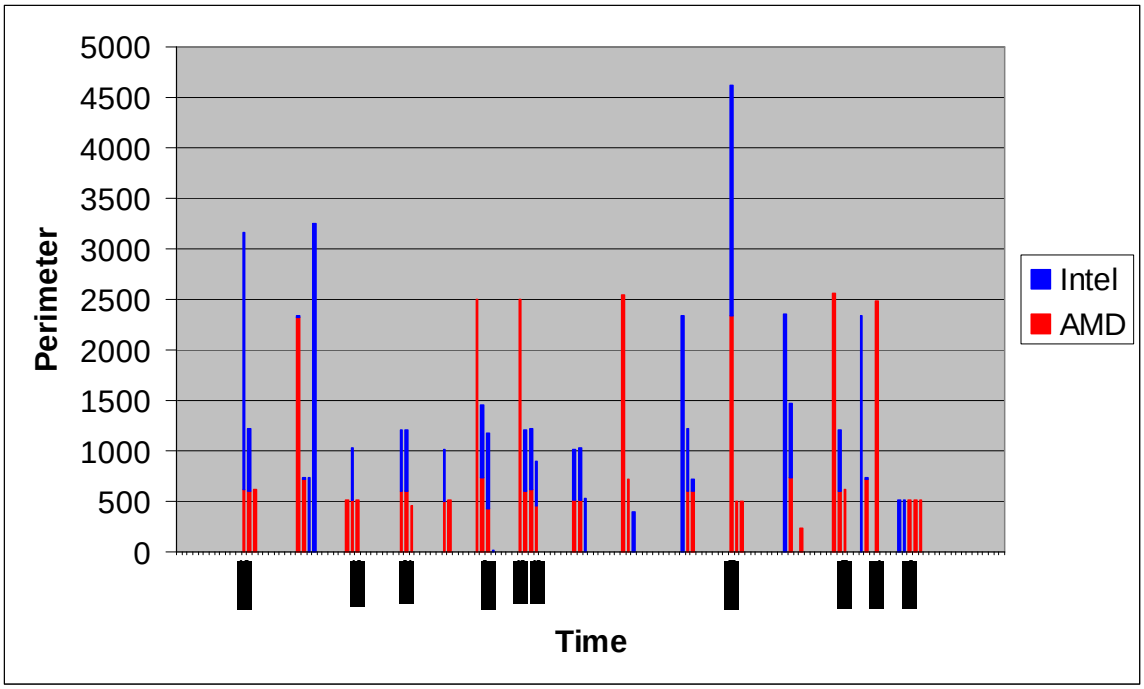

Figure 4 - 33 DVD\Blender Top\Playing Card Perimeter graph

Figure 4 - 33 indicates that the AMD-based system found perimeters above the cut-off value for each of the 15 objects passing under the camera. The Intel system, on the other hand, did not find an adequate perimeter on the 9th object. This means that, on this test, the AMD system had a 7% higher success rate at finding objects than the Intel system. These results are reflected in Figure 4 - 34, which displays the number of objects found on each system.

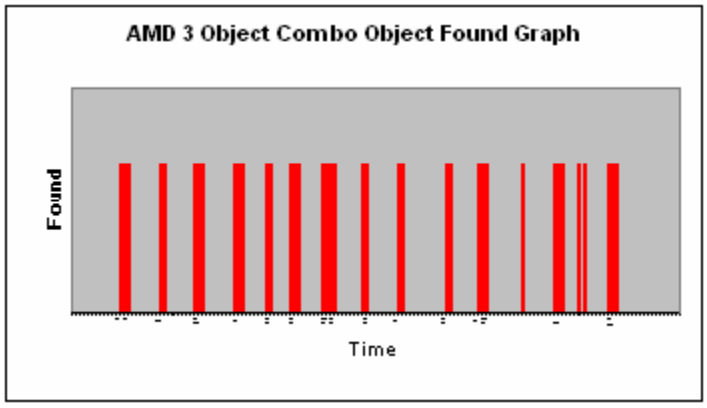

Figure 4 - 34a DVD\Blender Top\Playing Card Object Found graph

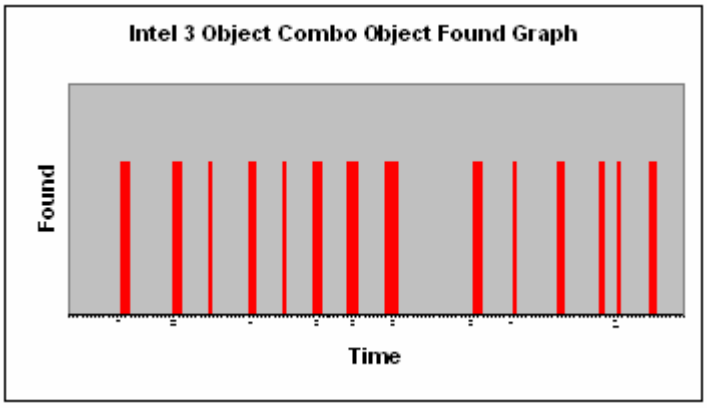

Figure 4 - 34b DVD\Blender Top\Playing Card Object Found graph

Lastly, Figure 4 - 35 indicates that both systems found 5 DVD objects in the DVD\Blender Top\Playing Card video. This gives both systems a 100% success rate at matching the model in Continuous mode. This also negates the 7% lead that the AMD system had in finding objects on the video, as both systems ended up with the same, correct final result.

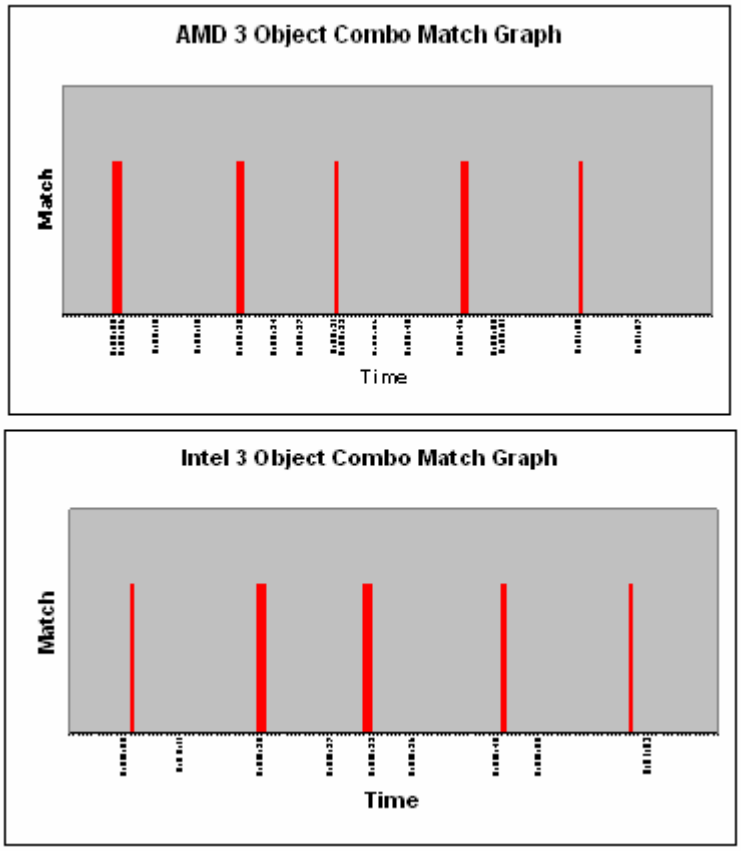

Figure 4 - 35 DVD\Blender Top\Playing Card Match graph

4.2.7 Recognition time

The Video Mode tests have all shown that both systems are indeed capable of correctly applying the chosen model. This is good news, as it means that the system will be capable of running on basically any desktop system that meets the minimum specifications. The last thing tested was the overall average time it took for each system to find either a positive match or no match. Figure 4 - 36 shows the results of this test.

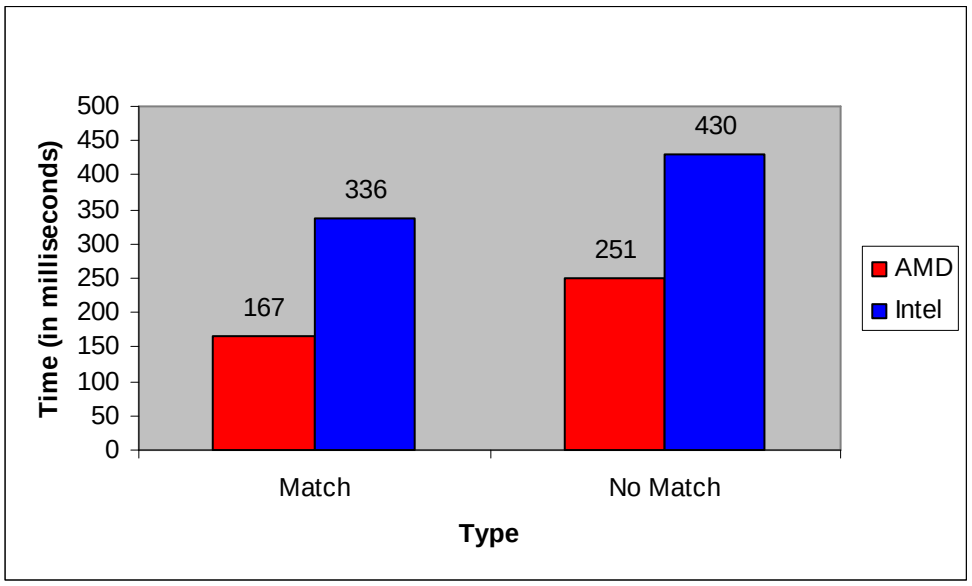

Figure 4 - 36 Average Video Recognition Speed graph

Figure 4 - 36 brings the following facts to light:

1. On the AMD system, it takes 84 milliseconds longer on average for the system to process an unmatched object than it does to process a matched one.
2. On the Intel system, it takes 94 milliseconds longer on average for the system to process an unmatched object than it does to process a matched one.

3. The closeness of these 2 differences signifies that the time difference between the processing of matched and unmatched objects is relatively constant.
4. The Intel system takes about 100% longer on average to find an object match than the AMD system does.
5. The Intel system takes about 71% longer on average to find an unmatched object than the AMD system does.

4.2.8 Final thought

Both of the systems have proved capable of running the RecMaster system at an acceptable success rate, thus proving the feasibility of running this image recognition system on an average desktop computer. The AMD system, however, has continuously proved faster at performing recognition tasks, both in Still Image and Video Mode. This seems to suggest that the system lends itself better to an AMD environment, which would probably make sense, as the system was developed on an AMD-based system and, as such, all the troubleshooting was done to accommodate this system. Thus, it is recommended that the RecMaster system be run on an AMD-based system in order to obtain the best performance.

The last chapter will outline the programmer's final thoughts and conclude the study.

Chapter 5

5. CONCLUSION

The development of the system was a great learning experience in the processes involved in image recognition. Apart from the algorithms used in the system, there are a great many others that did not fit into the system, but may be well suited to other tasks. Working through all of these algorithms would be a whole study in itself.

5.1 Agent implementation

One particularly problematic aspect of the development process was the actual implementation of the agents, as the programmer did not use an existing agent development framework. There are several methods that can be followed to implement agents, as outlined in Chapter 2. As Weiss [74] puts it; there is no universally accepted definition of the term agent - the only thing that everyone has seemed to reach consensus on is that they need to be autonomous. It was therefore decided to just conform to a few of the agent characteristics recognised in common literature. These characteristics are listed in Table 5 – 1.

Table 5 - 1 Agent characteristics

Agents in common literature	Agents in RecMaster
Communicate via Blackboard	*The agents in the RecMaster system pass information along to one another*
Cooperation and Specific Roles	*They each have an assigned task to complete before handing off the results to another agent*
Autonomy	*They are created and set to a waiting state until they have the necessary information on which to act*

The agents were implemented as classes in the RecMaster system, making them reusable in any other system. However, they were not compiled separately for the purpose of redistribution. The agents were created and then set to a waiting state until they are needed or another agent has completed its task.

5.2 Threading the system

The process of threading the system was limited to allowing several images to be processed at the same time, searching for an object's perimeter in an image and allowing the agents to run simultaneously in a waiting state in the background. It was attempted to thread the application so that different copies of the same agent would operate on different parts of the same image and the results obtained would then be combined by a master agent, but this proved to slow down the process of image recognition, rather than speed it up.

5.3 Satisfying the research hypothesis

The aim of this study was to prove the following:

It is possible for today's desktop computing systems to perform recognition-based quality control at an acceptable and reliable rate, using a multi-threaded, agent-based image recognition program. An acceptable rate is defined as matching objects to a model within the timeframe that the object is within view and a reliable rate is defined as providing consistent, accurate results.

In order to satisfy the hypothesis the following criteria needed to be met:

1. The RecMaster system is able to perform image recognition by matching an object to a previously created model at least 75 % of the time.

2. The RecMaster system is able to perform matching on still images within a time frame of at least 500 milliseconds.
3. The RecMaster system is able to perform matching on a video or camera feed while the object is still within the viewfinder at least 75 % of the time.
4. The RecMaster system is able to perform these matches consistently, i.e. creating matches at least 75 % of the time.

To this end, in Chapter 4, the system was tested in Still Image and Video Mode. The Still Image tests were more focused on testing the actual functioning of the image recognition system. Since the Still Image tests proved the system's functionality, the Video Mode was used to test the system's performance on an actual moving feed, which is the main purpose of the system. The results of testing the system generated the following facts, which are stated in order to satisfy the hypothesis.

1. The results of Table 4 – 2 show that the system has a success rate higher than 75 % for all of the object groups it was tested against, except for 2 of the object groups, as shown in Figure 4 -19 and Figure 4 -20, which has lower perimeters than the cut-off value. These objects were not tested to their full extent in order to save on processing resources. These results satisfy the first criterion of the hypothesis.
2. The results of Figure 4 – 26 show that both of the systems on which RecMaster was tested (AMD and Intel) are capable of forming matches, on still images, well under 500 ms. In fact the AMD system did so in an average time of 189 ms and the Intel system in an average time of 343 ms. This is far less than the 500 ms stated in order to satisfy the second criterion.
3. The results of Figure 4 - 36 shows that the AMD system is capable of forming matches, on a video feed, in only 167 ms and the Intel system takes 336 ms to do the same. This however does not prove that the system is able to perform the match while the object is still within the

viewfinder. In order to calculate whether the object is still within the viewfinder after the match has been made, the following measurements are needed:

a. Width of the conveyor: 47 cm.
b. Height of the conveyor: 35 cm.
c. Direction of the conveyor: Right to left in relation to the viewfinder.
d. Speed of the conveyor: Unknown.
e. Time object spends in view: About 2 s.

Unfortunately the exact speed of the conveyor was not known, so the following expression was used to calculate its speed:

47 cm / 2 s / 100 cm = 0.235 metres per second

Thus, the conveyor, on which the objects were placed, moves at a speed of about 0.235 m/s or 23.5 cm/s. These variables were used to calculate the distance an object would move during the recognition process on each of the systems:

AMD: 0.167 seconds * 23.5 cm = 3.9 cm
Intel: 0.336 seconds * 23.5 cm = 7.9 cm

Results are rounded off to 1 decimal value. This means that, on the AMD system, the object could be 3.9 cm from the edge of the screen and still be recognised before it leaves the view, while on the Intel system it could be 7.9 cm from the edge and still be recognised. On the AMD system, this means that the object could be recognised before leaving the view over 91.7 % of the total 47 cm view length. For the Intel system the percentage is a bit lower, with only 83.1 % of the total 47 cm view length guaranteeing recognition before the object leaves the view. In both cases the

percentage is higher than 75 %, meaning that the third criterion of the hypothesis has been satisfied.

4. Table 4 – 6 indicates that the AMD system had a 87 % success rate at forming correct matches, on still images, while it formed no incorrect matches at all. On the same test the Intel system has a 73 % success rate at forming correct matches, on still images

   As all the Video Mode data in Chapter 5 indicates, the system consistently located each of the objects the model was trained to find, on both of the system it was tested on. This means that the system has a 100% success rate at matching objects to the model; furthermore, it did not provide any false positives. In other words, it did not match any objects to the model that should not have been matched, giving it a 100% reliability and consistency rate. Thus, it was only in one of the tests that the system could not reach a 75 % match rate. In other words in three of the four tests the system did consistently form a match to the intended object, satisfying the fourth criterion of the hypothesis.

5.4 Avenues for future research

The aforementioned results satisfy all of the criteria set for the success of the study. However, no system is perfect. There are always those issues which need a little bit more polish or were not implemented exactly as the original vision entailed. Following are a few of the shortcomings of the study which may be improved by further study.

1. The system needs a relatively uniform background in order to function properly. This means that the RecMaster system is currently only suitable for use on a specific type of conveyor system, namely those consisting of a black belt running on a motor and a spool. Conveyor systems using a linked metal track conveyor would not be able to use the RecMaster

system, because the angular nature of the background would interfere with the functioning of the filter responsible for finding the object's outline (see 3.4 Measuring the object).

2. The system operates in a limited brightness spectrum, meaning that the light conditions during the model creation period and the recognition period should be as close as possible.
3. The system does not take lens distance into account, and therefore does not do any scaling. The recognition should therefore be done on a feed in which the object to lens distance is the same as when the model was created.
4. Colour values are ignored, so the system only looks at the edges and the general shape of the object. If an object accidentally comes out green instead of blue, the system would currently not detect this mistake.
5. The system also needs a relatively steadfast level of image contrast during the model creation and recognition phases. These limitations make the implementation scope of the system relatively narrow, and will be improved upon if another incarnation of the system is required.

5.5 Final thought

It has been demonstrated that the developed system is capable of performing recognition on both still images and a video or camera feed. Even though there are still some imperfections in the system which may benefit from future development, the system has proven itself capable of performing its intended task using the underlying technologies of agents and threading. Whether the system will be implemented as is or only after future additions has yet to be seen. There is no doubt that the final implementation of the system will benefit the process of quality control in a production environment by aiding the operator to distinguish between correctly and incorrectly manufactured objects. It also has the added benefit of being able to perform recognition by making use of standard

desktop computing equipment; therefore providing a financially sound incentive for its implementation.

This marks the end of a long journey of discovery, but as Greg Anderson, a best-selling American author and founder of the American Wellness Project, puts it

> *"Focus on the journey, not the destination. Joy is found not in finishing an activity but in doing it."*

REFERENCES

[1] *Abidi, Mongi, Boughorbel, Faysal and Koscan, Andreas*, modelling 3D Objects from Range Maps and Color Images using a Warping-based Approach, 6th International Conference on Quality Control and Vision, May 2003

[2] *Alsinet, T., Ansotegui, C., Bèjar, R., Fernandez, C. and Manyà, F.*, Automated monitoring of medical protocols: a secure and distributed architecture, Artificial Intelligence in Medicine 27 , Elsevier, 2003

[3] *Anderson, Dave (Site Founder)*, PC TechGuide: AMD Technology, http://www.pctechguide.com/22non-inte.htm, 18 April 2004

[4] *Armengol, Eva and Lopez de Mantaras, Ramon*, Machine Learning from examples: Inductive and Lazy methods, Data & Knowledge Engineering, Elsevier, November 1997

[5] *Bailey, Bob*, Moments in image processing, http://www.csie.ntnu.edu.tw/~bbailey/Moments%20in%20IP.htm, November 2002, Retrieved 6 July 2005

[6] *Ballard, D.H. and Brown, C.M.*, Computer Vision, Prentice Hall, New Jersey, 1982

[7] *Barnsley, Micheal J. and Barr, Stuart L.*, Distinguishing urban land-use categories in fine spatial resolution land-cover data using a graph-based, structural pattern recognition system, Computer, Environment and Systems 21, Pergamon, 1997

[8] *Barros, L., Bianchi, R. and Rillo, A.*, The VIBRA Multi-Agent Architecture: integrating purposive vision with deliberative and reactive planning, http://www.lti.pcs.usp.br/~rbianchi/publications/boletim-poli1998.pdf, 1998, Retrieved 19 May 2005

[9] *Barry, Alwyn*, LCSWEB, http://lcsweb.cs.bath.ac.uk/LCSWiki/InductionAlgorithm, 2005, Retrieved 25 January 2006

[10] *Berg, Daniel J. and Lewis, Bill*, Multithreaded Programming with PThreads, Sun Microsystems Press/Prentice Hall, 1998

[11] *Blum, Avrim L. and Langley, Pat*, Selection of relevant features and examples in machine learning, Artificial Intelligence, Elsevier, May 1996

[12] *Borges, Dibio L. and Fisher, Robert B.*, Class-based recognition of 3D objects represented by volumetric primitives, Image and Vision Computing 15 p. 655 - 664, August 1997

[13] *Bosch, J.G., Dijkstra, J. and Reiber, J.H.C.*, Multi-agent segmentation of IVUS images, EGP Bovenkamp, www.sciencedirect.com, 16 September 2003

[14] *Boyle, R., Hlavac, V. and Sonka, M.*, Image Processing, Analysis and Machine Vision, Chapman & Hall, 1993

[15] *Braue, David*, AI think, therefore I am, http://www.apcmag.com/apc/v3.nsf/0/C89F33A82CCA4BBCA256DEC00036F27, 16 December 2003, Retrieved 19 May 2005

[16] *BURLE Industries Inc.*, Photosensitive Camera Tubes and Devices Handbook, BURLE Industries Inc., http://www.burle.com/cgi-bin/byteserver.pl/pdf/pctdhbook.pdf, 2005, 25 January 2006

[17] *Canny, J.*, A Computational Approach to Edge Detection, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 8, No. 6, Nov. 1986

[18] *Chelberg, David M., Gillen, Matthew, Parott, David, Welch, Lonnie and Zhou, Qiang*, Agent-based computer vision in a dynamic, real-time environment, Pattern Recognition 37, Issue 4, pages 691 - 705, April 2004

[19] *Chira, Camelia*, Software Agents, IDIMS Report, 21 February 2003

[20] *Christmas, William, Kittler, Josef and Kostin, Alexey*, Object recognition by symmetrised graph matching using relaxation labeling with an inhibitory mechanism, Pattern Recognition Letters 26 , Issue 3, pages 381 - 393, February 2005

[21] *Dance, Sandy and Gorman, Malcolm*, Intelligent Agents in the Australian Bureau of Meteorology, http://www.agentcities.org/Challenge02/Proc/, April 2004, Retrieved on 19 May 2005

[22] *Davies, E.*, Machine Vision: Theory, Algorithms and Practicalities, Academic Press, 1990

[23] *de Ridder, D., Egmont-Peterson, M. and Handels, H.*, Image Processing with

Neural Networks – a review, Pattern Recognition 35, Issue 10, pages 2279 - 2301, October 2002

[24] *Decker, K., Sycara, K. and Williamson, M.*, Middle-Agents for the Internet, Proceedings of IJCAI-97, January 1997

[25] *Deguchi, Koichiro and Yanai, Keiji*, A Multi-resolution Image Understanding System Based on Multi-agent Architecture for High-resolution Images, IEICE Transactions on Information and Systems, Vol.E84-D, No.12, December 2001

[26] *Doran, J.E., Franklin, S., Jennings, N.R. and Norman, T.J.*, On Cooperation in Multi-Agent Systems, http://www.agent.ai/doc/upload/200302/dora97.pdf, 1997, Retrieved on 19 May 2005

[27] *Ferber, Jacques*, Multi-Agent System: An Introduction to Distributed Artificial Intelligence, Harlow: Addison Wesley Longman, 1999

[28] *Franklin, Stan and Graesser, Art*, Agent or Program. Is it an Agent, or just a Program?, Proceedings of the Third International Workshop on Agent Theories, Architectures and Languages, 1996

[29] *Frey, Herbert,* Machine Vision, Machine Vision Workshop, Central University of Technology, Free State, 19 – 21 September 2005

[30] *Gonzalez, R. and Tou, J.*, Pattern Recognition Principles, Addison-Wesley Publishing Company, Massachusetts, 1974

[31] *Gonzalez, R.C. and Woods, R.E.*, Digital Image processing (3rd ed.), Addison Wesley, 1992

[32] *Green, Bill*, Edge Detection Tutorial, www.pages.drexel.edu/~weg22/edge.html, 2002, Retrieved on 22 March 2006

[33] *Haralick, R. and Shapiro, L.*, Computer and Robot Vision Vol. 1, Addison-Wesley Publishing Company, 1992

[34] *Harding, K.G. and Tait, R.*, Moire techniques applied to automated inspection of machined parts, In Vision 1986 Conference Proceedings (Machine Vision Association of SME), 1986

[35] *Hayes-Roth, B.*, A blackboard architecture for control, Artificial Intelligence Volume 26, Issue 3, pages 251 - 321, July 1985

[36] *Hedger, Stuart*, Artificial Intelligence: Intelligent Agents and the Internet,

http://osiris.sunderland.ac.uk/cbowww/AI/TEXTS/AGENTS5/ass_ht~1.htm, 23 January 1997, Retrieved on 19 May 2005

[37] *Holota, Radek and Nemecek, Stanislav*, Recognition of Oriented Structures by 2D Fourier Transform, http://home.zcu.cz/~holota5/publ/rdsb2dft.pdf, 30 September 2002, Retrieved on 22 March 2006

[38] *Horn, B.K.P.*, Robot Vision, MIT Press, 1986

[39] *Horn, B.K.P.*, Shape from Shading: A Method for Obtaining the shape of a Smooth Opaque from One View, Ph.D. Study, Massachusetts Institute Of Technology, 1970

[40] *Howard, Ian*, Speech Fundamental Period Estimation Using Pattern Classification, PhD Study, Faculty of Science, Phonetics & Linguistics, UCL, University of London, 1991

[41] *Hughes, Stephen, Lewis, Mike, Manojlvich, Josep and Prasithsangaree, Phongsak*, UTSAF: A Multi-Agent –Based Software Bridge for Interoperability between Distributed Military and Commercial Gaming Simulations, Simulation Vol. 80 No. 12, 2004

[42] *Iba, W and Langley, P*, Average-case analysis of a nearest neighbor algorithm, in : Proceedings IJCAI-93, Chanbery, France, 1993

[43] *Iossifidis, C., Karathanassi, V. and Rokos, D.*, Application of machine vision techniques in the quality control of pharmaceutical solutions, Computers in Industry Volume 32, Issue 2, pages 169 - 179, December 1996

[44] *Kalp, D., Paolucci, M., Shehory, O. and Sycara, K.*, A Planning Component for RETSINA Agents, Lecture Notes in Artificial Intelligence, Intelligent Agents VI. M. Wooldridge and Y. Lesperance (Eds.), 1999

[45] *Laaksonen, Jorma, Lampinen, Jouko and Oja, Erkki,* Pattern Recognition, Image Processing and Pattern Recognition, Academic Press, 1998

[46] *Langner, Jens*, Leaves Recognition V1.0: Neuronal Network based recognition system of leaf images, http://www.jens-langner.de/lrecog/, 28 October 2004, Retrieved on 20 March 2006

[47] *Lee, Heyoung, Son, Won-Kyung and Bien, Zeungnam*, KARES: Intelligent wheelchair-mounted robotic arm system using vision and force sensor, Robotics

and Autonomous Systems Volume 28, Issue 1, pages 83 - 94, July 1999

[48] *Lee, Raymond S.T.*, iJADE Surveillant – an intelligent multi-resolution composite neuro-oscillatory agent-based surveillance system, Pattern Recognition Volume 36, Issue 6, pages 1425 - 1444, June 2003

[49] *Lewis, Steven H. and Milewski, Allen E.*, Delegating to Software Agents, International Journal for Human-Computer Studies Volume 46, Issue 4, pages 485 - 500, April 1997

[50] *Lieberman, Henry*, Autonomous Interface Agents, Proceedings of the ACM Conference on Computers and Human Interface, CHI-97, Atlanta, Georgia, March 1997

[51] *Lucas, Chris*, Complex Systems Glossary, http://www.calresco.org/glossary.htm, June 2005, Retrieved on 20 March 2006

[52] *MASS Group Inc.*, Manufacturing Automation Software & Systems Group, http://www.massgroup.com/Products/visionsystems2.asp, 2005, Retrieved on 13 February 2006

[53] *McCallum, Andrew Kachites and Rennie, Jason*, Using Reinforcement to Learning to Spider the Web Efficiently, Proceedings of the Sixteenth International Conference on Machine Learning (ICML), 1999

[54] *Mitchell, T.N. and Utgoff, P.E.*, Acquisition of appropriate bias for inductive concept learning, Proceedings of the National Conference on Artificial Intelligence, 1982

[55] *Munneke, Derek, Wahlstrohm, Kirsten and Zaccara, Linda*, Intelligent Software Robots on the Internet, http://www.cis.unisa.edu.au/~cisdm/papers/iagents /IntelligentAgentsInternet.html, October 1998, Retrieved on 18 May 2005

[56] *Norvig, Peter and Russel, Stuart J.*, Artificial Intelligence: A Modern Approach, Englewood Cliffs, Prentice Hall, 1995

[57] *Nwana, Hyacinth S.*, Software Agents: An Overview, Knowledge Engineering Review Vol. 11, Cambridge University Press, 1996

[58] *Okkalides, Demetrios*, Assessment of commercial compression algorithms, of the lossy DCT and lossless types, applied to diagnostic digital image files, Computerized Medical Imaging and Graphics Volume 22, Issue 1, pages 25 - 30,

January/February 1998

[59] *Pedrycz, W. and Vasilakos, A.V. (Eds.)*, Computational Intelligence in Telecommunications Networks, CRC Press, 2000

[60] *Pernkopf, Franz*, Bayesian network classifiers versus selective k-NN classifier, Pattern Recognition, Volume 38, Issue 1, pages 1 - 10, October 2003

[61] *Public Domain*, Dr Dobbs: Microprocessor Resources, http://www.x86.org/articles/computalk/help.htm, Retrieved on 15 June 2005

[62] *Public Domain*, Wikipedia (Keywords: Edge Detection), www.en.wikipedia.org, Retrieved on 20 March 2006

[63] *Rana, Omer F. and Rosin, Paul L.*, Agent-based computer vision, Pattern Recognition Volume 37, Issue 4, pages 627 - 629, April 2004

[64] *Ravishankar Rao, A.*, Future directions in industrial machine vision: a case study of semiconductor manufacturing applications, Image and Vision Computing Volume 14, Issue 1, pages 3 -19, February 1996

[65] *Scharstein, Daniel and Szeliski, Richard*, High Accuracy Stereo Depth Maps using Structured Light, CVPR, pages 195-202, 2003

[66] *Scheidegger, Thomas*, The Code Project: DirectShow.NET, http://www.codeproject.com/cs/media/directshownet.asp, 23 Jul 2002, Retrieved on 6 July 2006

[67] *Schleiffer, Ralf*, An Intelligent Agent model, European Journal of Operational Research Volume 166, Issue 3, pages 666 - 693, November 2005

[68] *Shapiro, L. and Stockman, G.*, Computer Vision, Prentice Hall, 2001

[69] *Shirai, Y.*, Three-Dimensional Computer Vision, Springer-Verlag, New York, 1987

[70] *Smirnova, Vira*, Multi-Agent System for Distributed Data Fusion in a Peer-to-Peer Environment, Master's Study, November 2002

[71] *Sycara, Katia P.*, Multiagent Systems, AI Magazine, pages 79 - 92, Summer 1998

[72] *Sylla, Cheickna,* Experimental investigation of human and machine-vision arrangements in inspection tasks, Control Engineering Practice Volume 10, Issue 3, pages 347 - 361, March 2002

[73] *Tadrous, Paul J.*, A simple and sensitive method for directional edge detection in noisy A simple and sensitive method for directional edge detection in noisy images, Pattern Recognition Volume 28, Issue 10, pages 1575 - 1586, October 1995

[74] *Weiss, Gerhard,* Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence, MIT Press, Cambridge, MA, 1999

[75] *Zheng, Lihong and He, Xiangjian*, Classification Techniques in Pattern Recognition, http://wscg.zcu.cz/wscg2005/Papers_2005/Poster/K43-full.pdf, 2005, 20 March 2006

Appendix A

A. PROGRAM DESIGN

This section outlines the most important aspects of the RecMaster system. The function of each aspect, of the programme, is described and a listing of its components with their individual functions is provided. Every aspect is also accompanied by a diagram outlining its main procedural flow. The top-most item of each flowchart always indicates the form or process from which processing has passed.

A.1 Splash Screen

This form provides the user with a load screen to show which program is loading and who the author of the program is. It acts as an intermediary between the operating system and the program's main interface.

The most important components of the form are as follows:

A.1.1 Important unnamed components

1. A picture box in which to display the program's logo.
2. A label in which to display the program's name.
3. A label in which to display the author's name.

A.1.2 Timer tmSplash

This Timer is used to keep track of the length of time that the Splash Screen is being displayed.

A.1.3 Procedural flow diagram

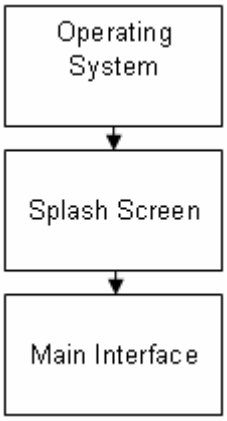

Figure A - 1 Splash Screen procedural flow diagram

A.2 Main Interface

This form acts as the link to all of the other program functions. Basically, it only acts as a menu and a container for all of the other forms.

The most important components of the form are as follows:

A.2.1 MainMenu MainMenu

This MainMenu acts as a drop-down menu, containing links to all of the program's main aspects.

A.2.2 ToolBar tbMain

This ToolBar acts as a visual menu, containing links to most of the program's main aspects.

A.2.3 Panel pnlMain

This Panel acts as an always visual information display.

A.2.5 Procedural flow diagram

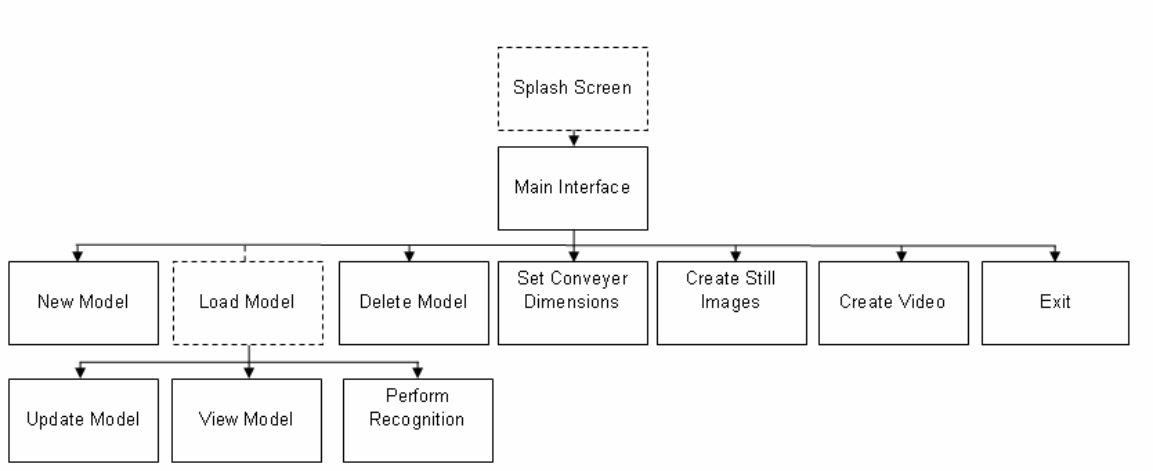

Figure A - 2 Main Interface procedural flow diagram

Procedural flow continues on from the Splash Screen into the various functions of the program. The Load Model function is not visible on the main interface; it acts as an intermediary for 3 functions that are visible on the main interface, namely Update Model, View Model and Perform Recognition.

A.3 New Model

This form allows the user to view the names of the Models which are already in the system and then type the name of a new Model to create. The user has the options of either creating the Model and continuing on to the Update Model process or returning to the Main Interface.

The most important components of the form are as follows:

A.3.1 ListBox lbModels

This ListBox displays the names of all of the Models currently in the system.

A.3.2 TextBox txtModel

This TextBox allows the user to type the name of the new Model to be created.

A.3.3 Buttons

Table A - 1 New Model buttons

Name	Function
cmdCreate	*This Button makes sure that the new Model name does not exist and then creates the Model file and directory.*
cmdCancel	*This Button returns the user to the main interface without saving anything.*

A.3.4 Procedural flow diagram

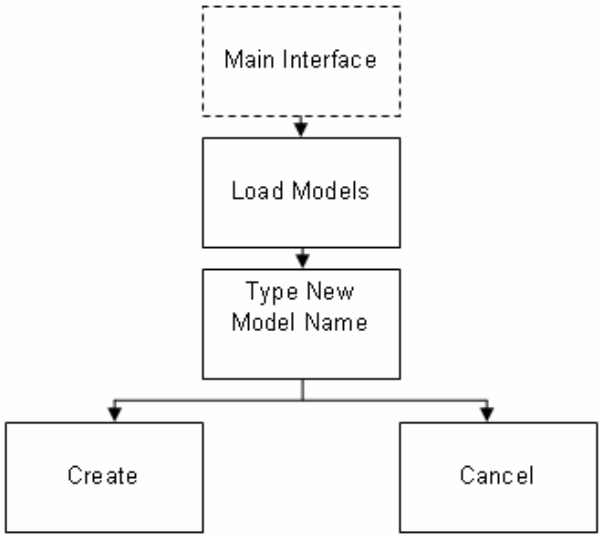

Figure A - 3 New Model procedural flow diagram

In practice the procedural flow would pass from Create to the Update Model process, but this is not really integral and as such is omitted. The Update Model process is typically accessed from the main interface and is only added here as a convenience.

A.4 Delete Model

This form allows the user to view the names of the Models which are already in the system and then select the name of a Model to delete. The user has the options of either deleting the selected Model or returning to the Main Interface.

The most important components of the form are as follows:

A.4.1 ListBox lbModels

This ListBox displays the names of all of the Models currently in the system and allows the user to select one.

A.4.2 TextBoxes

Table A - 2 Delete Model textboxes

Name	Function
txtModel	*This TextBox displays the name of the Model to be deleted.*
txtLastModified	*This TextBox displays the last time that the current Model was updated.*

A.4.3 Buttons

Table A - 3 Delete Model buttons

Name	Function
cmdDelete	*This Button deletes the selected Model and determines if there are any Models left in the system after deletion.*
cmdCancel	*This Button returns the user to the main interface.*

A.4.4 Procedural flow diagram

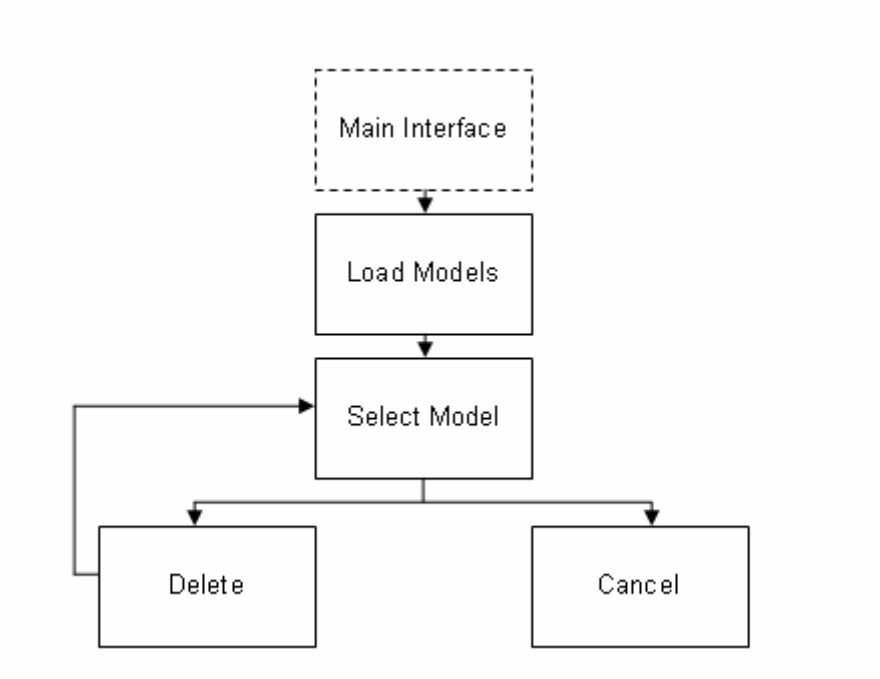

Figure A - 4 Delete Model procedural flow diagram

A.5 Load Model

This form allows the user to view the names of the Models which are already in the system and then select the name of a Model(s) to use. This form acts as an intermediary between the main interface and either the Update Model, View Model or Perform Recognition processes. The Model(s) selected on this form is used in the follow-up process. The user has the options of either loading the selected Model(s) or returning to the Main Interface.

The most important components of the form are as follows:

A.5.1 ListBoxes

Table A - 4 Load Model listboxes

Name	Function
lbModels	*This ListBox displays the names of all of the Models currently in the system and allows the user to select one.*
lbSelected	*This ListBox displays the names of the Models selected to be used in the Perform Recognition process.*

A.5.2 TextBoxes

Table A - 5 Load Model textboxes

Name	Function
txtModel	*This TextBox displays the name of the Model to be deleted.*
txtLastModified	*This TextBox displays the last time that the current Model was updated.*

A.5.3 Buttons

Table A - 6 Load Model buttons

Name	Function
cmdLoad	*This Button loads either the Update Model, View Model or Perform Recognition process and initializes them with the Model(s) chosen.*
cmdCancel	*This Button returns the user to the main interface.*

A.5.4 Procedural flow diagram

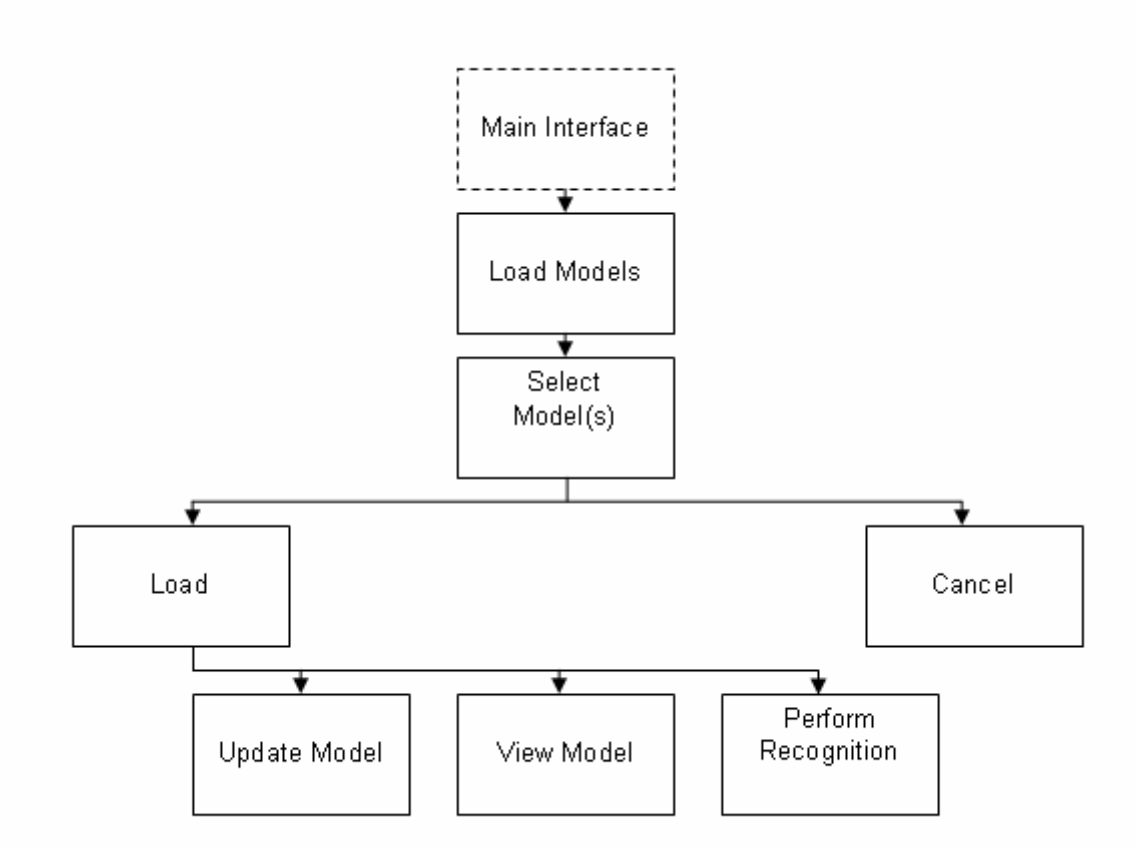

Figure A - 5 Load Model procedural flow diagram

A.6 Update Model

This form updates the measurements of a chosen Model. These measurements consist of 5 fields:

1. Object Perimeter
2. Object Area
3. Distance between furthest X-axis points of object (bounding box)
4. Distance between furthest y-axis points of object (bounding box)
5. Number of Edges on the object's Edgegraph

These values are obtained by running the various agents on a single image, multiple images, frames of a video file or frames of a camera (device) feed. In single image, batch image and single frame of video/camera mode the decision is in the user's hands as to whether or not to accept the calculated measurements. In continuous video/camera mode frames are continuously captured so there is no time for the user to accept each set of measurements. It works by allowing the user to accept a range of values for the perimeter. If the perimeter measurements fall within the range, then all of the measurements are accepted. In order for this to work correctly, a few samples have to be taken in single image, batch image or single frame of video/camera mode so that a frame of reference is known for the perimeter values.

The most important components of the form are as follows:

A.6.1 ToolBars

Table A - 7 Update Model toolbars

Name	Function
tbUpdateModel	*This ToolBar allows the user to select which Mode processing should be performed in.*
tbPictureAdjustment	*This ToolBar allows the user to select the Adjust Brightness and Adjust Contrast options in Single Image Mode.*
tbVideoOptions	*This ToolBar allows the user to choose between Single Frame and Continuous Mode in Video or Camera Mode.*

A.6.2 TabPages

Table A - 8 Update Model tabpages

Name	Function
tabNormal	*This TabPage displays the Single Image Mode components.*
tabRoughEdges	*This TabPage displays the extended results from Single Image Mode processing.*
tabBatch	*This TabPage displays the Batch Image Mode components.*
tabCamVideo	*This TabPage displays the Video and Camera Modes.*

A.6.3 PictureBoxes

Table A - 9 Update Model pictureboxes

Name	Function
picBatchGraphs	*This PictureBox displays EdgeGraphs generated during Batch Image Mode.*
picCapturePreview	*This PictureBox displays the results of the currently processed frame in Video/Camera Mode.*
picCamVideo	*This PictureBox displays the video file or camera (device) feed.*
picCurrentEdges	*This PictureBox displays the EdgeGraph generated during Single Image Mode.*
picPicturePreview	*This PictureBox displays the results of the currently processed image in Single Image Mode.*
picQueue1 – picQueue4	*These PictureBoxes display the image currently being processed in the specific queue in Batch Image Mode.*
picStudy	*This PictureBox displays the image currently being processed in Single Image Mode.*

A.6.4 Buttons

Table A - 10 Update Model buttons

Name	Function
cmdAccept	*This Button accepts the measurements generated in Single Image Mode.*
cmdAcceptBatch	*This Button accepts the measurements generated in Batch Image Mode.*
cmdAcceptCamVideo	*This Button accepts the measurements generated in Single Frame Video/Camera Mode.*
cmdActivate	*This Button activates Single Frame Video/Camera Mode or starts and stops Continuous Frame Video/Camera Mode.*
cmdBatchActivate	*This Button activates Batch Image Mode.*
cmdPictureProcess	*This Button activates Single Image Mode.*

A.6.5 RadioButtons

Table A - 11 Update Model radiobuttons

Name	Function
optArea	*This RadioButton selects the Area Single Image Mode.*
optWhole	*This RadioButton selects the Whole Image Single Image Mode.*

A.6.6 Timer tmCamVideoCapture

This Timer captures a frame from the video file or camera (device) feed and processed it every 500 ticks (milliseconds).

A.6.7 Labels

There are a variety of Labels on the form which are used to display the information generated during processing.

A.6.8 Procedural flow diagram

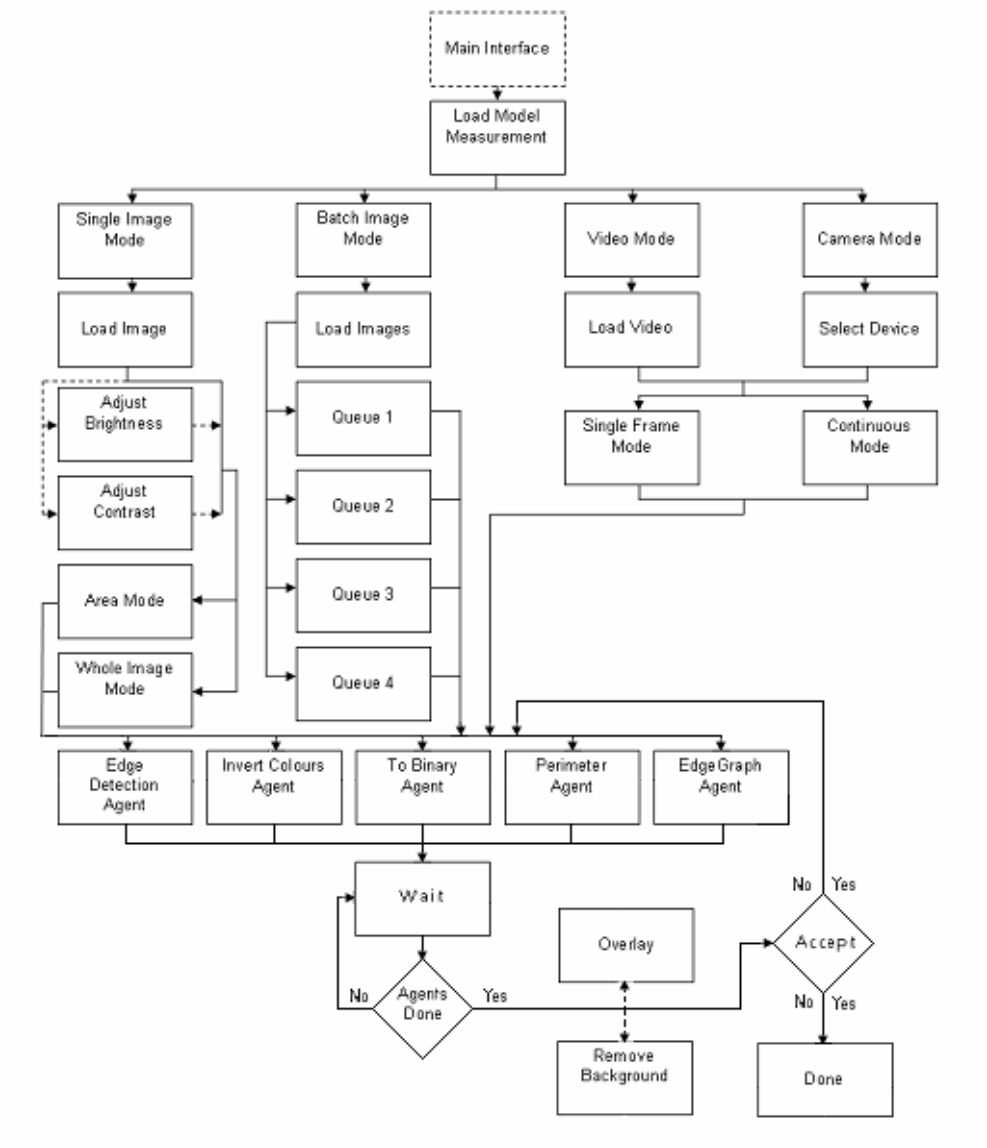

Figure A - 6 Update Model procedural flow diagram

A.7 View Model

This form allows the user to view measurements of a Model chosen on the Load Model form. All of the EdgeGraphs, generated during the Update Model process, are also displayed.

The most important components of the form are as follows:

A.7.1 PictureBox picModelImages

This PictureBox displays the EdgeGraphs one at a time.

A.7.2 Buttons

Table A - 12 View Model buttons

Name	Function
cmdNext	*These Buttons are used to navigate the various EdgeGraphs.*
cmdPrevious	

A.7.3 Labels

Various Labels are used to display the measurements of the current Model.

A.7.4 Procedural flow diagram

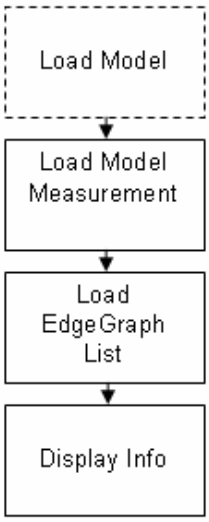

Figure A - 7 View Model procedural flow diagram

A.8 Set Conveyor Measurements

This form allows the user to set up some settings regarding the conveyer belt using the feed from the imaging device (camera), which is stationed over the conveyer belt. In particular:

1. The length in millimeter of the X-axis.
2. The length in millimeter of the Y-axis.
3. The speed of the conveyer belt in meters per second.
4. The direction, in relation to the on-screen image, that the conveyer belt is moving in.

These fields can be updated one at a time and need to be as accurate as possible in order for the Perform Recognition phase to report accurate results.

The most important components of the form are as follows:

A.8.1 Panel pnlCam

This Panel is used to display the camera (device) feed.

A.8.2 Labels

Various Labels are used to display the current saved conveyer settings.

A.8.3 TextBoxes

Table A - 13 Set Conveyor Measurements textboxes

Name	Function
txtXDistance	*This TextBox allows the user to enter the X-axis measurement.*
txtYDistance	*This TextBox allows the user to enter the Y-axis measurement.*
txtSpeed	*This TextBox allows the user to enter the conveyer speed measurement.*

A.8.4 RadioButtons

Table A - 14 Set Conveyor Measurements radiobuttons

Name	Function
optUpwards	*These RadioButtons allow the user to select the direction in which the conveyer is moving in relation to the on-screen image.*
optDownwards	
optLeft	
optRight	

A.8.5 Button cmdUpdate

This Button allows the user to update the conveyer settings.

A.8.6 Procedural flow diagram

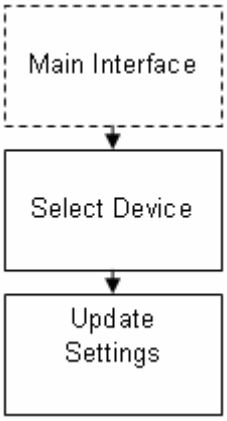

Figure A - 8 Set Conveyor Measurements procedural flow diagram

A.9 Perform Recognition

This form takes object measurements from a single image, a video file or a camera (device) feed and compares them to those of a chosen Model(s) to determine a match. These values are obtained by running the various agents on a single image, frames of a video file or frames of a camera (device) feed. Any matching fields are indicated with a picture next to the field. The time taken to perform the recognition is also indicated. In Single Image Mode and Single Frame Video/Camera Mode the results remain visible until the process is run again, but during Continuous Video/Camera Mode the results are refreshed every time that a frame is processed, thus the user may not always have time to study the results. To get around this, options are provided to the user to have the system pause whenever any object is found, when an object is matched or when an object isn't matched.

The most important components of the form are as follows:

A.9.1 ToolBars

Table A - 15 Perform Recognition toolbars

Name	Function
tbStillImage	*This ToolBar allows the user to select which Mode processing should be performed in.*
tbRecognitionPicAdjustments	*This ToolBar allows the user to select the Adjust Brightness and Adjust Contrast options in Single Image Mode.*
tbCamVideoType	*This ToolBar allows the user to choose between Single Frame and Continuous Mode in Video or Camera Mode.*

A.9.2 TabPages

Table A - 16 Perform Recognition tabpages

Name	Function
tabPicture	*This TabPage displays the Single Image Mode components.*
tabCamVideo	*This TabPage displays the Video and Camera Modes.*

A.9.3 PictureBoxes

Table A - 17 Perform Recognition pictureboxes

Name	Function
picCam	*This PictureBox displays the video file or camera (device) feed.*
picCapturePreview	*This PictureBox displays the results of the currently processed frame in Video or Camera Mode.*
picPicturePreview	*This PictureBox displays the results of the currently processed image in Single Image Mode.*
picStudy	*This PictureBox displays the image currently being processed in Single Image Mode.*

Match PictureBoxes	*Each of these PictureBoxes displays an image if the field next to which it is situated is a match.*

A.9.4 Buttons

Table A - 18 Perform Recognition buttons

Name	Function
cmdActivate	*This Button activates Single Frame Video/Camera Mode or starts and stops Continuous Frame Video/Camera Mode.*
cmdPictureProcess	*This Button activates Single Image Mode.*

A.9.5 RadioButtons

Table A - 19 Perform Recognition radiobuttons

Name	Function
optArea	*This RadioButton selects the Area Single Image Mode.*
optWhole	*This RadioButton selects the Whole Image Single Image Mode.*
optNever	*This RadioButton selects the Never Pause option.*
optObject	*This RadioButton selects the Pause on Object Found option.*
optNoMatch	*This RadioButton selects the Pause on No Object Match option.*
optMatch	*This RadioButton selects the Pause on Object Match option.*

A.9.6 Timer tmContinuousMode

This Timer captures a frame from the video file or camera (device) feed and processed it every 500 ticks.

A.9.7 Labels

There are a variety of Labels on the form which are used to display the information generated during processing.

A.9.8 Procedural flow diagram

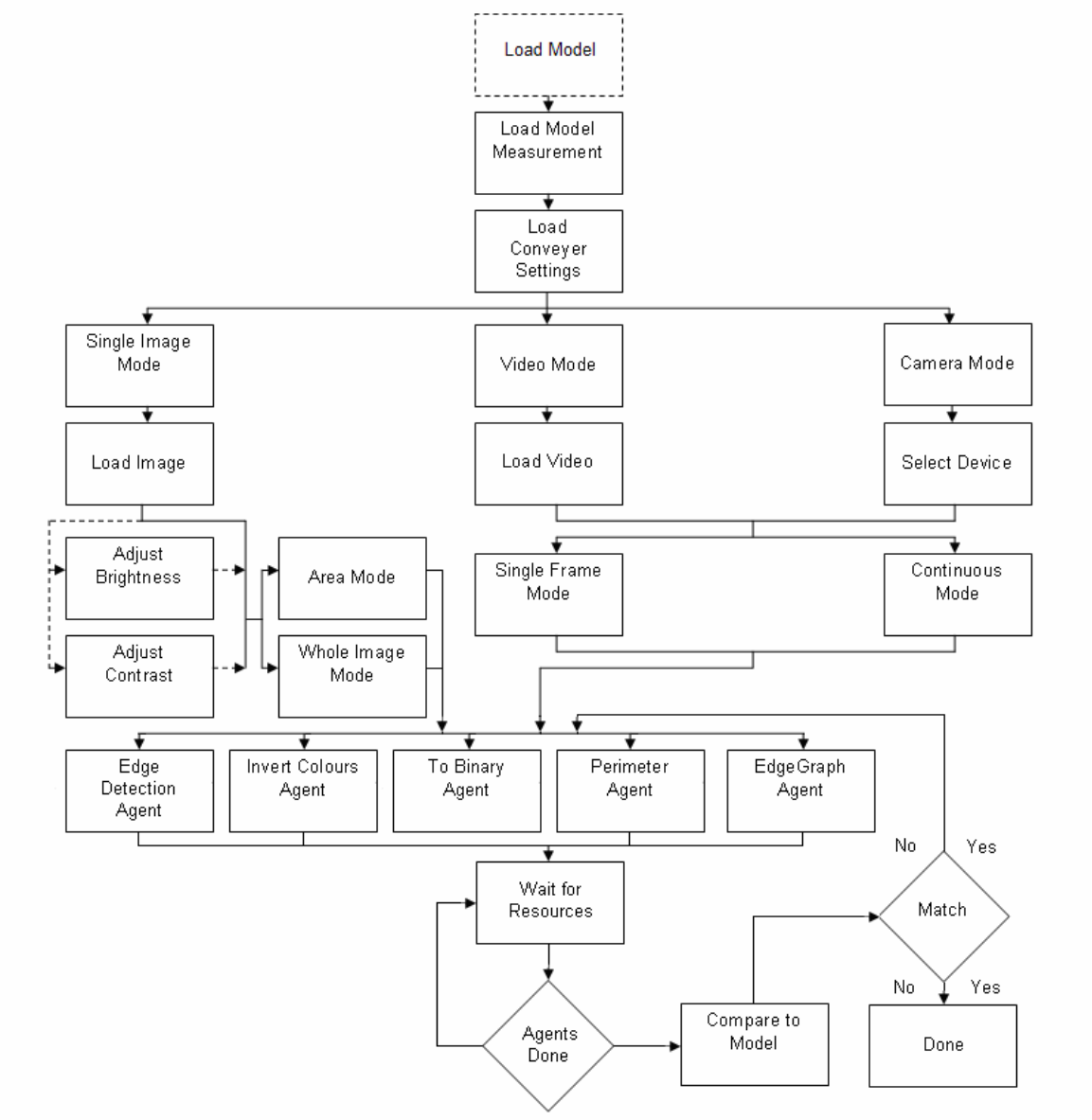

Figure A - 9 Perform Recognition procedural flow diagram

A.10. Capture Still Images

This form allows the user to capture still images from either a video file or a camera (device) feed.

The most important components of the form are as follows:

A.10.1 ToolBar tbType

This ToolBar allows the user to select which Mode processing should be performed in.

A.10.2 TabPage tabCamVideo

This TabPage displays the Video and Camera Modes.

A.10.3 PictureBoxes

Fifteen PictureBoxes display the images captured from the video file or the camera (device) feed.

A.10.4 Panel pnlCamVideo

This Panel displays the video file or camera (device) feed.

A.10.5 Buttons

Table A - 20 Capture Still Images buttons

Name	Function
cmdBurst	*This Button takes image captures until all of the PictureBoxes are filled.*
cmdCapture	*This Button takes a single image capture.*
cmdRemoveAll	*This Button clears all of the PictureBoxes.*
cmdSaveAll	*This Button saves all the images in the PictureBoxes to permanent storage.*

A.10.6 MenuItems

Table A - 21 Capture Still Images menu items

Name	Function
miRemove0	*This MenuItem removes the currently selected PictureBox's image.*
miSaveAs0	*This MenuItem saves the currently selected PictureBox's image to permanent storage and removes it from the PictureBox.*

A.10.7 Procedural flow diagram

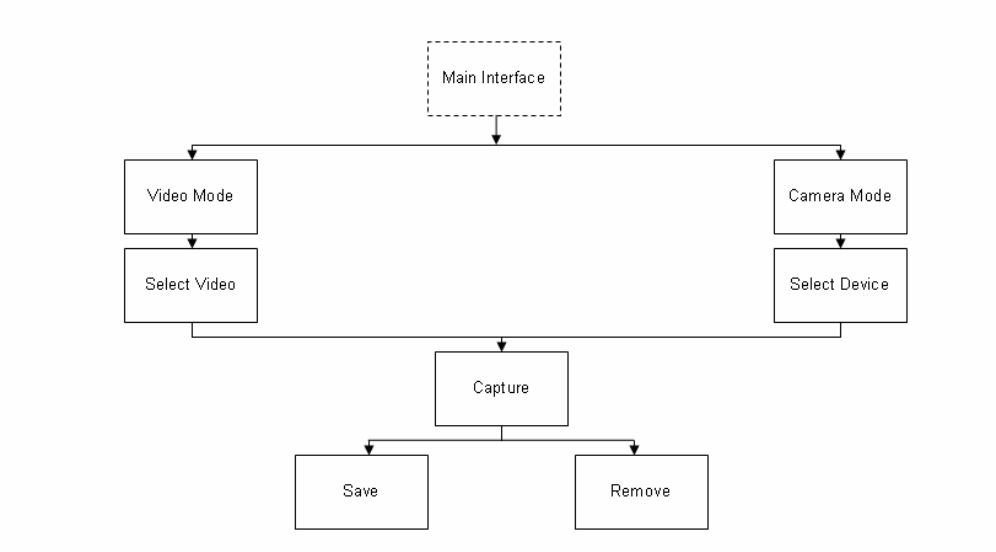

Figure A - 10 Capture Still Images procedural flow diagram

A.11 Capture Video

This form allows the user to capture a video from a camera (device) feed.

The form depends on an external .dll file. The code in this .dll file is public domain [66], but was somewhat altered by myself before compilation, so that it fits in with the rest of the project. Because the form being displayed does not appear in the project by itself, the components on it do not fall within the limits of this study.

A.11.1 Procedural flow diagram

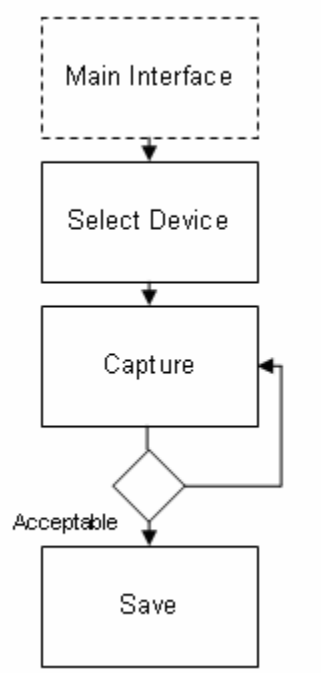

Figure A - 11 Create Video procedural flow diagram

A.12 Adjust Brightness

This form allows the user to adjust the brightness levels of the image to be processed in Single Image Mode.

The most important components of the form are as follows:

A.12.1 PictureBoxes

A.12.1.1 picOriginal

This PictureBox displays the image before adjustment.

A.12.1.2 picPreview

This PictureBox displays the image after adjustment.

A.12.2 TrackBar scrollBrightness

This TrackBar allows the user to change the brightness levels.

A.12.3 Label lblValue

This Label displays the value of the TrackBar position.

A.12.4 Buttons

Table A - 22 Adjust Brightness buttons

Name	Function
cmdApply	*This Button carries the brightness level changes over to the Single Image Mode image.*
cmdCancel	*This Button cancels any brightness level changes.*

A.12.5 Procedural flow diagram

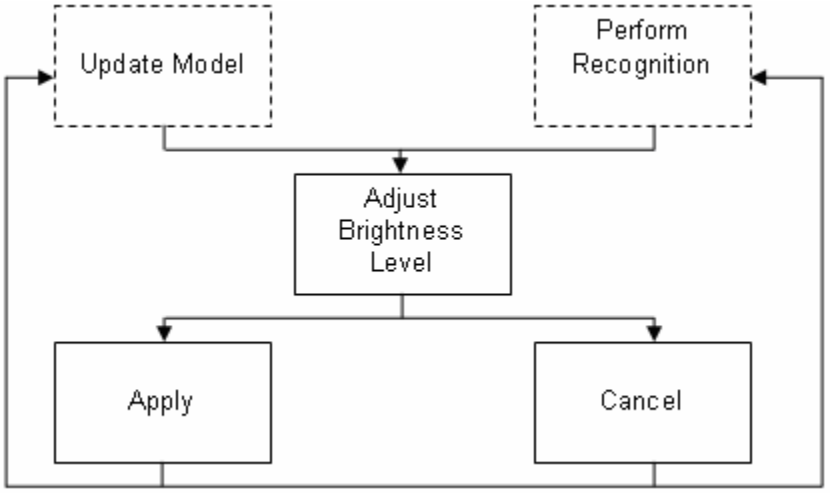

Figure A - 12 Adjust Brightness procedural flow diagram

A.13 Adjust Contrast

This form allows the user to adjust the contrast levels of the image to be processed in Single Image Mode.

The most important components of the form are as follows:

A.13 PictureBoxes

A.13.1.1 picOriginal

This PictureBox displays the image before adjustment.

A.13.1.2 picPreview

This PictureBox displays the image after adjustment.

A.13.2 TrackBar scrollContrast

This TrackBar allows the user to change the contrast levels.

A.13.3 Label lblValue

This Label displays the value of the TrackBar position.

A.13.4 Buttons

Table A - 23 Adjust Contrast buttons

Name	Function
cmdApply	*This Button carries the contrast level changes over to the Single Image Mode image.*
cmdCancel	*This Button cancels any contrast level changes.*

A.13.5 Procedural flow diagram

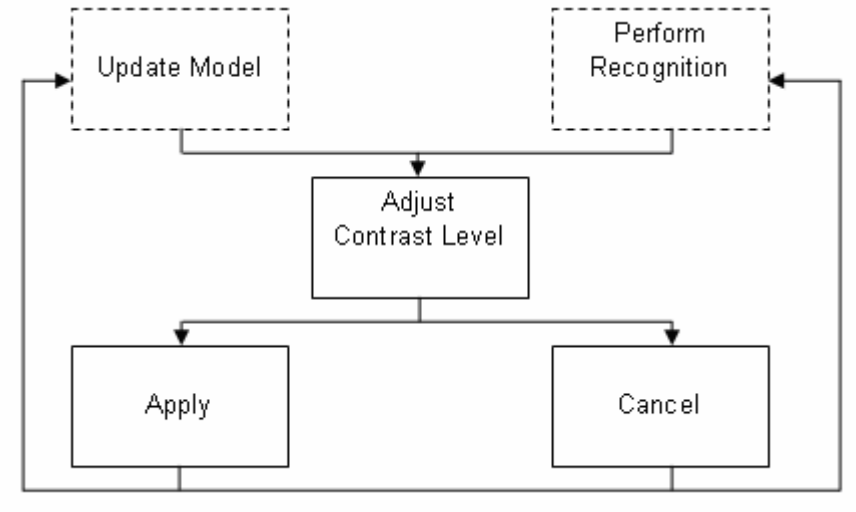

Figure A - 13 Adjust Contrast procedural flow diagram

A.14 Edge Detection Agent

This agent is responsible for finding the edges on the image being processed.

A.14.1 Procedural flow diagram

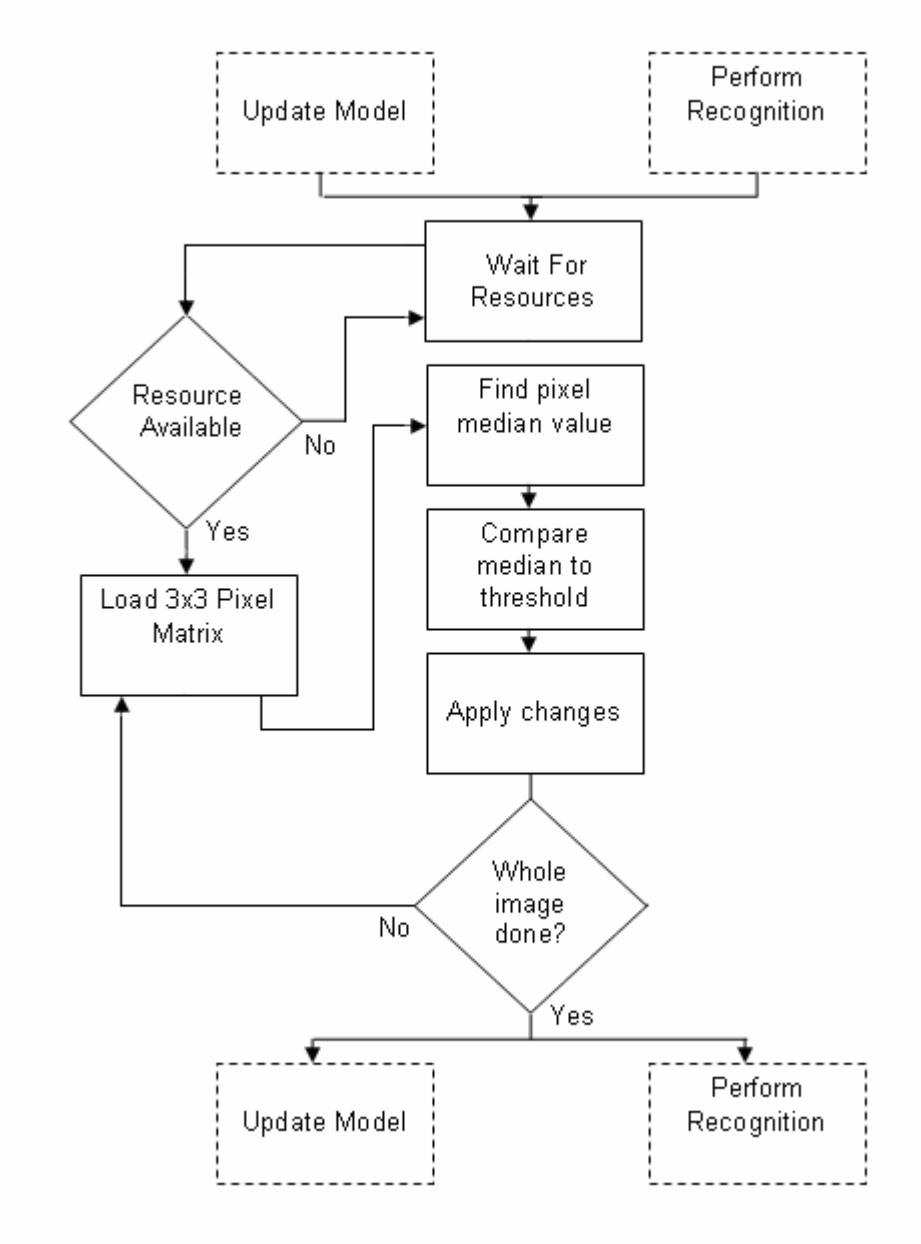

Figure A - 14 Edge Detection Agent procedural flow diagram

A.15 Invert Colours Agent

The image returned from the Edge Detection Agent consists of a dark-coloured background and the edges indicated in their original bright colours. This agent makes the background light and the edges dark.

A.15.1 Procedural flow diagram

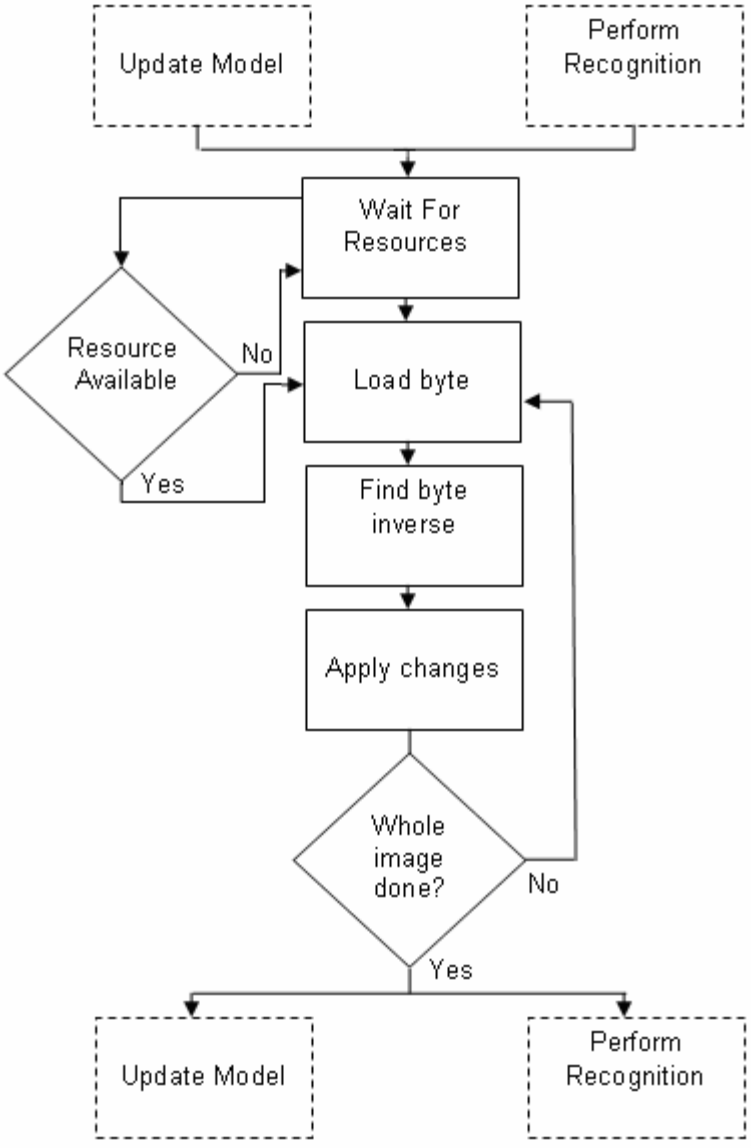

Figure A - 15 Invert Colours Agent procedural flow diagram

A.16 To Binary Agent

The image returned from the Invert Colours Agent consists of a light background and the edges indicated in darker colours. This agent makes the light colours of the background white and the darker colours of the edges black.

A.16.1 Procedural flow diagram

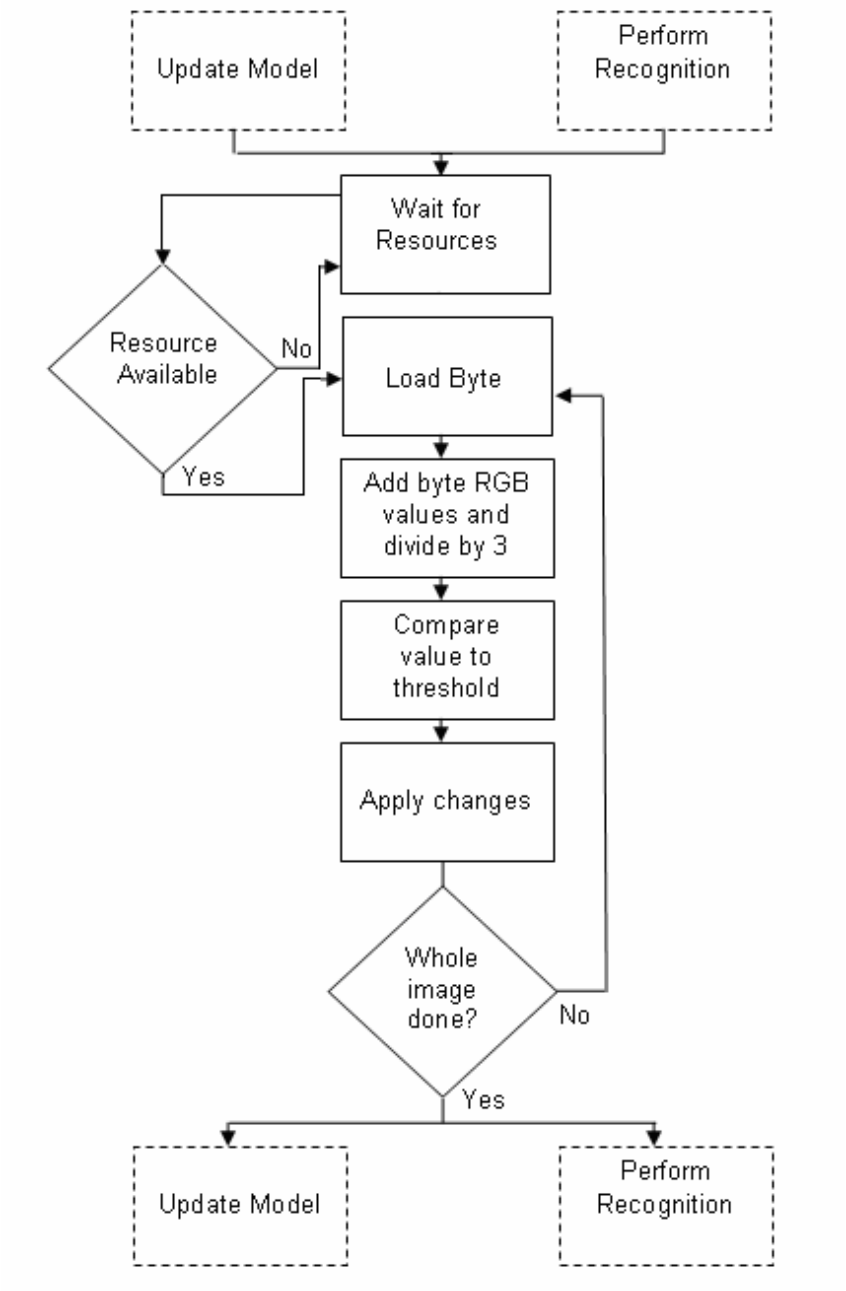

Figure A - 16 To Binary Agent procedural flow diagram

A.17 Perimeter Agent

The image returned from the To Binary Agent consists of a white background and the edges indicated in black. This agent finds a single pixel on the outside edge of an object and then traces the outline of the object until the original pixel is found again.

A.17.1 Procedural flow diagram

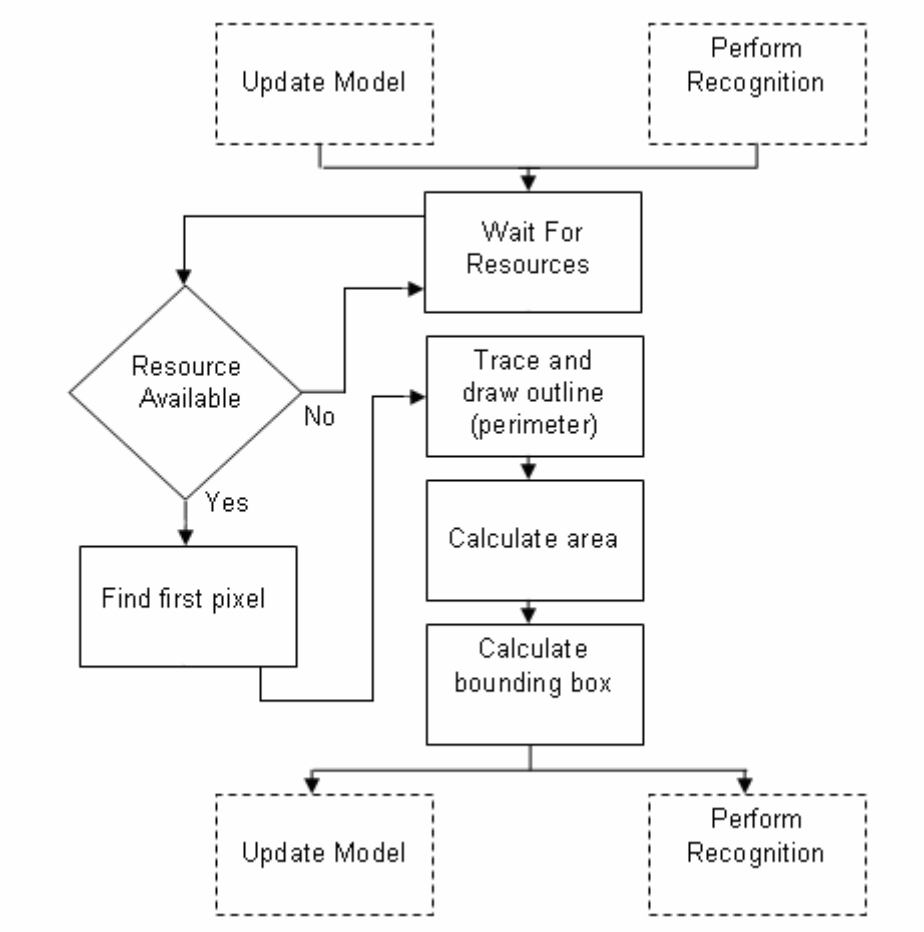

Figure A - 17 Perimeter Agent procedural flow diagram

A.18 Edge Graph Agent

The image returned from the To Binary Agent consists of a white background and the edges indicated in black. This agent runs through the middle of the object, counting the number of edges.

A.18.1 Procedural flow diagram

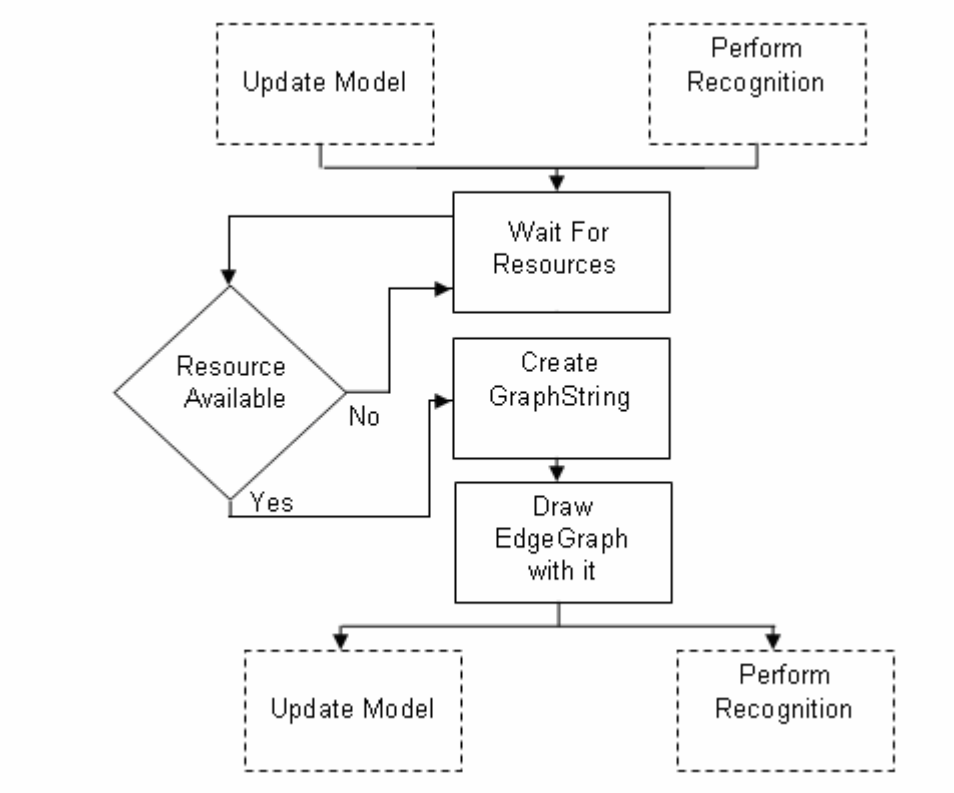

Figure A - 18 Edge Graph Agent procedural flow diagram

A.19 Remove Background Class

This class sets any pixels outside the bounding box to white.

A.19.1 Procedural flow diagram

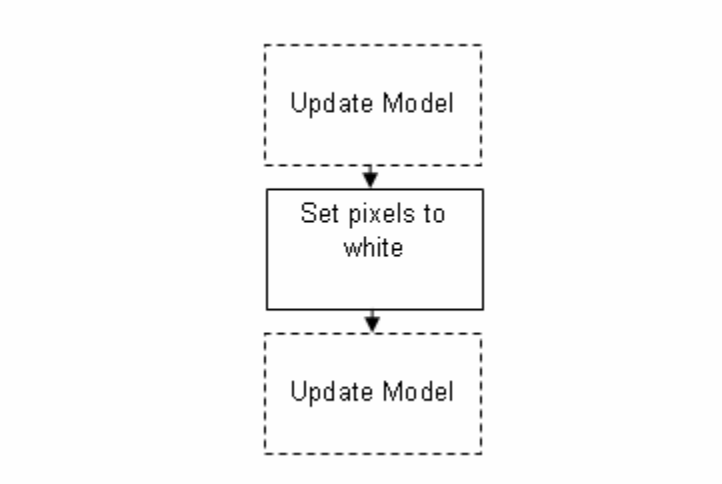

Figure A - 19 Remove Background Class procedural flow diagram

A.20 Overlay Class

This class overlays the perimeter and the edges unto the original image.

A.20.1 Procedural flow diagram

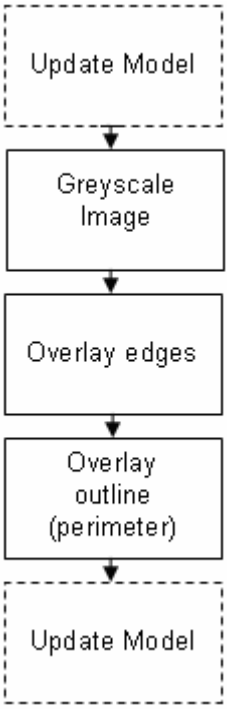

Figure A - 20 Overlay Class procedural flow diagram

Appendix C

C. RECMASTER ACCURACY TESTS

This appendix consists of preliminary tests performed using the RecMaster programme on different shaped objects. These tests were conducted to serve as a frame of reference as to prove the accuracy of the programme in performing recognition, by first building up a model and then testing images against it.

C.1 Test purpose

The test performed in this section was done by creating models of four different shaped objects and then testing the model against these shapes. The only purpose of this test is to determine the accuracy of the RecMaster programme's recognition process. By proving that the programme delivers constant results no matter what the sample size, it enables the sample sizes used in Chapter 4 to be of a smaller size.

C.2 Test Description

In order to perform this test, four shapes were selected to be used in the recognition process. These shapes, along with the Figures in which their examples appear, are listed in Table C – 1.

Table C - 1 Test shapes

	Shape	**Example**
1	Cross	Figure C - 1
2	Triangular	Figure C - 2
3	Rectangular	Figure C - 3
4	Circular	Figure C - 4

A model built up for each of these shapes using 20 representative images of each. The test was then performed by using the four models to perform recognition on the following batch of images:

1. 10 images containing cross shapes.
2. 20 images containing triangular shapes.
3. 50 images containing rectangular shapes.
4. 100 images containing circular shapes.
5. 20 empty images to act as a control.

C.3 Test results

The results of performing recognition on the batch of images are shown in Table C – 2.

Table C - 2 Recognition results

Model	Images	Positive Matches Found	Negative Matches Found	Accuracy
Cross	10	8	0	80 %
Triangle	20	17	0	85 %
Rectangle	50	43	0	86 %
Circle	100	77	0	77 %

The results, in Table C – 2, clearly indicate that the RecMaster programme consistently has a a higher than 75 % accuracy rate no matter what the sample size. The empty images, which were used as a control, did not trigger any object's to be found. It is worth noting that many of the images on which positive matches were not found, were not the result of no object being found, but rather the result of the model not being well-enough trained. Table C – 3 gives a listing of the images on which objects were found, but on which no matches were made. Included in Table C – 3 is the amount of objects in which the programme

actually extracted the right shape, but the model was not robust enough to recognize the object's features.

Table C - 3 Model deficiencies

Model	Number of images on which no matches were made	Number of images in which the model was not robust enough	Final recognition percentage for accurately trained model
Cross	2	0	80%
Triangle	3	0	85%
Rectangle	7	4	94%
Circle	23	17	94%

As the results of Table C – 3 indicate, with a well-trained model, the accuracy of the RecMaster programme never dips below 80%.

The results of these tests prove that the RecMaster programme accurately performs recognition, no matter how large or small the sample size. In a controlled environment, the main factor influencing recognition is the robustness of the model that is being used. The development of the model is entirely up to the user and rests on his or her decision to make use of quality images and to use them in sufficient quantity so as to have a broad enough range of values to meet the model criteria. The quantity of images used will vary from model type to model type.

Addendum B

Source Code

This section contains all of the code generated for the RecMaster project. It is split up into 2 sections:

- Forms
- Code Pages

The Forms section contains all of the source code for the forms in the project. The Form Component Initialization section has however been left out. The reason for this is that this section of code is mostly compiler generated with minimal changes to accommodate some of my own function calls. Because of this, it does not justify adding this code to the Addendum as it would almost double the size of it.

The Code Section consists of the Agents and other classes, written to accommodate the image acquisition process.

1. Forms

1.1 Splash Screen/ Startup Form (frmSplash)

This form is displayed at program startup. It functions as a means of telling the user which program is being executed as well as who the author of the program is. The screen will be displayed for about 2 seconds before loading the program's main interface screen.

1.1.1 System References

```
//  Declaration of References
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
```

1.1.2 Form Items and Global Variables

```
namespace AntsNest
{
/// <summary>
/// Summary description for frmSplash.
/// </summary>
public class frmSplash : System.Windows.Forms.Form
{
        private System.Windows.Forms.Timer tmSplash;
        private System.ComponentModel.IContainer components;
        private System.Windows.Forms.GroupBox groupBox1;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.PictureBox pictureBox1;
        private System.Windows.Forms.Label label1;
        //  Integer variable used to keep track of how long
        //  the form has been displayed.
        private int counter;
```

1.1.3 Form Initialization

```
//  Startup subroutine for frmSplash.
public frmSplash()
{
        InitializeComponent();
        //  Initializes the counter and starts the timer.
        counter = 0;
        tmSplash.Enabled = true;
}
```

1.1.4 Program Start Point

```
/// <summary>
/// The main entry point for the application.
/// </summary>
[STAThread]
static void Main()
{
        // Application start point
        Application.Run(new frmSplash());
}
```

1.1.5 Form Designer Generated Code

```
/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
        if( disposing )
        {
                if(components != null)
                {
                        components.Dispose();
                }
        }
        base.Dispose( disposing );
}

#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
```

```
{
        //  Component code omitted.
}
#endregion
```

1.1.6 Splash Screen Display Timer

```
//  This subroutine is called by the timer whenever its time interval
//  has been reached (100)
private void tmSplash_Tick(object sender, System.EventArgs e)
{
        //  Increments the counter and tests whether it has reached 20.
        //  When 20 is reached frmMain is loaded and frmSplash is hidden.
        counter++;
        if (counter == 20)
        {
                tmSplash.Enabled = false;
                frmMain MainForm = new frmMain();
                MainForm.Show();
                this.Hide();
        }
}
}
}
```

1.2 Main Program Interface (frmMain)

This form functions as the program's main interface. All of the available features of the program are available for selection through this form. It consists of a main menu as well as a toolbar, which will present the user with all available options. All program features, except for dialog boxes, will be loaded as MDI children to this form.

1.2.1 System References

```
//  Declaration of References
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.IO;
using System.Diagnostics;
```

1.2.2 Form Items and Global Variables

```
namespace AntsNest
{
/// <summary>
/// Summary description for frmMain.
/// </summary>
public class frmMain : System.Windows.Forms.Form
{
        private System.Windows.Forms.MenuItem mnuFile;
        private System.Windows.Forms.MenuItem mnuTraining;
        private System.Windows.Forms.MenuItem mnuUpdateModel;
        private System.Windows.Forms.MenuItem mnuImageRecognition;
        private System.Windows.Forms.MenuItem mnuNewModel;
        private System.Windows.Forms.MenuItem mnuExit;
        private System.ComponentModel.IContainer components;
        public System.Windows.Forms.ProgressBar pbProgress;
        private System.Windows.Forms.Panel pnlMain;
        private System.Windows.Forms.MenuItem mnuViewModel;
        public System.Windows.Forms.ImageList ilButtons;
        private System.Windows.Forms.ToolBarButton tbbNewModel;
        private System.Windows.Forms.ToolBarButton tbbUpdateModel;
        private System.Windows.Forms.ToolBarButton tbbRecognition;
        private System.Windows.Forms.ToolBar tbMain;
```

```
        private System.Windows.Forms.ToolBarButton tbbViewModel;
        private System.Windows.Forms.ToolBarButton tbbDeleteModel;
        private System.Windows.Forms.MainMenu MainMenu;
        private System.Windows.Forms.MenuItem mnuDeleteModel;
        private System.Windows.Forms.MenuItem menuItem1;
        private System.Windows.Forms.MenuItem mnuCreateStill;
        private System.Windows.Forms.MenuItem mnuCreateVideo;
        private System.Windows.Forms.ToolBarButton Separator;
        private System.Windows.Forms.ToolBarButton tbbCreateStills;
        private System.Windows.Forms.ToolBarButton tbbCreateVideo;
        private System.Windows.Forms.MenuItem menuItem2;
        private System.Windows.Forms.MenuItem menuItem3;
        private System.Windows.Forms.MenuItem miAbout;
        private System.Windows.Forms.MenuItem menuItem5;
        private System.Windows.Forms.ToolBarButton Separator2;
        private System.Windows.Forms.ToolBarButton
                tbbConveyerMeasurments;
        private System.Windows.Forms.MenuItem mnuDimensions;

        //  String variable used to pass the location of the Models
        //  directory to the rest of the program
        private string RootDirectory;
```

1.2.3 Form Initialization

```
//  Startup subroutine for frmMain.
public frmMain()
{
        InitializeComponent();
        //  Makes the progress bar invisible.
        pbProgress.Visible = false;
        //  Sets the value of RootDirectory to the path of the Models
        //  directory in the directory where the executable file of
        //  the application resides.
        RootDirectory = Application.StartupPath + "\\Models";
}
```

1.2.4 Form Closing

```
//  This subroutine is called when the user clicks on the "X" button in the
//  control box in the upper right-hand corner of the form.
private void frmMain_Closing(object sender,
                System.ComponentModel.CancelEventArgs e)
{
        //  Exit the program.
        Application.Exit();
```

```
}
```

1.2.5 Form Designer Generated Code

```
/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
        if( disposing )
        {
                if (components != null)
                {
                        components.Dispose();
                }
        }
        base.Dispose( disposing );
}


#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
        //  Component code omitted.
}
#endregion
```

1.2.6 Main toolbar

```
//  This subroutine functions as a switchboard to decide which button was
//  pressed on  the main toolbar (tbMain).
private void tbMain_ButtonClick(object sender,
                System.Windows.Forms.ToolBarButtonClickEventArgs e)
{
        switch (tbMain.Buttons.IndexOf(e.Button))
        {
                //  Called when the 'New Model' button is clicked on the main
                //  toolbar.
                case 0: mnuNewModel_Click(sender,e);
                        break;
                //  Called when the 'View Model' button is clicked on the main
                //  toolbar.
```

```
                case 1: mnuViewModel_Click(sender,e);
                        break;
                //  Called when the 'Delete Model' button is clicked on the main
                //  toolbar.
                case 2: mnuDeleteModel_Click(sender,e);
                        break;
                //  Called when the 'Update Model' button is clicked on the main
                //  toolbar.
                case 3: mnuUpdateModel_Click(sender, e);
                        break;
                //  Called when the 'Perform Recognition' button is clicked on the
                //  main toolbar.
                case 4: mnuImageRecognition_Click(sender, e);
                        break;
                //  Called when the 'Create Still Image' button is clicked on the main
                //  toolbar.
                case 6: mnuCreateStill_Click(sender, e);
                        break;
                //  Called when the 'Create Video' button is clicked on the main
                //  toolbar.
                case 7: mnuCreateVideo_Click(sender, e);
                        break;
                //  Called when the 'Adjust conveyer belt dimensions' button is
                //  clicked on the main toolbar.
                case 9: mnuDimensions_Click(sender, e);
                        break;
        }
}
```

1.2.7 Start the Create New Model process

```
//  This subroutine is called when 'File -> New Model' is
//  selected on the main menu (tbMain)or the 'New Model'
//  button is clicked on the main toolbar.  It is used when the user
//  wants to create a new Recognition Model.
private void mnuNewModel_Click(object sender, System.EventArgs e)
{
        //  Calls frmNewModel to start the New Model Creation process.
        frmNewModel NewModel = new frmNewModel(this,pbProgress);
        NewModel.ShowDialog();
}
```

1.2.8 Start the View Model process

```
//  This subroutine is called when 'File -> View Model' is
//  selected on the main menu (tbMain) or the 'View Model'
//  button is clicked on the main toolbar.  It is used when the user
//  wants to see information about a specific model.
private void mnuViewModel_Click(object sender, System.EventArgs e)
{
        // String array used to store the list of directories in the
        // Models folder.  Each Folder signifies a specific model.
        string[] directorylist;
        //  Checks whether the Models directory exists.
        if (Directory.Exists(RootDirectory))
        {
                //  The Models directory exists.
                //  Assigns the list of directories to the directorylist string array
                directorylist = Directory.GetDirectories(RootDirectory);
                //  Checks whether there are any existing models/directories in the
                //  Models folder.
                if (directorylist.Length  == 0)
                {
                        //  If there are no models/directories in the Models folder,
                        //  display this message.
                        MessageBox.Show("No Model files found", "Message",
                                MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
                else
                {
                        //  If there are models/directories in the Models folder,
                        //  call frmLoadModel.
                        frmLoadModel LoadModel = new
                                        frmLoadModel(3,this,pbProgress);
                        LoadModel.ShowDialog();
                }
        }
        else
        {
                //  The Models directory does not exist.
                //  Creates the Models directory and displays a message that tells
                //  the user that there are  no models/directories currently in the
                //  Models folder.
                Directory.CreateDirectory(RootDirectory);
                MessageBox.Show("No Model files found", "Message",
                        MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
}
```

1.2.9 Start the Delete Model process

```
//  This subroutine is called when 'File -> Delete Model' is
//  selected on the main menu (tbMain) or the 'View Model'
//  button is clicked on the main toolbar.  It is used when the user
//  wants to delete a specific model.
private void mnuDeleteModel_Click(object sender, System.EventArgs e)
{
        // String array used to store the list of directories in the
        // Models folder.  Each Folder signifies a specific model.
        string[] directorylist;
        //  Checks whether the Models directory exists.
        if (Directory.Exists(RootDirectory))
        {
                //  The Models directory exists.
                //  Assigns the list of directories to the directorylist string array
                directorylist = Directory.GetDirectories(RootDirectory);

                //  Checks whether there are any existing models/directories in the
                //  Models folder.
                if (directorylist.Length  == 0)
                {
                        //  If there are no models/directories in the Models folder,
                        //  display this message.
                        MessageBox.Show("No Model files found", "Message",
                                MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
                else
                {
                        //  If there are models/directories in the Models folder,
                        //  call frmDeleteModel.
                        frmDeleteModel DeleteModel = new frmDeleteModel();
                        DeleteModel.ShowDialog();
                }
        }
        else
        {
                //  The Models directory does not exist.
                //  Creates the Models directory and displays a message that tells
                //  the user that there are no models/directories currently in the
                //  Models folder.
                Directory.CreateDirectory(RootDirectory);
                MessageBox.Show("No Model files found", "Message",
                        MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
}
```

1.2.10 Start the Update Model process

```
//  This subroutine is called when 'Training -> Update Model' is
//  selected on the main menu (tbMain) or the 'Update Model'
//  button is clicked on the main toolbar.  It is used when the user
//  wants to update an existing model with new information.
private void mnuUpdateModel_Click(object sender, System.EventArgs e)
{
        // String array used to store the list of directories in the
        // Models folder.  Each Folder signifies a specific model.
        string[] directorylist;
        //  Checks whether the Models directory exists.
        if (Directory.Exists(RootDirectory))
        {
           //  The Models directory exists.
           //  Assigns the list of directories to the directorylist string array.
           directorylist = Directory.GetDirectories(RootDirectory);

            //  Checks whether there are any existing models/directories in the
            //  Models folder.
            if (directorylist.Length  == 0)
            {
                //  If there are no models/directories in the Models folder, display
                //  this message.
                MessageBox.Show("No Model files found", "Message",
                        MessageBoxButtons.OK, MessageBoxIcon.Error);
           }
           else
           {
                //  If there are models/directories in the Models folder,
                //  call frmLoadModel.
                frmLoadModel LoadModel = new frmLoadModel(1,this,pbProgress);
                LoadModel.ShowDialog();
            }
        }
        else
        {
                //  The Models directory does not exist.
                //  Creates the Models directory and displays a message that tells
                //  the user that there are no models/directories currently in the
                //  Models folder.
                Directory.CreateDirectory(RootDirectory);
                MessageBox.Show("No Model files found", "Message",
                        MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
}
```

1.2.11 Start the Perform Recognition process

```
//  This subroutine is called when 'Perform Recognition' is
//  selected on the main menu (tbMain) or the 'Perform Recognition'
//  button is clicked on the main toolbar.  It is used when the user wants to
//  perform recognition, using one of the previously created models.
private void mnuImageRecognition_Click(object sender, System.EventArgs e)
{
        // String array used to store the list of directories in the
        // Models folder.  Each Folder signifies a specific model.
        string[] directorylist;
        //  Checks whether the Models directory exists.
        if (Directory.Exists(RootDirectory))
        {
                //  The Models directory exists.
                //  Assigns the list of directories to the directorylist string array
                directorylist = Directory.GetDirectories(RootDirectory);
                //  Checks whether there are any existing models/directories in the
                //  Models folder.
                if (directorylist.Length  == 0)
                {
                        //  If there are no models/directories in the Models folder,
                         //  display this message.
                        MessageBox.Show("No Model files found", "Message",
                                MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
                else
                {
                        //  If there are models/directories in the Models folder,
                        //  call frmLoadModel.
                        frmLoadModel LoadModel = new
                                        frmLoadModel(2,this,pbProgress);
                        LoadModel.ShowDialog();
                }
        }
        else
        {
                //  The Models directory does not exist.
                //  Creates the Models directory and displays a message that tells
                 //  the user that there are no models/directories currently in the
                 //  Models folder.
                Directory.CreateDirectory(RootDirectory);
                MessageBox.Show("No Model files found", "Message",
                        MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
}
```

1.2.12 Start the Create Still Image process

```
//  This subroutine is called when 'Training -> Create Still Image' is
//  selected on the main menu (tbMain) or the 'Create Still Image'
//  button is clicked on the main toolbar.  It is used when the user
//  wants to capture still images from a video file or camera feed.
private void mnuCreateStill_Click(object sender, System.EventArgs e)
{
        //  Calls frmCreateStills.
        frmCreateStills CreateStills = new frmCreateStills();
        CreateStills.MdiParent = this;
        CreateStills.Show();
}
```

1.2.13 Start the Create Video process

```
//  This subroutine is called when 'Training -> Create Video' is
//  selected on the main menu (tbMain) or the 'Create Video from Camera Feed'
//  button is clicked on the main toolbar.  It is used when the user
//  wants to record a video from the camera feed.
private void mnuCreateVideo_Click(object sender, System.EventArgs e)
{
        //  Changes the cursor to an hourglass; to indicate processing.
        Cursor.Current = Cursors.WaitCursor;
        //  Loads the CaptureNET .dll file's MainForm.  The program sometimes
        //  has issues with connected devices.  Catching the exception ensures
        //  that the program doesn't bomb out unexpectedly, but rather displays an
        //  appropriate message.
        try
        {
                CaptureNET.MainForm MainForm = new CaptureNET.MainForm();
                MainForm.MdiParent = this;
                MainForm.Show();
        }
        catch(Exception exception)
        {
                MessageBox.Show("An error was encountered while trying to load
                        one of the capture devices.","Message",
                        MessageBoxButtons.OK,MessageBoxIcon.Information);
        }

        //  Changes the cursor back to the normal arrow cursor.  This indicates
        //  that processing is completed.
        Cursor.Current = Cursors.Arrow;
}
```

1.2.14 Start the Set Conveyer Dimensions process

```
//  This subroutine is called when 'File -> Set Dimensions' is
//  selected on the main menu (tbMain) or the 'Set Conveyer Dimensions'
//  button is clicked on the main toolbar.  It is used when the user
//  wants to map the real-world conveyer specifications to the camera view.
private void mnuDimensions_Click(object sender, System.EventArgs e)
{
        //  Calls frmDimensions
        frmDimensions Dimensions = new frmDimensions();
        Dimensions.MdiParent = this;
        Dimensions.Show();
}
```

1.2.15 Show the About Box

```
//  This subroutine is called when 'File -> About' is
//  selected on the main menu (tbMain). It is used to convey information
//  about the program and the author, to the user.
private void miAbout_Click(object sender, System.EventArgs e)
{
        MessageBox.Show("This Program was written by Bertram Haskins\n" +
                "as part of a Master's Degree in Information Technology.\n" +
                "It was completed in March of 2006.  This is version 1.0",
                "About RecMaster",MessageBoxButtons.OK,
                MessageBoxIcon.Information);
}
```

1.2.16 Exit the Program

```
//  This subroutine is called when 'File -> Exit' is
//  selected on the main menu (tbMain).  It is used when the user
//  wants to exit the program.
private void mnuExit_Click(object sender, System.EventArgs e)
{
        //  Exit the program.
        Application.Exit();
}

}
}
```

1.3 Create a New Model (frmNewModel)

This form is called whenever the user needs to create a new Model. Basically all this form does is provide the user with a list of already existing models and allows the user to enter the name of the new Model which needs to be created.

1.3.1 System References

```
//  Declaration of References
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.IO;
```

1.3.2 Form Items and Global Variables

```
namespace AntsNest
{
/// <summary>
/// Summary description for frmNewModel.
/// </summary>
public class frmNewModel : System.Windows.Forms.Form
{
        private System.Windows.Forms.TextBox txtModel;
        private System.Windows.Forms.Button cmdCreate;
        private System.Windows.Forms.Button cmdCancel;
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.Container components = null;
        //  String variable used to pass the location of the Models
        //  directory to the rest of the program
        private string RootDirectory;
        //  Form Variable used to store a reference to the Main / Startup
        //  form.
        private System.Windows.Forms.Form Main;
        //  Progress Bar variable used to store a reference to the
        //  progress bar on the Main / Startup form.
        private System.Windows.Forms.ProgressBar pbProgress;
        private System.Windows.Forms.GroupBox groupBox1;
        private System.Windows.Forms.GroupBox groupBox2;
        private System.Windows.Forms.ListBox lbModels;
```

1.3.3 Form Initialization

```
//  Startup Subroutine for frmNewModel.
//  Takes 2 arguments: - A reference to frmMain
//                     - A reference to the progress bar on frmMain
public frmNewModel(System.Windows.Forms.Form a,
                   System.Windows.Forms.ProgressBar pb,
                   System.Windows.Forms.Label lblFile, Label lblModel)
{
        InitializeComponent();
        //  Assign passed variables to their local counterparts.
        Main = a;
        pbProgress = pb;
        lblCurrentFile = lblFile;
        lblCurrentModel = lblModel;
        //  Sets the value of RootDirectory to the path of the Models
        //  directory in the directory where the executable file of
        //  the application resides.
        RootDirectory = Application.StartupPath + "\\Models";
        // String array used to store the list of directories in the
        // Models folder.  Each Folder signifies a specific model.
        string[] directorylist;
        //  String value used as a temporary storage variable for
        //  values read off the directorylist array.
        string listvalue;
        //  Assigns the list of directories to the directorylist string array
        directorylist = Directory.GetDirectories(RootDirectory);
        //  Runs through all of the items in the directorylist array and
        //  adds the entries (Model names) to the listbox named lbModels.
        foreach(string ModelName in directorylist)
        {
                listvalue = ModelName.Substring(RootDirectory.Length + 1);
                lbModels.Items.Add(listvalue);
        }
        //  Sets the focus to the textbox, so the user can start typing a new
        //  model name.
        txtModel.Focus();
}
```

1.3.4 Form Designer Generated Code

```
/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
	if( disposing )
	{
		if(components != null)
		{
			components.Dispose();
		}
	}
	base.Dispose( disposing );
}

#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
	//  Component code omitted.
}
#endregion
```

1.3.5 Create Button

```
//  This subroutine is called when the 'Create' button is clicked.
//  It closes frmNewModel, creates a new model and calls the next
//  form.
private void cmdCreate_Click(object sender, System.EventArgs e)
{
	//  Makes sure that no model / directory exists with the
	//  name the user entered into the txtModel textbox.
	if (!Directory.Exists(RootDirectory + "\\" + txtModel.Text))
	{
		//  Creates a directory in the Models directory, with the name of the
		//  new model.
		Directory.CreateDirectory(RootDirectory + "\\" + txtModel.Text);
		//  Creates the new model file in the newly created model directory.
		File.Create(RootDirectory + "\\" + txtModel.Text + "\\" +
				txtModel.Text + ".mdl");
		//  Creates the model configuration file to be used by the
```

```
                //  recognition process.  It sets all of the options, except
                //  the first one, to false;
                TextWriter tw;
                tw = new StreamWriter(RootDirectory + "\\" + txtModel.Text + "\\" +
                                        txtModel.Text + ".mcf");
                tw.WriteLine("True");
                for (int i = 0; i < 8; i++)
                     tw.WriteLine("False");
                tw.Close();
                //  Calls frmUpdateModel and closes frmNewModel.
                frmUpdateModel UpdateModel = new
                       frmUpdateModel(RootDirectory + "\\" + txtModel.Text, "New
                       Model",txtModel.Text,1,pbProgress,
                       lblCurrentFile,lblCurrentModel);
                UpdateModel.MdiParent = Main;
                UpdateModel.Show();
                this.Close();
        }
        else
                //  Displays this message when the new model name, typed into the
                //  txtModel textbox, already exists.
                MessageBox.Show("A model with this name already exists",
                                        "Message", MessageBoxButtons.OK,
                                         MessageBoxIcon.Information);
}
```

1.3.6 Cancel Button

```
//  This subroutine is called when the 'Cancel' button is clicked.
//  It closes frmNewModel, without creating a new model.
private void cmdCancel_Click(object sender, System.EventArgs e)
{
        //  Closes the form.
        this.Close();
}

}
}
```

1.4 Delete an Existing Model (frmDeleteModel)

This form is called whenever the user needs to delete an existing Model. Basically all this form does is provide the user with a list of existing models and allows the user to select the name of the Model which needs to be deleted.

1.4.1 System References

```
//  Declaration of References
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.IO;
```

1.4.2 Form Items and Global Variables

```
namespace AntsNest
{
/// <summary>
/// Summary description for frmDeleteModel.
/// </summary>
public class frmDeleteModel : System.Windows.Forms.Form
{
        private System.Windows.Forms.GroupBox groupBox2;
        private System.Windows.Forms.ListBox lbModels;
        private System.Windows.Forms.GroupBox groupBox1;
        private System.Windows.Forms.TextBox txtModel;
        private System.Windows.Forms.Button cmdCancel;
        private System.Windows.Forms.Button cmdDelete;
        private System.Windows.Forms.TextBox txtLastModified;
        private System.Windows.Forms.Label label3;
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.Container components = null;

        //  String variable used to pass the location of the Models
        //  directory to the rest of the program
        private string RootDirectory;
```

1.4.3 Form Initialization

```
//  Startup Subroutine for frmDeleteModel.
public frmDeleteModel()
{
        InitializeComponent();
        //  Sets the value of RootDirectory to the path of the Models
        //  directory in the directory where the executable file of
        //  the application resides.
        RootDirectory = Application.StartupPath + "\\Models";
        if (!Directory.Exists(RootDirectory))
        Directory.CreateDirectory(RootDirectory);
        // String array used to store the list of directories in the
        // Models folder.  Each Folder signifies a specific model.
        string[] directorylist;
        //  String value used as a temporary storage variable for
        //  values read off the directorylist array.
        string listvalue;
        //  Assigns the list of directories to the directorylist string array
        directorylist = Directory.GetDirectories(RootDirectory);
        //  Runs through all of the items in the directorylist array and
        //  adds the entries (Model names) to the listbox named lbModels.
        foreach(string ModelName in directorylist)
        {
                listvalue = ModelName.Substring(RootDirectory.Length + 1);
                lbModels.Items.Add(listvalue);
        }
        cmdDelete.Enabled = false;
        //  Sets the focus to the textbox, so the user can start typing a new
        //  model name.
        txtModel.Focus();
}
```

1.4.4 Form Designer Generated Code

```
/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
        if( disposing )
        {
                if(components != null)
                {
                        components.Dispose();
                }
```

```
        }
        base.Dispose( disposing );
}

#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
        //  Component code omitted.
}
#endregion
```

1.4.5 Delete Button

```
//  This subroutine is called whenever the Delete button is clicked.

private void cmdDelete_Click(object sender, System.EventArgs e)
{
        //  Ensures that the user really wants to delete the model.  If the user
        //  confirms; the Model is deleted.
        if (MessageBox.Show("Are you sure you want to delete " + txtModel.Text
            + "?", "Message", MessageBoxButtons.YesNo,
            MessageBoxIcon.Question) == DialogResult.Yes)
        {
                Directory.Delete(RootDirectory + "\\" + txtModel.Text + "\\",true);
                string[] directorylist;
                //  String value used as a temporary storage variable for
                //  values read off the directorylist array.
                string listvalue;
                //  Assigns the list of directories to the directorylist string array
                directorylist = Directory.GetDirectories(RootDirectory);
                //  Runs through all of the items in the directorylist array and
                //  adds the entries (Model names) to the listbox named lbModels.
                lbModels.Items.Clear();
                foreach(string ModelName in directorylist)
                {
                        listvalue = ModelName.Substring(RootDirectory.Length + 1);
                                lbModels.Items.Add(listvalue);
                }
                //  Sets the focus to the textbox, so the user can start typing a new
                //  model name.
                lbModels.Invalidate();
                txtModel.Focus();
```

```
                // Clears all of the textboxes and disables the Delete button.
                txtModel.Text = null;
                txtLastModified.Text = null;
                cmdDelete.Enabled = false;
                // Displays an informative message and closes the form if all
                // Models have been deleted.
                if (lbModels.Items.Count == 0)
                {
                        MessageBox.Show("All models have been deleted",
                                "Message", MessageBoxButtons.OK,
                                MessageBoxIcon.Information);
                        this.Close();
                }
        }
}
```

1.4.6 Cancel Button

```
// This subroutine is called when the Cancel button is clicked.
private void cmdCancel_Click(object sender, System.EventArgs e)
{
        // Closes the form.
        this.Close();
}
}
}
```

1.5 Load an Existing Model (frmLoadModel)

This form is called when a Model needs to be chosen to update, perform recognition on or view. It acts as a multi-purpose middle ground between the program's main interface and a form that needs to have a Model's measurements loaded.

1.5.1 System References

```
//  Declaration of References
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.IO;
```

1.5.2 Form Items and Global Variables

```
namespace AntsNest
{
/// <summary>
/// Summary description for frmLoadModel.
/// </summary>
public class frmLoadModel : System.Windows.Forms.Form
{
        private System.Windows.Forms.Button cmdLoad;
        private System.Windows.Forms.Button cmdCancel;
        private System.Windows.Forms.TextBox txtModel;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.TextBox txtLastModified;
        private System.Windows.Forms.GroupBox groupBox1;
        private System.Windows.Forms.ListBox lbModels;
        private System.Windows.Forms.ListBox lbSelected;
        private System.Windows.Forms.Button cmdAdd;
        private System.Windows.Forms.Button cmdClear;
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.Container components = null;
        //  String variable used to pass the location of the Models
        //  directory to the rest of the program
        private string RootDirectory;
```

```
        //  Integer variable used by this form to store a value
        //  that identifies which subroutine called this form.
        private int CalledBy;
        //  Form Variable used to store a reference to the Main / Startup
        //  form.
        private System.Windows.Forms.Form Main;
        //  Progress Bar variable used to store a reference to the
        //  progress bar on the Main / Startup form.
        private System.Windows.Forms.ProgressBar pbProgress;
        //  Integer variable used to keep track of whether lbSelected
        //  was clicked or not.
        private int ItemSelected;
```

1.5.3 Form Initialization

```
//  Startup Subroutine for frmLoadModel.
//  Takes 3 arguments:  -  A reference to which subroutine called this form
//                      (1: Update Model, 2: Start Recognition,
//                       3: View Model)
//                  -  A reference to frmMain
//                  -  A reference to the progress bar on frmMain
public frmLoadModel(int caller, System.Windows.Forms.Form a,
                            System.Windows.Forms.ProgressBar pb)
{
        InitializeComponent();
        //  Assign passed variables to their local counterparts.
        CalledBy = caller;
        Main = a;
        pbProgress = pb;
        //  Initialize variable.
        ItemSelected = 0;
        // If called by the Update Model subroutine on the Main Form
        if (CalledBy == 1)
        {
                this.Icon = new Icon(Application.StartupPath + "\\Icons\\Update
                                Model.ico");
                this.Text = "Select a model to update";
        }
        // If called by the Start Recognition subroutine on the Main Form
        if (CalledBy == 2)
        {
                this.Icon = new Icon(Application.StartupPath + "\\Icons\\Perform
                                Recognition.ico");
                this.Text = "Select a model to perform recognition on";
                lbModels.Width = 144;
                lbSelected.Visible = true;
```

```
            cmdAdd.Visible = true;
            cmdClear.Visible = true;
            cmdAdd.Enabled = true;
            cmdClear.Enabled = false;
        }
        // If called by the View Model subroutine on the Main Form
        if (CalledBy == 3)
        {
            this.Icon = new Icon(Application.StartupPath + "\\Icons\\View
                        Model.ico");
            this.Text = "Select a model to view";
        }
        //  Sets the value of RootDirectory to the path of the Models
        //  directory in the directory where the executable file of
        //  the application resides.
        RootDirectory = Application.StartupPath + "\\Models";
        // String array used to store the list of directories in the
        // Models folder.  Each Folder signifies a specific model.
        string[] directorylist;
        //  String value used as a temporary storage variable for
        //  values read off the directorylist array.
        string listvalue;
        //  Assigns the list of directories to the directorylist string array
        directorylist = Directory.GetDirectories(RootDirectory);
        //  Runs through all of the items in the directorylist array and
        //  adds the entries (Model names) to the listbox named lbModels.
        foreach(string ModelName in directorylist)
        {
            listvalue = ModelName.Substring(RootDirectory.Length + 1);
            lbModels.Items.Add(listvalue);
        }
        //  Disable the 'Load' button until a model has been selected.
        cmdLoad.Enabled = false;
        //  Sets the focus to the textbox, so the user can start typing a new
        //  model name.
        txtModel.Focus();
}
```

1.5.4 Form Designer Generated Code

```
/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
	if( disposing )
	{
		if(components != null)
		{
			components.Dispose();
		}
	}
	base.Dispose( disposing );
}

#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
	//  Component code omitted.
}
#endregion
```

1.5.5 Load Button

```
//  This subroutine is called when the 'Load' button is clicked.
//  It closes frmLoadModel and calls the next form.
private void cmdLoad_Click(object sender, System.EventArgs e)
{
	// If called by the Update Model subroutine on the Main Form
	 if (CalledBy == 1)
	{
		//  Calls frmUpdateModel.
		frmUpdateModel UpdateModel = new
		    frmUpdateModel(RootDirectory + "\\" + txtModel.Text, "Update
		    Model", txtModel.Text, 2, pbProgress);
		UpdateModel.MdiParent = Main;
		UpdateModel.Show();
		//  Closes frmLoadModel.
		this.Close();
	}
```

```
        // If called by the Start Recognition subroutine on the Main Form
        if (CalledBy == 2)
        {
                //  Assigns the items in lbSelected to a string array.
                 string[] ModelList = new string[lbSelected.Items.Count];
                for(int i = 0; i < lbSelected.Items.Count; i++)
                        ModelList[i] = lbSelected.Items[i].ToString();
                //  Calls frmRecognition.
                frmRecognition Recognition = new
                     frmRecognition(RootDirectory,txtModel.Text, ModelList);
                Recognition.MdiParent = Main;
                Recognition.Show();
                 //  Closes frmLoadModel.
                this.Close();
        }
         // If called by the View Model subroutine on the Main Form
        if (CalledBy == 3)
        {
                //  Calls frmViewModel.
                frmViewModel ViewModel = new frmViewModel(txtModel.Text);
                this.Hide();
                ViewModel.ShowDialog();
                //  Closes frmLoadModel.
                this.Close();
        }
}
```

1.5.6 Cancel Button

```
//  This subroutine is called when the 'Cancel' button is clicked.
//  It closes frmLoadModel, without loading a model.
private void cmdCancel_Click(object sender, System.EventArgs e)
{
        //  Closes frmLoadModel.
        this.Close();
}
```

1.5.7 Clicked on Available Models List box

```
//  This subroutine is called when the user clicks on one of the items /
//  models listed in listbox lbModels.
private void lbModels_SelectedIndexChanged
                (object sender, System.EventArgs e)
{
        //  If not used by Recognition, enable the Load button.
        if (CalledBy != 2)
```

```
            cmdLoad.Enabled = true;
        //  Ensures the system knows whether the listbox has been clicked or not.
        ItemSelected = 1;
        //  Sets the text of textbox txtModel to match the name of the selected
        //  model.
        txtModel.Text = lbModels.SelectedItem.ToString();
        //  Makes sure that the .mdl(model)-file exists.
        if (File.Exists(RootDirectory + "\\" + txtModel.Text + "\\" + txtModel.Text +
                        ".mdl"))
        {
                // The .mdl file exists.
                //  Displays when the file was last accessed, enables the 'Load'
                //  button and sets focus to it.
                txtLastModified.Text = File.GetLastWriteTime(RootDirectory + "\\" +
                        txtModel.Text + "\\" + txtModel.Text + ".mdl").ToString();
                cmdLoad.Focus();
        }
        else
        {
                //  The .mdl file does not exist.

                //  Display an appropriate message and makes sure that the 'Load'
                //  is disabled.
                txtLastModified.Text = "Model descriptor not found";
                cmdLoad.Enabled = false;
        }
}
```

1.5.8 Clicked on Selected Models List box

```
//  This subroutine is called when the user clicks on one of the items /
//  models listed in listbox lbSelected.
private void lbSelected_SelectedIndexChanged(object sender,
                System.EventArgs e)
{
        //  Makes sure the Clear button is enabled, so that any selected Models
        //  can be removed again.
        cmdClear.Enabled = true;
}
```

1.5.9 Add Button

```
//  This subroutine is used to add the selected item from lbModels to
//  lbSelected.
private void cmdAdd_Click(object sender, System.EventArgs e)
{
        if (ItemSelected == 1)
        {
                //  Makes sure the Load button is enabled when at least one
                //  Model has been selected
                cmdLoad.Enabled = true;
                //  Add the selected item to lbSelected.
                lbSelected.Items.Add(lbModels.SelectedItem.ToString());
                //  Makes sure that no more than 4 Models can be selected.
                if (lbSelected.Items.Count == 4)
                        cmdAdd.Enabled = false;
        }
        else
                MessageBox.Show("Please select one of the Models","Message",
                                MessageBoxButtons.OK,
                                MessageBoxIcon.Information);
}
```

1.5.10 Clear Button

```
//  This subroutine is used to clear the selected item from lbSelected.
private void cmdClear_Click(object sender, System.EventArgs e)
{
        //  Remove the selected item from lbSelected.
        lbSelected.Items.RemoveAt(lbSelected.SelectedIndex);
        //  Makes sure that at least one model is selected.
        if (lbSelected.Items.Count == 0)
        {
                cmdLoad.Enabled = false;
        }
        // Makes sure that the Clear button cannot be clicked if
        // no Model is selected in lbSelected.
        cmdClear.Enabled = false;
}

}
}
```

1.6 Update a Model (frmUpdateModel)

This form is called whenever the user needs to update a Model. It is operated either in Single Image, Batch Image, Video or Camera Mode. The measurements made on this form are saved and later used to match objects found in recognition mode.

1.6.1 System References

```
//  Declaration of References
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.IO;
using System.Drawing.Imaging;
using System.Runtime.InteropServices;
using System.Threading;
using WIALib;
using WIAVIDEOLib;
using Microsoft.DirectX.AudioVideoPlayback;
```

1.6.2 Form Items and Global Variables

```
namespace AntsNest
{
/// <summary>
/// Summary description for frmUpdateModel.
/// </summary>
public class frmUpdateModel : System.Windows.Forms.Form
{
        private System.Windows.Forms.ToolBar tbUpdateModel;
        private System.Windows.Forms.OpenFileDialog ofdPicture;
        private System.ComponentModel.IContainer components;
        private System.Windows.Forms.ImageList ilButtons;
        private System.Windows.Forms.TabPage tabNormal;
        private System.Windows.Forms.TabPage tabRoughEdges;
        private System.Windows.Forms.PictureBox picStudy;
        private System.Windows.Forms.TabControl tabPictures;
        private System.Windows.Forms.SaveFileDialog sfdCurrentGraph;
        private System.Windows.Forms.ContextMenu picCurrentEdgesMenu;
        private System.Windows.Forms.ContextMenu picRoughEdgesMenu;
        private System.Windows.Forms.MenuItem mnuRoughEdges_SaveAs;
        private System.Windows.Forms.MenuItem mnuRoughEdges_Window;
```

```
private System.Windows.Forms.SaveFileDialog sfdRoughEdges;
private System.Windows.Forms.ContextMenu OriginalAndEdgesMenu;
private System.Windows.Forms.MenuItem
        mnuOriginalAndEdges_SaveAs;
private System.Windows.Forms.MenuItem
        mnuOrginalAndEdges_Window;
private System.Windows.Forms.SaveFileDialog sfdOriginalAndEdges;
private System.Windows.Forms.MenuItem mnuCurrentEdges_Window;
private System.Windows.Forms.SaveFileDialog sfdOutline;
private System.Windows.Forms.SaveFileDialog sfdBoth;
private System.Windows.Forms.SaveFileDialog sfdOriginalAndOutline;
private System.Windows.Forms.SaveFileDialog sfdOriginalAndBoth;
private System.Windows.Forms.ContextMenu OutlineMenu;
private System.Windows.Forms.ContextMenu BothMenu;
private System.Windows.Forms.ContextMenu OriginalAndOutlineMenu;
private System.Windows.Forms.ContextMenu OriginalAndBothMenu;
private System.Windows.Forms.MenuItem mnuOutline_SaveAs;
private System.Windows.Forms.MenuItem mnuOutline_Window;
private System.Windows.Forms.MenuItem mnuBoth_SaveAs;
private System.Windows.Forms.MenuItem mnuBoth_Window;
private System.Windows.Forms.MenuItem
        mnuOriginalAndOutline_SaveAs;
private System.Windows.Forms.MenuItem
        mnuOriginalAndOutline_Window;
private System.Windows.Forms.MenuItem mnuOriginalAndBoth_SaveAs;
private System.Windows.Forms.MenuItem mnuOriginalAndBoth_Window;
private System.Windows.Forms.ToolBarButton tbbBatchMode;
private System.Windows.Forms.TabPage tabBatch;
private System.Windows.Forms.GroupBox gbQueue2;
private System.Windows.Forms.GroupBox gbQueue1;
private System.Windows.Forms.GroupBox gbQueue4;
private System.Windows.Forms.PictureBox picQueue1;
private System.Windows.Forms.ListBox lbBatchQueue1;
private System.Windows.Forms.ProgressBar pbQueue1;
private System.Windows.Forms.Button cmdNextQueue1;
private System.Windows.Forms.Button cmdPrevQueue1;
private System.Windows.Forms.Button cmdPrevQueue2;
private System.Windows.Forms.Button cmdNextQueue2;
private System.Windows.Forms.ProgressBar pbQueue2;
private System.Windows.Forms.ListBox lbBatchQueue2;
private System.Windows.Forms.PictureBox picQueue2;
private System.Windows.Forms.Button cmdPrevQueue4;
private System.Windows.Forms.Button cmdNextQueue4;
private System.Windows.Forms.ProgressBar pbQueue4;
private System.Windows.Forms.ListBox lbBatchQueue4;
private System.Windows.Forms.PictureBox picQueue4;
```

```
private System.Windows.Forms.ContextMenu picModelEdgesMenu;
private System.Windows.Forms.ToolBarButton tbbCameraMode;
private System.Windows.Forms.TabPage tabCamVideo;
private System.Windows.Forms.PictureBox picCamVideo;
private System.Windows.Forms.ToolBar tbVideoOptions;
private System.Windows.Forms.Button cmdActivate;
private System.Windows.Forms.PictureBox picCapturePreview;
private System.Windows.Forms.Label label13;
private System.Windows.Forms.Label label14;
private System.Windows.Forms.Label label16;
private System.Windows.Forms.Label label21;
private System.Windows.Forms.Label lblInfo;
private System.Windows.Forms.Label lblCamVideoYDistance;
private System.Windows.Forms.Label lblCamVideoXDistance;
private System.Windows.Forms.Label lblCamVideoArea;
private System.Windows.Forms.Label lblCamVideoPerimeter;
private System.Windows.Forms.GroupBox gbCamVideoMeasurements;
private System.Windows.Forms.Timer tmCamVideoCapture;
private System.Windows.Forms.ToolBarButton tbbSingleCapture;
private System.Windows.Forms.ToolBarButton tbbContinuousCapture;
private System.Windows.Forms.Timer tmVideo;
private System.Windows.Forms.GroupBox gbControls;
private System.Windows.Forms.Button cmdStop;
private System.Windows.Forms.Button cmdPlayPause;
private System.Windows.Forms.CheckBox chkLoop;
private System.Windows.Forms.TrackBar tbVidVolume;
private System.Windows.Forms.CheckBox chkMute;
private System.Windows.Forms.GroupBox gbVidTimes;
private System.Windows.Forms.Label lblCurrentVidTimeLabel;
private System.Windows.Forms.Label lblVidCurrent;
private System.Windows.Forms.Label lblTotalVidTimeLabel;
private System.Windows.Forms.Label lblVidTotal;
private System.Windows.Forms.TrackBar tbVideo;
private System.Windows.Forms.ToolBarButton tbbPictureBrightness;
private System.Windows.Forms.ToolBarButton tbbPictureContrast;
private System.Windows.Forms.RadioButton optWhole;
private System.Windows.Forms.RadioButton optArea;
private System.Windows.Forms.Label lblPictureYDistance;
private System.Windows.Forms.Label lblPictureXDistance;
private System.Windows.Forms.Label label15;
private System.Windows.Forms.Label label19;
private System.Windows.Forms.Label lblPictureArea;
private System.Windows.Forms.Label label24;
private System.Windows.Forms.Label lblPicturePerimeter;
private System.Windows.Forms.Label label26;
private System.Windows.Forms.PictureBox picPicturePreview;
```

```
private System.Windows.Forms.ToolBar tbPictureAdjustment;
private System.Windows.Forms.Button cmdPictureProcess;
private System.Windows.Forms.Label lblPictureInfo;
private System.Windows.Forms.GroupBox gbPictureMeasurements;
private System.Windows.Forms.GroupBox gbProcessType;
private System.Windows.Forms.Button cmdFull;
private System.Windows.Forms.Button cmdMinimal;
private System.Windows.Forms.GroupBox groupBox3;
private System.Windows.Forms.Label label7;
private System.Windows.Forms.Label label5;
private System.Windows.Forms.PictureBox picOutline;
private System.Windows.Forms.PictureBox picOriginalAndOutline;
private System.Windows.Forms.Label label4;
private System.Windows.Forms.Label label3;
private System.Windows.Forms.PictureBox picOriginalAndEdges;
private System.Windows.Forms.PictureBox picRoughEdges;
private System.Windows.Forms.GroupBox groupBox4;
private System.Windows.Forms.Label label1;
private System.Windows.Forms.PictureBox picCurrentEdges;
private System.Windows.Forms.Label label12;
private System.Windows.Forms.Label label11;
private System.Windows.Forms.Label lblCurrentArea;
private System.Windows.Forms.Label label10;
private System.Windows.Forms.Label lblCurrentPerimeter;
private System.Windows.Forms.Label label9;
private System.Windows.Forms.Label lblCurrentXDistance;
private System.Windows.Forms.Label lblCurrentYDistance;
private System.Windows.Forms.GroupBox gbQueue3;
private System.Windows.Forms.Button cmdPrevQueue3;
private System.Windows.Forms.Button cmdNextQueue3;
private System.Windows.Forms.ProgressBar pbQueue3;
private System.Windows.Forms.ListBox lbBatchQueue3;
private System.Windows.Forms.PictureBox picQueue3;
private System.Windows.Forms.GroupBox gbBatchMeasurements;
private System.Windows.Forms.Label label17;
private System.Windows.Forms.Label label18;
private System.Windows.Forms.Label label20;
private System.Windows.Forms.Label label22;
private System.Windows.Forms.Button cmdPrevious;
private System.Windows.Forms.Button cmdNext;
private System.Windows.Forms.GroupBox groupBox1;
private System.Windows.Forms.Button cmdBatchActivate;
private System.Windows.Forms.Label lblBatchInfo;
private System.Windows.Forms.Label label25;
private System.Windows.Forms.Label label27;
private System.Windows.Forms.Label label28;
```

```
private System.Windows.Forms.Label label29;
private System.Windows.Forms.Label label30;
private System.Windows.Forms.Label label31;
private System.Windows.Forms.Label lblBatchYDistanceLow;
private System.Windows.Forms.Label lblBatchXDistanceLow;
private System.Windows.Forms.Label lblBatchAreaLow;
private System.Windows.Forms.Label lblBatchPerimeterLow;
private System.Windows.Forms.Label lblBatchYDistanceHigh;
private System.Windows.Forms.Label lblBatchXDistanceHigh;
private System.Windows.Forms.Label lblBatchAreaHigh;
private System.Windows.Forms.Label lblBatchPerimeterHigh;
private System.Windows.Forms.PictureBox picBatchGraphs;
private System.Windows.Forms.Label lblBatchEdgesNumber;
private System.Windows.Forms.ContextMenu cmPicturePreview;
private System.Windows.Forms.MenuItem miPictureShow;
private System.Windows.Forms.ToolBarButton tbbVideoMode;
private System.Windows.Forms.Label label2;
private System.Windows.Forms.Label lblPictureEdges;
private System.Windows.Forms.Label label8;
private System.Windows.Forms.Label label23;
private System.Windows.Forms.Label lblBatchEdgesHigh;
private System.Windows.Forms.Label lblBatchEdgesLow;
private System.Windows.Forms.Label label32;
private System.Windows.Forms.Label lblCurrentEdges;
private System.Windows.Forms.Label lblCamVideoEdges;
private System.Windows.Forms.Label label33;
private System.Windows.Forms.ToolBarButton toolBarButton1;
private System.Windows.Forms.ToolBarButton tbbImageMode;
private System.Windows.Forms.Button cmdAccept;
private System.Windows.Forms.Button cmdAcceptCamVideo;
private System.Windows.Forms.ImageList ilBrightnessContrast;
private System.Windows.Forms.Button cmdAcceptBatch;
private System.Windows.Forms.MenuItem
        mnuPicModelEdgesOwnWindow;
private System.Windows.Forms.GroupBox gbCamControls;
private System.Windows.Forms.Button cmdCamPlayPause;
private System.Windows.Forms.GroupBox gbPerimeter;
private System.Windows.Forms.Label label6;
private System.Windows.Forms.Label label36;
private System.Windows.Forms.Label label37;
private System.Windows.Forms.Label label38;
private System.Windows.Forms.NumericUpDown numMin;
private System.Windows.Forms.NumericUpDown numMax;
private System.Windows.Forms.ImageList ilControls;
```

```
//  Integer variables used in Batch processing to keep track of the
//  variablesin the first queue.
private int PerimeterLowQueue1,PerimeterHighQueue1,
            AreaLowQueue1,AreaHighQueue1,
            ShortestLowQueue1,ShortestHighQueue1,
            LongestLowQueue1,LongestHighQueue1,
            EdgesLowQueue1, EdgesHighQueue1;
//  Integer variables used in Batch processing to keep track of the
//  variables in the second queue.
private int PerimeterLowQueue2,PerimeterHighQueue2,
            AreaLowQueue2,AreaHighQueue2,
            ShortestLowQueue2,ShortestHighQueue2,
            LongestLowQueue2,LongestHighQueue2,
            EdgesLowQueue2, EdgesHighQueue2;
//  Integer variables used in Batch processing to keep track of the
//  variables in the third queue
private int PerimeterLowQueue3,PerimeterHighQueue3,
            AreaLowQueue3,AreaHighQueue3,
            ShortestLowQueue3,ShortestHighQueue3,
            LongestLowQueue3,LongestHighQueue3,
            EdgesLowQueue3, EdgesHighQueue3;
//  Integer variables used in Batch processing to keep track of the
//  variables in the fourth queue
private int PerimeterLowQueue4,PerimeterHighQueue4,
              AreaLowQueue4,AreaHighQueue4,
              ShortestLowQueue4,ShortestHighQueue4,
              LongestLowQueue4,LongestHighQueue4,
              EdgesLowQueue4, EdgesHighQueue4;
//  The Total integer variables are used to keep track of the total number of
//  images in each queue, the Found variables are used to keep track of
//  the number of items found on images in each queue.
private int TotalQueue1, FoundQueue1, TotalQueue2, FoundQueue2,
            TotalQueue3, FoundQueue3,
            TotalQueue4, FoundQueue4;
//  Integer variable used to keep track of the number of times the user has
//  clicked on picStudy, after he or she has selected Single File Mode.
private int clickcounter;
//  Integer variable used to keep track of the points of the rectangle drawn
//  on picStudy during the Single File Mode image selection process.
private int TopLeftX, TopLeftY;
private int BottomLeftX, BottomLeftY;
private int BottomRightX, BottomRightY;
private int TopRightX, TopRightY;
private int CentreX, CentreY;
// Integer variable used, as index, to keep track of the graph
// currently displayed in picturebox picModelImages.
```

```
private int filelistIndex;
// Integer variable used, as index, to keep track of the graph
// currently displayed in picturebox picCurrentEdges.
private int CurrentlistIndex;
//  Integer Variables used to store the individual values
//  values read off of the model's .mdl file.
private int PerimeterLow, PerimeterHigh;
private int AreaLow, AreaHigh;
private int ShortestLow, ShortestHigh;
private int LongestLow, LongestHigh;
private int EdgesLow, EdgesHigh;
//  Integer variable used to keep track of which File Processing mode
//  is currently in use.  1:  Single File,  2:  Batch File
private int Mode;
//  Integer variable used, as index, to keep track of the image file
//  currently displayed in each of the Batch File Processing
//  Queue pictureboxes.
private int Queue1Current, Queue2Current, Queue3Current,
            Queue4Current;
//  Integer variable used as a flag value to activate the
//  Batch File Processing Mode startup tab page.
private int BatchModeFileSelect;
//  Integer variable used to store the amount with which the
//  progress bar on frmMain must be incremented.
private int ProgressIncrementer;
//  Bitmap variables used as temporary storage for the bitmaps
//  displayed as results during the Single and Batch File
//  Processing modes.
private Bitmap picRoughOutlineBitmap;
private Bitmap bmpOriginalAndOutline, bmpOutline, bmpOriginalAndBoth,
          bmpBoth;
private Bitmap bitmap, bmpCurrentEdges, bmpOriginalAndEdges;
//  Boolean variable used to keep track of whether the Mouse is down
//  (true) or up (false).
private Boolean bHaveMouse;
//  Point variables used to keep track of where on picStudy the user
//  started drawing a rectangle.
private Point ptOriginal = new Point();
private Point Corner1 = new Point();
//  Point variables used to keep track of where on picStudy the user
//  stopped drawing a rectangle.
private Point ptLast = new Point();
private Point Corner2 = new Point();
//  String array used to store the paths of all the files chosen
//  to be processed in Batch File Processing Mode.
private string[] filenames;
```

```
//  String variable used to store the name of the current
//  model.
private string ModelName;
// String array used to store the list of graph files in the
// specific model's folder.
private string[] filelist;
//  String array used to store the paths of all of the graphs
//  created during Batch File Processing mode.
private ArrayList BatchGraphs = new ArrayList();
//  String variable used to pass the location of the Models
//  directory to the rest of the program
private string RootDirectory;
//  Arraylist variable used to store all of the Edge graph strings
//  read off of the model's .mdl-file.
private ArrayList PreviousGraphs = new ArrayList();
//  Arraylist variable used to store all of the Edge graph strings
//  created by the EdgeGraphAgent during Single File Processing.
private ArrayList CurrentGraphs = new ArrayList();
//  Arraylist variables used to store all of the Edge graph strings
//  created by the EdgeGraphAgent during Batch File Processing.
//  There is one for each of the Queues
private ArrayList CurrentGraphsQueue1 = new ArrayList();
private ArrayList CurrentGraphsQueue2 = new ArrayList();
private ArrayList CurrentGraphsQueue3 = new ArrayList();
private ArrayList CurrentGraphsQueue4 = new ArrayList();
//  Progress Bar variable used to store a reference to the
//  progress bar on the Main / Startup form.
private System.Windows.Forms.ProgressBar pbProgress;
//  Windows Image Aquisition Device variable used
//  to store the currently loaded Capture Device.
private WIA.Device CurrentCam;
//  WIA manager COM object used to store the current system WIA
//  information.
private WiaClass            wiaManager;
// WIA ItemClass object used to store the currently selected capture
//  device.
private ItemClass           wiaCamera;
// WIAVideoClass object used to store the current video stream.
private WiaVideoClass       wiaVideo;
//  Video variable used to store the current video object.
private Video CurrentVideo;
//  Integer variable used to store the current state of the
//  video stream.
private int PlayPause;
//  Integer variables used to store the currently recorded
//  object measurements in order to save it.
```

```
        private int SinglePerimeter, SingleArea, SingleXDistance,
                    SingleYDistance, SingleEdgeCount;
        //  Variables Used to maintain and activate the system's agents.
        private Whiteboard whiteboard, whiteboard2, whiteboard3, whiteboard4;
        private MainEdgeDetectionAgent MEDagent, MEDagent,2 MEDagent3,
                                                MEDagent4;
        private Thread MEDthread, MEDthread2, MEDthread3, MEDthread4;
        private InvertColoursAgent ICagent, ICagent2, ICagent3, ICagent4;
        private Thread ICthread, ICthread2, ICthread3, ICthread4;
        private ToBinaryAgent TBagent, TBagent2, TBagent3, TBagent4;
        private Thread TBthread, TBthread2, TBthread3, TBthread4;
        private PerimeterAgent Pagent, Pagent2, Pagent3, Pagent4;
        private Thread Pthread1, Pthread2, Pthread3, Pthread4;
        private EdgeGraphAgent EGagent, EGagent2, EGagent3, EGagent4;
        private Thread EGthread, EGthread2, EGthread3, EGthread4;
```

1.6.3 Form Initialization

```
//  Startup subroutine for frmUpdateModel.
//  Takes 5 arguments:  -  A reference to the current model's directory
//                      -  A reference to the string to be displayed
//                         in the form's header.
//                      -  A reference to the current model's name
//                      -  A reference to which subroutine called this form
//                         (1: New Model, 2: Update Model)
//                      -  A reference to the progress bar on frmMain
public frmUpdateModel(string root, string header, string modelname, int caller,
                      System.Windows.Forms.ProgressBar pb)
{
        InitializeComponent();
        //  Assign passed variables to their local counterparts.
        RootDirectory = root;
        ModelName = modelname;
        this.Text = "Update the " + ModelName + " model";
        pbProgress = pb;
        CurrentlistIndex = 0;
        //  Initialize variables
        filelistIndex = 0;
        CurrentlistIndex = 0;
        Mode = 0;
        BatchModeFileSelect = 0;
        clickcounter = 2;
        bitmap = null;
        Queue1Current = 0;
        Queue2Current = 0;
        Queue3Current = 0;
```

```
Queue4Current = 0;
ProgressIncrementer = 0;
//  Sets up the startup arrangement for frmUpdateModel's
//  toolbar buttons.
tbUpdateModel.Buttons[0].Enabled = true;
tbUpdateModel.Buttons[1].Enabled = true;
tbUpdateModel.Buttons[2].Enabled = true;
tbUpdateModel.Buttons[3].Enabled = true;
//  Hides the Progressbar on frmMain.
pbProgress.Visible = false;
//  Sets the label's value to indicate the current Model's
//  name.
cmdNext.Enabled = false;
cmdPrevious.Enabled = false;
//  Disables picStudy.
picStudy.Enabled = false;
//  Hides the following buttons.
cmdNextQueue1.Visible = false;
cmdPrevQueue1.Visible = false;
cmdNextQueue2.Visible = false;
cmdPrevQueue2.Visible = false;
cmdNextQueue3.Visible = false;
cmdPrevQueue3.Visible = false;
cmdNextQueue4.Visible = false;
cmdPrevQueue4.Visible = false;
cmdActivate.Visible = false;
cmdAccept.Visible = false;
cmdAcceptCamVideo.Visible = false;
//  Hides the following picture box.
picCapturePreview.Visible = false;
//  Make sure no tab pages are currently loaded.
tabPictures.TabPages.Remove(tabNormal);
tabPictures.TabPages.Remove(tabRoughEdges);
tabPictures.TabPages.Remove(tabBatch);
tabPictures.TabPages.Remove(tabCamVideo);
//  Hides the following groupboxes.
gbQueue1.Visible = false;
gbQueue2.Visible = false;
gbQueue3.Visible = false;
gbQueue4.Visible = false;
gbCamVideoMeasurements.Visible = false;
gbControls.Visible = false;
//  Hides the following picturebox.
picCamVideo.Visible = true;
//  Create a new TextReader object, to be used for reading
//  the values stored in the model's .mdl-file.
```

```
TextReader tr;
//  String variable used to temporarily store the value
//  read off the .mdl-file.
string compare = "";
//  Make sure no capture device is currently loaded.
CurrentCam = null;
PerimeterLow = PerimeterHigh = AreaLow = AreaHigh = 0;
ShortestLow = ShortestHigh = LongestLow = LongestHigh = 0;
EdgesLow = EdgesHigh = 0;
//  Checks whether the form was called by the Update Model function.
if (caller == 2)
{
        //  It was called by the Update Model function.
        //  Checks whether the specific model's Directory exists
        if (Directory.Exists(RootDirectory))
        {
                //  The model's directory exists.
                //  Checks whether the specific model's .mdl-file exists.
                if (File.Exists(RootDirectory + "\\" + ModelName + ".mdl"))
                {
                        //  The .mdl-file exists.
                        //  Read the .mdl file and store the values in the
                        //  temporary variables.
                        tr = new StreamReader(RootDirectory + "\\" +
                                ModelName + ".mdl");
                        for (int i = 0; i < 10; i++)
                        {
                                compare = tr.ReadLine();
                                if (i == 0)
                                        if (compare == null)
                                                PerimeterLow = 0;
                                        else
                                                PerimeterLow =
                                                  Int32.Parse(compare);
                                if (i == 1)
                                        if (compare == null)
                                                PerimeterHigh = 0;
                                        else
                                                PerimeterHigh =
                                                  Int32.Parse(compare);
                                if (i == 2)
                                        if (compare == null)
                                                AreaLow = 0;
                                        else
                                                AreaLow =
                                                  Int32.Parse(compare);
```

```
                if (i == 3)
                        if (compare == null)
                                AreaHigh = 0;
                        else
                                AreaHigh =
                                  Int32.Parse(compare);
                if (i == 4)
                        if (compare == null)
                                ShortestLow = 0;
                        else
                                ShortestLow =
                                  Int32.Parse(compare);
                if (i == 5)
                        if (compare == null)
                                ShortestHigh = 0;
                        else
                                ShortestHigh =
                                   Int32.Parse(compare);
                if (i == 6)
                        if (compare == null)
                                LongestLow = 0;
                        else
                                LongestLow =
                                  Int32.Parse(compare);
                if (i == 7)
                        if (compare == null)
                                LongestHigh = 0;
                        else
                                LongestHigh =
                                 Int32.Parse(compare);
                if (i == 8)
                        if (compare == null)
                                EdgesLow = 0;
                        else
                                EdgesLow =
                                  Int32.Parse(compare);
                if (i == 9)
                        if (compare == null)
                                EdgesHigh = 0;
                        else
                                EdgesHigh =
                                   Int32.Parse(compare);
        }
        //  All of the model's variables have been read,
        //  now the Edge graph strings must be read.
        //  This is done until the end of the file is
```

```
                        //  reached
                        while ((compare = tr.ReadLine()) != null)
                        {
                                //  Store the Edge Graph string read off the
                                //  .mdl-file in the PreviousGraphs ArrayList
                                PreviousGraphs.Add(compare);
                        }
                        //  Closes the .mdl-file.
                        tr.Close();
                }
                else
                {
                        //  The .mdl file does not exist, so it is created.
                        File.Create(RootDirectory + "\\" + ModelName +
                                ".mdl");
                }
        }
        else
        {
                //  The directory doesn't exist, so it and the .mdl-file
                //  is created.
                Directory.CreateDirectory(RootDirectory);
                File.Create(RootDirectory + "\\" + ModelName + ".mdl");
        }
    }
}
```

1.6.4 Form Designer Generated Code

```
/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
    if( disposing )
    {
        //      DisposePicure();
        if( wiaVideo != null )
            // release any COM instances
            Marshal.ReleaseComObject( wiaVideo );
        wiaVideo = null;
        if( wiaCamera != null )
            Marshal.ReleaseComObject( wiaCamera );
            wiaCamera = null;
        if( wiaManager != null )
            Marshal.ReleaseComObject( wiaManager );
```

```
                wiaManager = null;
            if (components != null)
            {
                components.Dispose();
            }
        }
        base.Dispose( disposing );
}

#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
//  Component code omitted.
}
#endregion
```

1.6.5 Mouse Down on Single File Mode Image

```
//  This subroutine is called when the left mouse button is pressed.
public void MyMouseDown( Object sender, MouseEventArgs e )
{
        //  Checks whether Single File Mode has been selected.
        if (clickcounter < 1)
        {
            //  Assigns the current position to a point.
            Point A = new Point(e.X,e.Y);
            Corner1 = A;
            // Mark the mouse as being down.
            bHaveMouse = true;
            // Store the starting point for drawing the rectangle.
            ptOriginal.X = e.X;
            ptOriginal.Y = e.Y;
            // Initialize the end points of the rectangle.
            ptLast.X = -1;
            ptLast.Y = -1;
            //  Increment the clickcounter.
            clickcounter++;
        }
}
```

1.6.6 Mouse Up on Single File Mode Image

```
//  This subroutine is called when the left mouse button is released.
public void MyMouseUp( Object sender, MouseEventArgs e )
{
        //  Assigns the current position to a point.
        Point A = new Point(e.X,e.Y);
        Corner2 = A;
        // Mark the mouse as being up.
        bHaveMouse = false;
        //  Draw the final dragging rectangle
        if( ptLast.X != -1 )
        {
                //  Assigns the current position to a point and
                //  calls the subroutine to erase the previous
                //  rectangle
                Point ptCurrent = new Point( e.X, e.Y );
                MyDrawReversibleRectangle( ptOriginal, ptLast );
        }
        // Re-initialize the end and start points of the rectangle.
        ptLast.X = -1;
        ptLast.Y = -1;
        ptOriginal.X = -1;
        ptOriginal.Y = -1;
        //  Make sure which point is top left and which is
        //  bottom right.
        if (Corner2.Y > Corner1.Y)
        {
                TopLeftY = Corner1.Y;
                BottomRightY = Corner2.Y;
        }
        else
        {
                TopLeftY = Corner2.Y;
                BottomRightY = Corner1.Y;
        }
        if (Corner2.X > Corner1.X)
        {
                TopLeftX = Corner1.X;
                BottomRightX = Corner2.X;
        }
        else
        {
                TopLeftX = Corner2.X;
                BottomRightX = Corner1.X;
        }
```

```
//  Create a temporary bitmap value and assign it a reference
//  to the bitmap currently displayed in picturebox
//  picStudy.
Bitmap Main;
Main = (Bitmap) bitmap.Clone();
//  Set the cloned bitmap up as a Graphics object, so that it
//  can be drawn on.
Graphics g = Graphics.FromImage(Main);
//  Create a new pen, with which to draw.
Pen myPen = new Pen(Color.Red);
myPen.Width = 3;
//  Temporary storage value used in calculating
//  relative corner values for the drawing
//  rectangle.
double newvalue = 0;
//  Calculate the relative values, so that the
//  rectangle will be correctly aligned on
//  picturebox pixStudy.
newvalue = TopLeftX * (double)(Main.Width /640.0);
BottomLeftX = TopLeftX = (int) newvalue;
newvalue = 0;
newvalue = BottomRightY * (double)(Main.Height /480.0);
BottomLeftY = BottomRightY = (int) newvalue;
newvalue = 0;
newvalue = BottomRightX * (double)(Main.Width /640.0);
TopRightX = BottomRightX = (int) newvalue;
newvalue = 0;
newvalue = TopLeftY * (double)(Main.Height /480.0);
TopRightY = TopLeftY = (int) newvalue;
newvalue = 0;
CentreX = TopLeftX + ((TopRightX - TopLeftX) / 2);
CentreY = TopLeftY + ((BottomLeftY - TopLeftY) / 2);
//  Draw the rectangle
g.DrawRectangle(myPen,TopLeftX,TopLeftY,TopRightX –
   TopLeftX,BottomLeftY - TopLeftY);
//  Set the picturebox picStudy's image to the newly
//  drawn on bitmap and size it appropriately.
picStudy.Image = Main;
picStudy.SizeMode = PictureBoxSizeMode.StretchImage;
//  Sets up the arrangement for frmUpdateModel's
//  toolbar buttons.
tbUpdateModel.Buttons[0].Enabled = true;
tbUpdateModel.Buttons[1].Enabled = true;
tbUpdateModel.Buttons[2].Enabled = true;
tbUpdateModel.Buttons[3].Enabled = true;
//  Calls the Area Processing subroutine.
```

```
        SingleFileArea();
}
```

1.6.7 Mouse Move on Single File Mode Image

```
// This subroutine is called when the mouse is moved.
public void MyMouseMove( Object sender, MouseEventArgs e )
{
        //  Assigns the current position to a point.
        Point ptCurrent = new Point( e.X, e.Y );
        // Checks whether the mouse is down.
        if( bHaveMouse )
        {
                //  The mouse is down
                //  Checks whether this is the first time this
                //  subroutine has run.
                if( ptLast.X != -1 )
                {
                        //  Not the first time
                        //  Calls the subroutine to erase the previous
                        //  rectangle.
                        MyDrawReversibleRectangle( ptOriginal, ptLast);
                }
                // Update the last point.
                ptLast = ptCurrent;
                // Draw the new rectangle.
                MyDrawReversibleRectangle( ptOriginal, ptLast);
        }
}
```

1.6.8 Draw Rectangle on Single File Mode Image

```
//  This subroutine draws the dragging rectangle.
private void MyDrawReversibleRectangle( Point p1, Point p2 )
{
        //  Create a new rectangle, to be used for drawing a
        //  rectangle on screen.
        Rectangle rc = new Rectangle();
        //  Convert the picStudy coordinates to screen coordinates.
        p1 = picStudy.PointToScreen( p1 );
        p2 = picStudy.PointToScreen( p2 );
        //  Make sure which point is top left and which is
        //  bottom right.  Then calculate the height and
        //  width of the rectangle.
        if( p1.X < p2.X )
        {
```

```
                rc.X = p1.X;
                rc.Width = p2.X - p1.X;
        }
        else
        {
                rc.X = p2.X;
                rc.Width = p1.X - p2.X;
        }
        if( p1.Y < p2.Y )
        {
                rc.Y = p1.Y;
                rc.Height = p2.Y - p1.Y;
        }
        else
        {
                rc.Y = p2.Y;
                rc.Height = p1.Y - p2.Y;
        }
        //  Draw/Erase the temporary rectangle.
        ControlPaint.DrawReversibleFrame(rc,Color.Red, FrameStyle.Dashed);
}
```

1.6.9 Override Mouse Functions

```
// This subroutine defines the overloads for the mouse events.
protected override void OnLoad(System.EventArgs e)
{
        picStudy.MouseDown += new MouseEventHandler( MyMouseDown );
        picStudy.MouseUp += new MouseEventHandler( MyMouseUp );
        picStudy.MouseMove += new MouseEventHandler( MyMouseMove );
        bHaveMouse = false;
}
```

1.6.10 Update toolbar

```
//  This subroutine functions as a switchboard to decide which button was
//  pressed on frmUpdateModel's toolbar (tbUpdateModel).
private void tbUpdateModel_ButtonClick(object sender,
                System.Windows.Forms.ToolBarButtonClickEventArgs e)
{
        switch (tbUpdateModel.Buttons.IndexOf(e.Button))
        {
                //  Called when the 'Single File Mode' button is clicked on the main
                //  toolbar.
                case 0: SelectImage();
                        break;
```

```
                //  Called when the 'Batch File Mode' button is clicked on the
                //  main toolbar.
                case 1: BatchMode();
                        break;
                //  Called when the 'Camera Mode' button is clicked on the main
                //  toolbar.
                case 2: VideoMode();
                        break;
                case 3: CameraMode();
                        break;
        }
}
```

1.6.11 Reset Page

```
//  This subroutine is called when the page needs to be reset and all the
//  variables re-initialized.
private void ResetPage()
{
        //  Initialize variables
        clickcounter = 2;
        Mode = 0;
        BatchModeFileSelect = 0;
        ProgressIncrementer = 0;
        //  Sets up the startup arrangement for frmUpdateModel's
        //  toolbar buttons.
        tbUpdateModel.Buttons[0].Enabled = true;
        tbUpdateModel.Buttons[1].Enabled = true;
        tbUpdateModel.Buttons[2].Enabled = true;
        tbUpdateModel.Buttons[3].Enabled = true;
        //  Enables the following components
        cmdPictureProcess.Enabled = true;
        gbProcessType.Enabled = true;
        tbPictureAdjustment.Enabled = true;
        //  Disables the following components
        picStudy.Enabled = false;
        cmdNext.Enabled = false;
        cmdPrevious.Enabled = false;
        cmdNext.Enabled = false;
        cmdPrevious.Enabled = false;
        //  Remove all the tab pages
        tabPictures.TabPages.Remove(tabNormal);
        tabPictures.TabPages.Remove(tabRoughEdges);
        tabPictures.TabPages.Remove(tabBatch);
        tabPictures.TabPages.Remove(tabCamVideo);
        //  Clears all of the pictureboxes of their images.
```

```
picStudy.Image = null;
picRoughEdges.Image = null;
picCurrentEdges.Image = null;
picOriginalAndEdges.Image = null;
picOriginalAndOutline.Image = null;
picOutline.Image = null;
picQueue1.Image = null;
picQueue2.Image = null;
picQueue3.Image = null;
picQueue4.Image = null;
picCapturePreview.Image = null;
picCapturePreview.BackColor = Color.FromName("Control");
picPicturePreview.Image = null;
picPicturePreview.BackColor = Color.FromName("Control");
picCurrentEdges.Image = null;
picBatchGraphs.Image = null;
//  Hide all of the following buttons.
cmdNextQueue1.Visible = false;
cmdPrevQueue1.Visible = false;
cmdNextQueue2.Visible = false;
cmdPrevQueue2.Visible = false;
cmdNextQueue3.Visible = false;
cmdPrevQueue3.Visible = false;
cmdNextQueue4.Visible = false;
cmdPrevQueue4.Visible = false;
cmdFull.Visible = false;
cmdAccept.Visible = false;
cmdAcceptBatch.Visible = true;
cmdAcceptCamVideo.Visible = false;
//  Clear the text of the following labels.
lblCurrentPerimeter.Text = "";
lblCurrentArea.Text = "";
lblCurrentXDistance.Text = "";
lblCurrentYDistance.Text = "";
lblInfo.Text = "";
lblPictureInfo.Text = "";
//  Clear all items in the following listboxes.
lbBatchQueue1.Items.Clear();
lbBatchQueue2.Items.Clear();
lbBatchQueue3.Items.Clear();
lbBatchQueue4.Items.Clear();
//  Hides the following groupboxes.
gbQueue1.Visible = false;
gbQueue2.Visible = false;
gbQueue3.Visible = false;
gbQueue4.Visible = false;
```

```
        gbCamVideoMeasurements.Visible = false;
        gbPictureMeasurements.Visible = false;
        gbControls.Visible = false;
        gbCamControls.Visible = false;
        gbBatchMeasurements.Visible = false;
        gbPerimeter.Visible = false;
        //  Initializes all the queue item counters.
        Queue1Current = 0;
        Queue2Current = 0;
        Queue3Current = 0;
        Queue4Current = 0;
        //  Reset the video components.
        if (CurrentVideo != null)
        {
                CurrentVideo.Stop();
        }
        CurrentVideo = null;
        tmVideo.Enabled = false;
        tbVidVolume.Value = -1;
        optWhole.Checked = true;
        //  Reset the tabpage components
        tmCamVideoCapture.Enabled = false;
        tbVideoOptions.Buttons[0].Pushed = true;
        tbVideoOptions.Buttons[1].Pushed = false;
        cmdActivate.Text = "Capture";
        cmdActivate.Visible = true;
        picCapturePreview.Visible = true;
        tbVideoOptions.Buttons[0].Enabled = true;
        tbVideoOptions.Buttons[1].Enabled = true;
        BatchGraphs = new ArrayList();
        CurrentlistIndex = 0;
        cmdBatchActivate.Enabled = true;
}
```

1.6.12 Select Image for Still Image Mode

```
//  This subroutine is called when the 'Single File' button is
//  clicked on frmUpdateModel's toolbar.
private void SelectImage()
{
        //  Sets the variable to declare Single File Mode
        Mode = 1;
        //  Sets up the filter and specifications for the
        //  open file dialog ofdPicture.
        ofdPicture.Filter = "All Image Files (*.bmp, *.jpg)|*.bmp; *.jpg|Bitmap Files
                            (*.bmp)|*.bmp|JPEG Files (*.jpg)|*.jpg";
```

```
        ofdPicture.RestoreDirectory = true ;
        //  Open the open file dialog ofdPicture
        if(ofdPicture.ShowDialog() == DialogResult.OK)
        {
                ResetPage();
                //  Adds the tab page that acts as start page for Single File mode
                tabPictures.TabPages.Add(tabNormal);
                tabNormal.Text = "Single Image Mode:  " + ofdPicture.FileName;
                lblPictureInfo.Text = "Image Loaded";
                //  Assigns the selected image to the variable bitmap.
                bitmap = (Bitmap)Bitmap.FromFile(ofdPicture.FileName,false);
                //  Update the form.
                this.Invalidate();
                //  Sets the picturebox picStudy's image to that of bitmap
                //  and sizes it accordingly.
                picStudy.Image = bitmap;
                picStudy.SizeMode = PictureBoxSizeMode.StretchImage;
                //  Hides the following items on tabNormal.
                picPicturePreview.Visible = false;
                gbPictureMeasurements.Visible = false;
                cmdFull.Visible = false;
                //  Sets up the arrangement for frmUpdateModel's
                //  toolbar buttons.
                tbUpdateModel.Buttons[0].Enabled = true;
                tbUpdateModel.Buttons[1].Enabled = true;
                tbUpdateModel.Buttons[2].Enabled = true;
                tbUpdateModel.Buttons[3].Enabled = true;
                // Creates the agent communication directory.
                if (Directory.Exists(Application.StartupPath + "\\Recognition")
                    == true)
                  Directory.Delete(Application.StartupPath + "\\Recognition",true);
                Directory.CreateDirectory(Application.StartupPath +
                  "\\Recognition");
        }
}
```

1.6.13 Select Image for Batch Image Mode

```
//  This subroutine is called when the 'Batch File' button is
//  clicked on frmUpdateModel's toolbar.
private void BatchMode()
{
        //  Sets the variable to declare Batch File Mode
        Mode = 2;
        //  This variable keeps track of the number of files
        //  to be processed during Batch File processing.
```

```
int count = 0;
//  Sets up the filter and specifications for the
//  open file dialog ofdPicture.
ofdPicture.Filter = "All Image Files (*.bmp, *.jpg)|*.bmp; *.jpg|Bitmap Files
                    (*.bmp)|*.bmp|JPEG Files (*.jpg)|*.jpg";
ofdPicture.RestoreDirectory = true ;
ofdPicture.Multiselect = true;
//  Open the open file dialog ofdPicture
if(ofdPicture.ShowDialog() == DialogResult.OK)
{
        if (ofdPicture.FileNames.Length > 1)
        {
                ResetPage();
                lblBatchInfo.Text = "Images loaded";
                // Creates the agent communication directory.
                if (Directory.Exists(Application.StartupPath + "\\Recognition")
                  == true)
                   Directory.Delete(Application.StartupPath +
                        "\\Recognition",true);
                Directory.CreateDirectory(Application.StartupPath +
                  "\\Recognition");
                //  Adds the tab page that acts as start page for Batch File
                //  mode
                tabPictures.TabPages.Add(tabBatch);
                tabBatch.Text = "Batch File Mode";
                //  Hides the following item on tabBatch
                gbBatchMeasurements.Visible = false;
                //  Assigns the selected array of selected filename paths
                //  to the string array filenames.
                filenames = new string[ofdPicture.FileNames.Length];
                filenames = ofdPicture.FileNames;
                //  Divide the list of filenames between the 4 Batch queues
                for (int i = 0; i < filenames.Length; i++)
                {
                        count++;
                        if (count - 1 == 0)
                                lbBatchQueue1.Items.Add(filenames[i]);
                        if (count - 2 == 0)
                                lbBatchQueue2.Items.Add(filenames[i]);
                        if (count - 3 == 0)
                                lbBatchQueue3.Items.Add(filenames[i]);
                        if (count - 4 == 0)
                        {
                                lbBatchQueue4.Items.Add(filenames[i]);
                                count = 0;
                        }
```

```
}
//  Sets the amount by which each finished, processed image
//  increments the progress bar on frmMain.
ProgressIncrementer = 100 / (lbBatchQueue1.Items.Count +
        lbBatchQueue2.Items.Count +
lbBatchQueue3.Items.Count + lbBatchQueue4.Items.Count);
//  Checks whether there are images in the first queue.
if (lbBatchQueue1.Items.Count > 0)
{
        // There are images.
        // Checks whether there is more than one image in
        //  the queue.
        if (lbBatchQueue1.Items.Count > 1)
        {
                // There is, so show the navigation buttons.
                cmdNextQueue1.Visible = true;
                cmdPrevQueue1.Visible = true;
        }
        //  Displays the first image in the queue and
        //  sizes it appropriately.
        picQueue1.Image = (Bitmap)Bitmap.FromFile
              (lbBatchQueue1.Items[0].ToString());
        picQueue1.SizeMode =
              PictureBoxSizeMode.StretchImage;
        picQueue1.Refresh();
        //  Makes the queue's display area visible.
        gbQueue1.Visible = true;
}
//  Checks whether there are images in the second queue.
if (lbBatchQueue2.Items.Count > 0)
{
        // There are images.
        // Checks whether there is more than one image in
        //  the queue.
        if (lbBatchQueue2.Items.Count > 1)
        {
                // There is, so show the navigation buttons.
                cmdNextQueue2.Visible = true;
                cmdPrevQueue2.Visible = true;
        }
        //  Displays the first image in the queue and
        //  sizes it appropriately.
        picQueue2.Image = (Bitmap)Bitmap.FromFile
             (lbBatchQueue2.Items[0].ToString());
        picQueue2.SizeMode =
              PictureBoxSizeMode.StretchImage;
```

```
            picQueue2.Refresh();
            //  Makes the queue's display area visible.
            gbQueue2.Visible = true;
        }
        //  Checks whether there are images in the third queue.
        if (lbBatchQueue3.Items.Count > 0)
        {
            // There are images.
            // Checks whether there is more than one image in
            //  the queue.
            if (lbBatchQueue3.Items.Count > 1)
            {
                // There is, so show the navigation buttons.
                cmdNextQueue3.Visible = true;
                cmdPrevQueue3.Visible = true;
            }
            //  Displays the first image in the queue and
            //  sizes it appropriately.
            picQueue3.Image = (Bitmap)Bitmap.FromFile
                (lbBatchQueue3.Items[0].ToString());
            picQueue3.SizeMode =
                  PictureBoxSizeMode.StretchImage;
            picQueue3.Refresh();
            //  Makes the queue's display area visible.
            gbQueue3.Visible = true;
        }
        //  Checks whether there are images in the fourth queue.
        if (lbBatchQueue4.Items.Count > 0)
        {
            // There are images.
            // Checks whether there is more than one image in
            //  the queue.
            if (lbBatchQueue4.Items.Count > 1)
            {
                // There is, so show the navigation buttons.
                cmdNextQueue4.Visible = true;
                cmdPrevQueue4.Visible = true;
            }
            //  Displays the first image in the queue and
            //  sizes it appropriately.
            picQueue4.Image = (Bitmap)Bitmap.FromFile
                  (lbBatchQueue4.Items[0].ToString());
            picQueue4.SizeMode =
                  PictureBoxSizeMode.StretchImage;
            picQueue4.Refresh();
            //  Makes the queue's display area visible.
```

```
                    gbQueue4.Visible = true;
                }
                //  Update the form.
                this.Invalidate();
                //  Sets up the arrangement for frmUpdateModel's
                //  toolbar buttons.
                tbUpdateModel.Buttons[0].Enabled = true;
                tbUpdateModel.Buttons[1].Enabled = true;
                tbUpdateModel.Buttons[2].Enabled = true;
                tbUpdateModel.Buttons[3].Enabled = true;
                //  Initializes the progress bar on frmMain.
                pbProgress.Value = 0;
                //  Increments the variable, this ensures that the
                //  Batch Mode tab page only gets loaded once.
                BatchModeFileSelect++;
            }
            else
                MessageBox.Show("Batch Mode requires more than one
                    image to be selected at a time.","Message",
                    MessageBoxButtons.OK,MessageBoxIcon.Information);
        }
}
```

1.6.14 Single Image Area Mode

```
//  This subroutine is called when "learning" needs to happen in Single File Mode
//  for a selected area.
private void SingleFileArea()
{
        //  Disable picStudy, so no further object selection is possible.
        picStudy.Enabled = false;
        //  ArrayList variable used to store the EdgeGraph ArrayList returned
        //  by the EdgeGraphAgent.
        ArrayList NewGraphs;
        //  Creates the shared Whiteboard object. Then initializes all of the agents
        //  and loads them onto individual threads.
        whiteboard = new Whiteboard((Bitmap) bitmap.Clone());
        MEDagent = new  MainEdgeDetectionAgent(640,480,whiteboard);
        MEDthread = new Thread(new ThreadStart(MEDagent.StartWait));
        MEDthread.Start();
        ICagent = new InvertColoursAgent(640,480,whiteboard);
        ICthread = new Thread(new ThreadStart(ICagent.StartWait));
        ICthread.Start();
        TBagent = new ToBinaryAgent(640,480,whiteboard);
        TBthread = new Thread(new ThreadStart(TBagent.StartWait));
        TBthread.Start();
```

```
    Pagent = new PerimeterAgent(whiteboard,
                TopLeftX,TopLeftY,BottomRightX,BottomRightY,1);
    Pthread = new Thread(new ThreadStart(Pagent.StartWait));
    Pthread.Start();
    EGagent = new EdgeGraphAgent(whiteboard,
                  TopLeftX,TopLeftY,BottomRightX,BottomRightY);
    EGthread = new Thread(new ThreadStart(EGagent.StartWait));
    EGthread.Start();
    //  Show the user that the program is processing.
    System.Windows.Forms.Cursor.Current =
              System.Windows.Forms.Cursors.WaitCursor;
//  Increment the counter to let the agents know that they may start
//  processing.
whiteboard.Increment();
//  Halts all of the threads when they have finished processing.
while (whiteboard.RunStartWait == 0)
{
        if (whiteboard.Counter > 6)
        {
            if (Pagent.getPerimeter() != 0)
            {
            //  An object was found.
            cmdAccept.Visible = true;
            picPicturePreview.Image = whiteboard.picInUseDone;
            picPicturePreview.SizeMode = PictureBoxSizeMode.StretchImage;
            picPicturePreview.BackColor = Color.White;
            lblPictureInfo.Text = "Object Found";
            gbPictureMeasurements.Visible = true;
            //  Calls the OverlayAgent, to superimpose the rough edges on the
            //  image that recognition was performed on.
            Overlay OO = new
                Overlay(bitmap,TBagent.getpicInUse(), Pagent.getpicInUse());
            picStudy.Image = OO.getbmpOutline();
            //  Set the image to be displayed in the picturebox picRougheEdges
            //  and sizes it appropriately.  This displays only the rough edges
            //  in black on a white background.
            picRoughOutlineBitmap = TBagent.getpicInUse();
            picRoughEdges.Image = TBagent.getpicInUse();
            picRoughEdges.SizeMode = PictureBoxSizeMode.StretchImage;
            //  Set the image to be displayed in the picturebox
            //  picOriginalAndEdges
            //  and sizes it appropriately.  This displays a greyscale version of
            //  the original image with the rough edges superimposed in red.
            bmpOriginalAndEdges = OO.getbmpEdges();
            picOriginalAndEdges.Image = bmpOriginalAndEdges;
            picOriginalAndEdges.SizeMode =
```

```
    PictureBoxSizeMode.StretchImage;
//  Set the image to be displayed in the picturebox
//  picOriginalAndOutline
//  and sizes it appropriately.  This displays a greyscale version of
//  the original image with the object's outline superimposed in red.
bmpOriginalAndOutline = OO.getbmpOutline();
picOriginalAndOutline.Image = bmpOriginalAndOutline;
picOriginalAndOutline.SizeMode =
     PictureBoxSizeMode.StretchImage;
//  Set the image to be displayed in the picturebox picOutline
//  and sizes it appropriately.  This displays only the object's
//  outline in black on a white background.
//  Calls the RemoveOutsideBackgroundAgent, to set any pixel
//  around the object's selected area to white.
bmpOutline = Pagent.getpicInUse();
picOutline.Image = bmpOutline;
picOutline.SizeMode = PictureBoxSizeMode.StretchImage;
//  Sets the picturebox picCurrentEdges image and the temporay
//  bitmap  to that of the Edge Graph generated for the object on the
//  current image.
picCurrentEdges.Image = EGagent.getGraph();
bmpCurrentEdges = EGagent.getGraph();
picCurrentEdges.SizeMode = PictureBoxSizeMode.StretchImage;
//  Assigns all of the Graph Edge strings, generated by the
//  EdgeGraphAgent to the ArrayList NewGraphs.
NewGraphs = EGagent.getGraphs();
//  Adds all of the items in the NewGraphs ArrayList to the
//  CurrentGraphs ArrayList.
foreach(string i in NewGraphs)
          CurrentGraphs.Add(i);
//  Assigns the newly generated statistics, for the found object,
//  to the following labels.
lblPicturePerimeter.Text = lblCurrentPerimeter.Text =
    Pagent.getPerimeter().ToString();
lblPictureArea.Text = lblCurrentArea.Text =
   Pagent.getArea().ToString();
lblPictureXDistance.Text = lblCurrentXDistance.Text =
   Pagent.getXDistance().ToString();
lblPictureYDistance.Text = lblCurrentYDistance.Text =
   Pagent.getYDistance().ToString();
lblPictureEdges.Text = lblCurrentEdges.Text =
    EGagent.getEdgeCount().ToString();
//  Assigns the current measurements to global variables.
SinglePerimeter = Pagent.getPerimeter();
SingleArea = Pagent.getArea();
SingleXDistance = Pagent.getXDistance();
```

```
                SingleYDistance = Pagent.getYDistance();
                SingleEdgeCount = EGagent.getEdgeCount();
                //  Gets a list of all the previously created Graph images.
                filelist = Directory.GetFiles(RootDirectory,"*.jpg");
                tbUpdateModel.Buttons[0].Enabled = true;
                tbUpdateModel.Buttons[1].Enabled = true;
                tbUpdateModel.Buttons[2].Enabled = true;
                tbUpdateModel.Buttons[3].Enabled = true;
                //  Makes the following items on tabNormal visible.
                picPicturePreview.Visible = true;
                gbPictureMeasurements.Visible = true;
                cmdFull.Visible = true;
                }
                else
                {
                        //  Displays an appropriate message when no object is
                        //  found.
                        lblPictureInfo.Text = "Object couldn't be found";
                        //  Sets up the arrangement for frmUpdateModel's
                        //  toolbar buttons.
                        tbUpdateModel.Buttons[0].Enabled = true;
                        tbUpdateModel.Buttons[1].Enabled = true;
                        tbUpdateModel.Buttons[2].Enabled = true;
                        tbUpdateModel.Buttons[3].Enabled = true;
                }
            }
        }
}
```

1.6.15 Single Image Whole Image Mode

```
//  This subroutine is called when "learning" needs to happen in Single File Mode
//  for the whole image.
private void SingleFileWhole()
{
        //  Disable picStudy, so no further object selection is possible.
        picStudy.Enabled = false;
        //  ArrayList variable used to store the EdgeGraph ArrayList returned
        //  by the EdgeGraphAgent.
        ArrayList NewGraphs;
        //  Creates the shared Whiteboard object. Then initializes all of the agents
        //  and loads them onto individual threads.
        whiteboard = new Whiteboard((Bitmap) bitmap.Clone());
        MEDagent = new  MainEdgeDetectionAgent(640,480,whiteboard);
        MEDthread = new Thread(new ThreadStart(MEDagent.StartWait));
        MEDthread.Start();
```

```
    ICagent = new InvertColoursAgent(640,480,whiteboard);
    ICthread = new Thread(new ThreadStart(ICagent.StartWait));
    ICthread.Start();
    TBagent = new ToBinaryAgent(640,480,whiteboard);
    TBthread = new Thread(new ThreadStart(TBagent.StartWait));
    TBthread.Start();
    Pagent = new PerimeterAgent(whiteboard,5,5,635,475,1);
    Pthread = new Thread(new ThreadStart(Pagent.StartWait));
    Pthread.Start();
    EGagent = new EdgeGraphAgent(whiteboard,5,435,5,635);
    EGthread = new Thread(new ThreadStart(EGagent.StartWait));
    EGthread.Start();
    //  Show the user that the program is processing.
    System.Windows.Forms.Cursor.Current =
              System.Windows.Forms.Cursors.WaitCursor;
//  Increment the counter to let the agents know that they may start
//  processing.
whiteboard.Increment();
while (whiteboard.RunStartWait == 0)
{
        if (whiteboard.Counter > 6)
        {
            if (Pagent.getPerimeter() != 0)
            {
            //  An object was found.
            cmdAccept.Visible = true;
            picPicturePreview.Image = whiteboard.picInUseDone;
            picPicturePreview.SizeMode = PictureBoxSizeMode.StretchImage;
            picPicturePreview.BackColor = Color.White;
            lblPictureInfo.Text = "Object Found";
            gbPictureMeasurements.Visible = true;
            //  Calls the OverlayAgent, to superimpose the rough edges on the
            //  image that recognition was performed on.
            Overlay OO = new
                Overlay(bitmap,TBagent.getpicInUse(), Pagent.getpicInUse());
            picStudy.Image = OO.getbmpOutline();
            //  Set the image to be displayed in the picturebox picRougheEdges
            //  and sizes it appropriately.  This displays only the rough edges
            //  in black on a white background.
            picRoughOutlineBitmap = TBagent.getpicInUse();
            picRoughEdges.Image = TBagent.getpicInUse();
            picRoughEdges.SizeMode = PictureBoxSizeMode.StretchImage;
            //  Set the image to be displayed in the picturebox
            //  picOriginalAndEdges
            //  and sizes it appropriately.  This displays a greyscale version of
            //  the original image with the rough edges superimposed in red.
```

```
bmpOriginalAndEdges = OO.getbmpEdges();
picOriginalAndEdges.Image = bmpOriginalAndEdges;
picOriginalAndEdges.SizeMode =
    PictureBoxSizeMode.StretchImage;
//  Set the image to be displayed in the picturebox
//  picOriginalAndOutline
//  and sizes it appropriately.  This displays a greyscale version of
//  the original image with the object's outline superimposed in red.
bmpOriginalAndOutline = OO.getbmpOutline();
picOriginalAndOutline.Image = bmpOriginalAndOutline;
picOriginalAndOutline.SizeMode =
     PictureBoxSizeMode.StretchImage;
//  Set the image to be displayed in the picturebox picOutline
//  and sizes it appropriately.  This displays only the object's
//  outline in black on a white background.
//  Calls the RemoveOutsideBackgroundAgent, to set any pixel
//  around the object's selected area to white.
bmpOutline = Pagent.getpicInUse();
picOutline.Image = bmpOutline;
picOutline.SizeMode = PictureBoxSizeMode.StretchImage;
//  Sets the picturebox picCurrentEdges image and the temporay
//  bitmap  to that of the Edge Graph generated for the object on the
//  current image.
picCurrentEdges.Image = EGagent.getGraph();
bmpCurrentEdges = EGagent.getGraph();
picCurrentEdges.SizeMode = PictureBoxSizeMode.StretchImage;
//  Assigns all of the Graph Edge strings, generated by the
//  EdgeGraphAgent to the ArrayList NewGraphs.
NewGraphs = EGagent.getGraphs();
//  Adds all of the items in the NewGraphs ArrayList to the
//  CurrentGraphs ArrayList.
foreach(string i in NewGraphs)
        CurrentGraphs.Add(i);
//  Assigns the newly generated statistics, for the found object,
//  to the following labels.
lblPicturePerimeter.Text = lblCurrentPerimeter.Text =
    Pagent.getPerimeter().ToString();
lblPictureArea.Text = lblCurrentArea.Text =
   Pagent.getArea().ToString();
lblPictureXDistance.Text = lblCurrentXDistance.Text =
   Pagent.getXDistance().ToString();
lblPictureYDistance.Text = lblCurrentYDistance.Text =
   Pagent.getYDistance().ToString();
lblPictureEdges.Text = lblCurrentEdges.Text =
    EGagent.getEdgeCount().ToString();
//  Assigns the current measurements to global variables.
```

```
                SinglePerimeter = Pagent.getPerimeter();
                SingleArea = Pagent.getArea();
                SingleXDistance = Pagent.getXDistance();
                SingleYDistance = Pagent.getYDistance();
                SingleEdgeCount = EGagent.getEdgeCount();
                //  Gets a list of all the previously created Graph images.
                filelist = Directory.GetFiles(RootDirectory,"*.jpg");
                tbUpdateModel.Buttons[0].Enabled = true;
                tbUpdateModel.Buttons[1].Enabled = true;
                tbUpdateModel.Buttons[2].Enabled = true;
                tbUpdateModel.Buttons[3].Enabled = true;
                //  Makes the following items on tabNormal visible.
                picPicturePreview.Visible = true;
                gbPictureMeasurements.Visible = true;
                cmdFull.Visible = true;
                }
                else
                {
                        //  Displays an appropriate message when no object is
                        //  found.
                        lblPictureInfo.Text = "Object couldn't be found";
                        //  Sets up the arrangement for frmUpdateModel's
                        //  toolbar buttons.
                        tbUpdateModel.Buttons[0].Enabled = true;
                        tbUpdateModel.Buttons[1].Enabled = true;
                        tbUpdateModel.Buttons[2].Enabled = true;
                        tbUpdateModel.Buttons[3].Enabled = true;
                }
            }
        }
}
```

1.6.16 Batch Image Mode

```
//  This subroutine is called when "learning" needs to happen in Batch File Mode.
private void BatchFile()
{
        //  Integer variables used to store the total number of
        //  images processed and the total number of objects found.
        int total, found;
        //  Checks whether the "Batch" directory exists.
        if(Directory.Exists(RootDirectory + "\\Batch"))
        {
                //  It exists, so delete it and then create it
                //  again.  This ensures that there are no items
                //  in the batch directory before processing begins.
```

```
            Directory.Delete(RootDirectory + "\\Batch",true);
            Directory.CreateDirectory(RootDirectory + "\\Batch");
        }
        else
            //  The directory doesn't exist, so create it.
            Directory.CreateDirectory(RootDirectory + "\\Batch");
           //  Makes the progress bar on frmMain visible and
            //  initializes it.
            pbProgress.Visible = true;
            pbProgress.Value = 0;
            //  Create 4 new processing threads.  One for each queue.
            Thread Queue1 = new Thread(new
                 ThreadStart(BatchFileQueue1));
            Thread Queue2 = new Thread(new
                 ThreadStart(BatchFileQueue2));
            Thread Queue3 = new Thread(new
                ThreadStart(BatchFileQueue3));
            Thread Queue4 = new Thread(new
                 ThreadStart(BatchFileQueue4));
            //  Assign each queue thread a name.
            Queue1.Name = "Queue1";
            Queue2.Name = "Queue2";
            Queue3.Name = "Queue3";
            Queue4.Name = "Queue4";
            //  Start all of the queue threads.
            Queue1.Start();
            Queue2.Start();
            Queue3.Start();
            Queue4.Start();
            //  Ensures that all of the queue threads complete processing
            //  before the program continues.
            Queue1.Join();
            Queue2.Join();
            Queue3.Join();
            Queue4.Join();
            //  Sets the Batch File process' progress bar to 100%,
            //  indicating that processing is complete.
            //  The progress bar is then hidden.
            pbProgress.Value = 100;
            pbProgress.Refresh();
            pbProgress.Visible = false;
            //  Gets a list of all the Graph images created during Batch
            //  File processing.
            //  BatchGraphs = Directory.GetFiles(RootDirectory + "\\Batch");
            //  Assigns the picturebox picCurrentEdges the first image
            //  in the BatchGraphs ArrayList.
```

```
picCurrentEdges.Image = (Bitmap)BatchGraphs[CurrentlistIndex];
picCurrentEdges.SizeMode = PictureBoxSizeMode.StretchImage;
//  If the BatchGraphs ArrayList consists of more than one Graph,
//  enable the 'CurrentNext' button to navigate through the images.
if (BatchGraphs.Count > 1)
{
        cmdNext.Enabled = true;
}

//  If the following variables don't have assigned values, they
//  are assigned default values.
if(PerimeterLow == 0)
        PerimeterLow = Int32.MaxValue;
if(PerimeterHigh == 0)
        PerimeterHigh = 0;
if(AreaLow == 0)
        AreaLow = Int32.MaxValue;
if(AreaHigh == 0)
        AreaHigh = 0;
if(ShortestLow == 0)
        ShortestLow = Int32.MaxValue;
if(ShortestHigh == 0)
        ShortestHigh = 0;
if(LongestLow == 0)
        LongestLow = Int32.MaxValue;
if(LongestHigh == 0)
        LongestHigh = 0;
if(EdgesLow == 0)
        EdgesLow = Int32.MaxValue;
if(EdgesHigh == 0)
        EdgesHigh = 0;
//  Runs through the values generated for the first queue, to see
//  if they will replace the current values.
if ((PerimeterLowQueue1 < PerimeterLow) &&
   (PerimeterLowQueue1 > 0))
PerimeterLow = PerimeterLowQueue1;
 if (PerimeterHighQueue1 > PerimeterHigh)
    PerimeterHigh = PerimeterHighQueue1;
if ((AreaLowQueue1 < AreaLow) && (AreaLowQueue1 > 0))
   AreaLow = AreaLowQueue1;
if (AreaHighQueue1 > AreaHigh)
   AreaHigh = AreaHighQueue1;
if ((ShortestLowQueue1 < ShortestLow) && (ShortestLowQueue1 >
   0))
    ShortestLow = ShortestLowQueue1;
if (ShortestHighQueue1 > ShortestHigh)
```

```
    ShortestHigh = ShortestHighQueue1;
if ((LongestLowQueue1 < LongestLow) && (LongestLowQueue1 >
    0))
    LongestLow = LongestLowQueue1;
if (LongestHighQueue1 > LongestHigh)
  LongestHigh = LongestHighQueue1;
if ((EdgesLowQueue1 < EdgesLow) && (EdgesLowQueue1 > 0))
  EdgesLow = EdgesLowQueue1;
if (EdgesHighQueue1 > EdgesHigh)
  EdgesHigh = EdgesHighQueue1;
lblBatchPerimeterLow.Text = PerimeterLowQueue1.ToString();
lblBatchPerimeterHigh.Text = PerimeterHighQueue1.ToString();
lblBatchAreaLow.Text = AreaLowQueue1.ToString();
lblBatchAreaHigh.Text = AreaHighQueue1.ToString();
lblBatchXDistanceLow.Text = ShortestLowQueue1.ToString();
lblBatchXDistanceHigh.Text = ShortestHighQueue1.ToString();
lblBatchYDistanceLow.Text = LongestLowQueue1.ToString();
lblBatchYDistanceHigh.Text = LongestHighQueue1.ToString();
lblBatchEdgesLow.Text = EdgesLowQueue1.ToString();
lblBatchEdgesHigh.Text = EdgesHighQueue1.ToString();
//  Runs through the values generated for the second queue, to see
//  if they will replace the current values.
if ((PerimeterLowQueue2 < PerimeterLow) &&
   (PerimeterLowQueue2 > 0))
    PerimeterLow = PerimeterLowQueue2;
if (PerimeterHighQueue2 > PerimeterHigh)
  PerimeterHigh = PerimeterHighQueue2;
if ((AreaLowQueue2 < AreaLow) && (AreaLowQueue2 > 0))
  AreaLow = AreaLowQueue2;
if (AreaHighQueue2 > AreaHigh)
  AreaHigh = AreaHighQueue2;
if ((ShortestLowQueue2 < ShortestLow) && (ShortestLowQueue2 >
    0))
    ShortestLow = ShortestLowQueue2;
if (ShortestHighQueue2 > ShortestHigh)
  ShortestHigh = ShortestHighQueue2;
 if ((LongestLowQueue2 < LongestLow) && (LongestLowQueue2 >
    0))
    LongestLow = LongestLowQueue2;
if (LongestHighQueue2 > LongestHigh)
    LongestHigh = LongestHighQueue2;
if ((EdgesLowQueue2 < EdgesLow) && (EdgesLowQueue2 > 0))
  EdgesLow = EdgesLowQueue2;
if (EdgesHighQueue2 > EdgesHigh)
  EdgesHigh = EdgesHighQueue2;
if (PerimeterLowQueue2 <
```

```
            Int32.Parse(lblBatchPerimeterLow.Text))
            lblBatchPerimeterLow.Text =
            PerimeterLowQueue2.ToString();
if (PerimeterHighQueue2 >
            Int32.Parse(lblBatchPerimeterHigh.Text))
            lblBatchPerimeterHigh.Text =
             PerimeterHighQueue2.ToString();
if (AreaLowQueue2 < Int32.Parse(lblBatchAreaLow.Text))
            lblBatchAreaLow.Text = AreaLowQueue2.ToString();
if (AreaHighQueue2 > Int32.Parse(lblBatchAreaHigh.Text))
            lblBatchAreaHigh.Text = AreaHighQueue2.ToString();
if (ShortestLowQueue2 < Int32.Parse(lblBatchXDistanceLow.Text))
           lblBatchXDistanceLow.Text =
            ShortestLowQueue2.ToString();
if (ShortestHighQueue2 >
           Int32.Parse(lblBatchXDistanceHigh.Text))
           lblBatchXDistanceHigh.Text =
           ShortestHighQueue2.ToString();
if (LongestLowQueue2 < Int32.Parse(lblBatchYDistanceLow.Text))
            lblBatchYDistanceLow.Text =
            LongestHighQueue2.ToString();
if (LongestHighQueue2 > Int32.Parse(lblBatchYDistanceHigh.Text))
            lblBatchYDistanceHigh.Text =
            ShortestHighQueue2.ToString();
if (EdgesLowQueue2 < Int32.Parse(lblBatchEdgesLow.Text))
            lblBatchEdgesLow.Text = EdgesLowQueue2.ToString();
if (EdgesHighQueue2 > Int32.Parse(lblBatchEdgesHigh.Text))
            lblBatchEdgesHigh.Text = EdgesHighQueue2.ToString();
//  Runs through the values generated for the third queue, to see
//  if they will replace the current values.
if ((PerimeterLowQueue3 < PerimeterLow) &&
           (PerimeterLowQueue3 > 0))
            PerimeterLow = PerimeterLowQueue3;
if (PerimeterHighQueue3 > PerimeterHigh)
            PerimeterHigh = PerimeterHighQueue3;
if ((AreaLowQueue3 < AreaLow) && (AreaLowQueue3 > 0))
            AreaLow = AreaLowQueue3;
if (AreaHighQueue3 > AreaHigh)
            AreaHigh = AreaHighQueue3;
if ((ShortestLowQueue3 < ShortestLow) && (ShortestLowQueue3 >
           0))
            ShortestLow = ShortestLowQueue3;
if (ShortestHighQueue3 > ShortestHigh)
            ShortestHigh = ShortestHighQueue3;
if ((LongestLowQueue3 < LongestLow) && (LongestLowQueue3 >
           0))
```

```
            LongestLow = LongestLowQueue3;
    if (LongestHighQueue3 > LongestHigh)
            LongestHigh = LongestHighQueue3;
    if ((EdgesLowQueue3 < EdgesLow) && (EdgesLowQueue3 > 0))
            EdgesLow = EdgesLowQueue3;
    if (EdgesHighQueue3 > EdgesHigh)
            EdgesHigh = EdgesHighQueue3;
    if (PerimeterLowQueue3 <
            Int32.Parse(lblBatchPerimeterLow.Text))
            lblBatchPerimeterLow.Text =
            PerimeterLowQueue3.ToString();
    if (PerimeterHighQueue3 >
            Int32.Parse(lblBatchPerimeterHigh.Text))
            lblBatchPerimeterHigh.Text =
            PerimeterHighQueue3.ToString();
    if (AreaLowQueue3 < Int32.Parse(lblBatchAreaLow.Text))
            lblBatchAreaLow.Text = AreaLowQueue3.ToString();
    if (AreaHighQueue3 > Int32.Parse(lblBatchAreaHigh.Text))
            lblBatchAreaHigh.Text = AreaHighQueue3.ToString();
    if (ShortestLowQueue3 < Int32.Parse(lblBatchXDistanceLow.Text))
            lblBatchXDistanceLow.Text =
            ShortestLowQueue3.ToString();
    if (ShortestHighQueue3 >
            Int32.Parse(lblBatchXDistanceHigh.Text))
            lblBatchXDistanceHigh.Text =
            ShortestHighQueue3.ToString();
    if (LongestLowQueue3 < Int32.Parse(lblBatchYDistanceLow.Text))
            lblBatchYDistanceLow.Text =
            LongestHighQueue3.ToString();
    if (LongestHighQueue3 > Int32.Parse(lblBatchYDistanceHigh.Text))
            lblBatchYDistanceHigh.Text =
            ShortestHighQueue3.ToString();
    if (EdgesLowQueue3 < Int32.Parse(lblBatchEdgesLow.Text))
            lblBatchEdgesLow.Text = EdgesLowQueue3.ToString();
    if (EdgesHighQueue3 > Int32.Parse(lblBatchEdgesHigh.Text))
            lblBatchEdgesHigh.Text = EdgesHighQueue3.ToString();
    //  Runs through the values generated for the fourth queue, to see
    //  if they will replace the current values.
    if ((PerimeterLowQueue4 < PerimeterLow) &&
            (PerimeterLowQueue4 > 0))
            PerimeterLow = PerimeterLowQueue4;
    if (PerimeterHighQueue4 > PerimeterHigh)
            PerimeterHigh = PerimeterHighQueue4;
    if ((AreaLowQueue4 < AreaLow) && (AreaLowQueue4 > 0))
            AreaLow = AreaLowQueue4;
    if (AreaHighQueue4 > AreaHigh)
```

```
            AreaHigh = AreaHighQueue4;
if ((ShortestLowQueue4 < ShortestLow) && (ShortestLowQueue4 >
             0))
            ShortestLow = ShortestLowQueue4;
if (ShortestHighQueue4 > ShortestHigh)
            ShortestHigh = ShortestHighQueue4;
if ((LongestLowQueue4 < LongestLow) && (LongestLowQueue4 >
             0))
            LongestLow = LongestLowQueue4;
if (LongestHighQueue4 > LongestHigh)
            LongestHigh = LongestHighQueue4;
if ((EdgesLowQueue4 < EdgesLow) && (EdgesLowQueue4 > 0))
            EdgesLow = EdgesLowQueue4;
if (EdgesHighQueue4 > EdgesHigh)
            EdgesHigh = EdgesHighQueue4;
if (PerimeterLowQueue4 <
            Int32.Parse(lblBatchPerimeterLow.Text))
            lblBatchPerimeterLow.Text =
            PerimeterLowQueue4.ToString();
if (PerimeterHighQueue4 >
            Int32.Parse(lblBatchPerimeterHigh.Text))
            lblBatchPerimeterHigh.Text =
            PerimeterHighQueue4.ToString();
if (AreaLowQueue4 < Int32.Parse(lblBatchAreaLow.Text))
            lblBatchAreaLow.Text = AreaLowQueue4.ToString();
if (AreaHighQueue4 > Int32.Parse(lblBatchAreaHigh.Text))
            lblBatchAreaHigh.Text = AreaHighQueue4.ToString();
if (ShortestLowQueue4 < Int32.Parse(lblBatchXDistanceLow.Text))
            lblBatchXDistanceLow.Text =
            ShortestLowQueue4.ToString();
if (ShortestHighQueue4 >
             Int32.Parse(lblBatchXDistanceHigh.Text))
            lblBatchXDistanceHigh.Text =
            ShortestHighQueue4.ToString();
if (LongestLowQueue4 < Int32.Parse(lblBatchYDistanceLow.Text))
            lblBatchYDistanceLow.Text =
            LongestHighQueue4.ToString();
if (LongestHighQueue4 > Int32.Parse(lblBatchYDistanceHigh.Text))
            lblBatchYDistanceHigh.Text =
            ShortestHighQueue4.ToString();
if (EdgesLowQueue4 < Int32.Parse(lblBatchEdgesLow.Text))
            lblBatchEdgesLow.Text = EdgesLowQueue4.ToString();
if (EdgesHighQueue4 > Int32.Parse(lblBatchEdgesHigh.Text))
            lblBatchEdgesHigh.Text = EdgesHighQueue4.ToString();
//  Sets up the arrangement for frmUpdateModel's
//  toolbar buttons.
```

```
                tbUpdateModel.Buttons[0].Enabled = true;
                tbUpdateModel.Buttons[1].Enabled = true;
                tbUpdateModel.Buttons[2].Enabled = true;
                tbUpdateModel.Buttons[3].Enabled = true;
                //  Gets the total number of images processed.
                total = TotalQueue1 + TotalQueue2 + TotalQueue3 + TotalQueue4;
                //  Gets the total number of images on which objects were found.
                found = FoundQueue1 + FoundQueue2 + FoundQueue3 +
                        FoundQueue4;
                //  Makes the following items on tabBatch visible
                gbBatchMeasurements.Visible = true;
                //  Displays a message to the user, showing how many objects
                //  were found.
                lblBatchInfo.Text = "Processing Done. " + found.ToString() + "
                   objects found in " + total.ToString() + " images.";
}
```

1.6.17 Batch Queue 1

```
//  This subroutine is responsible for processing queue number 1 during
//  Batch File processing.
private void BatchFileQueue1()
{
        //  Integer Variables used to store the highest and lowest x and y
        //  values returned by the PerimeterAgent.
        int topx,topy,bottomx,bottomy;
        topx = topy = bottomx = bottomy = 0;
        //  Bitmap variable used to temporarily store the EdgeGraph generated
        //  by the Batch File process.
        Bitmap Current;
        //  Sets the image of picturebox picQueue1 to the current item in
        //  listbox lbBatchQueue1.
        picQueue1.Image = (Bitmap)
        Bitmap.FromFile(lbBatchQueue1.Items[0].ToString());
        //  Creates a temporary bitmap variable and set it to the current item
        //  in listbox lbBatchQueue1.
        Bitmap temp = (Bitmap)
                Bitmap.FromFile(lbBatchQueue1.Items[0].ToString());
        //  Initializes the total image and object found counters
        TotalQueue1 = FoundQueue1 = 0;
        //  Initializes the low values to their highest possible value and sets the
        //  high values to zero.  This is done to enable the system, to later on, find
        //  the respective lowest and highest values for each variable.
        PerimeterLowQueue1 = AreaLowQueue1 = ShortestLowQueue1 =
           LongestLowQueue1 = EdgesLowQueue1= Int32.MaxValue;
        PerimeterHighQueue1 = AreaHighQueue1 = ShortestHighQueue1 =
```

```
    LongestHighQueue1 = EdgesHighQueue1 = 0;
//  Makes the queue's progress bar visible and initializes it.
pbQueue1.Visible = true;
pbQueue1.Value = 0;
//  Bitmap variable used as temporary storage for the bitmap returned by
//  the ToBinaryAgent
Bitmap Intermediate;
//  ArrayList variable used to store the EdgeGraph ArrayList returned
//  by the EdgeGraphAgent.
ArrayList NewGraphs;
//  Creates the shared Whiteboard object. Then initializes all of the agents
//  and loads them onto individual threads.
whiteboard = new Whiteboard((Bitmap) temp.Clone());
MEDagent = new  MainEdgeDetectionAgent(640,480,whiteboard);
MEDthread = new Thread(new ThreadStart(MEDagent.StartWait));
MEDthread.Start();
ICagent = new InvertColoursAgent(640,480,whiteboard);
ICthread = new Thread(new ThreadStart(ICagent.StartWait));
ICthread.Start();
TBagent = new ToBinaryAgent(640,480,whiteboard);
TBthread = new Thread(new ThreadStart(TBagent.StartWait));
TBthread.Start();
Pagent = new PerimeterAgent(whiteboard,5,5,635,475,1);
Pthread = new Thread(new ThreadStart(Pagent.StartWait));
Pthread.Start();
EGagent = new EdgeGraphAgent(whiteboard,5,435,5,635);
EGthread = new Thread(new ThreadStart(EGagent.StartWait));
EGthread.Start();
Whiteboard.Increment();
//  Process every image int the queue.
for(int i = 0; i < lbBatchQueue1.Items.Count; i++)
{
        //  Increments the total number of images processed in the queue.
        TotalQueue1++;
        //  Update the form.
        this.Refresh();
        //  Sets the image of picturebox picQueue1 to the current item in
        //  listbox lbBatchQueue1.
        picQueue1.Image = (Bitmap)
            Bitmap.FromFile(lbBatchQueue1.Items[i].ToString());
        //  Creates a temporary bitmap variable and set it to the current item
        //  in listbox lbBatchQueue1.
        Bitmap temp = (Bitmap)
            Bitmap.FromFile(lbBatchQueue1.Items[i].ToString());
        //  Show the user that the program is processing.
        System.Windows.Forms.Cursor.Current =
```

```
            System.Windows.Forms.Cursors.WaitCursor;
        //  Start the Detection process.
        while (whiteboard.RunStartWait == 0)
{
  if (whiteboard.Counter > 6)
  {
    if (Pagent.getPerimeter() != 0)
    {
            //  Sets the values for the bounding box.
            topx = Pagent.getLowestX();
            topy = Pagent.getLowestY();
            bottomx = Pagent.getHighestX();
            bottomy = Pagent.getHighestY();
            //  Gets the lowest and highest values from the
            //  PerimeterAgent.
            if (Pagent.getPerimeter() < PerimeterLowQueue1)
                PerimeterLowQueue1 = Pagent.getPerimeter();
            if (Pagent.getPerimeter() > PerimeterHighQueue1)
                PerimeterHighQueue1 = Pagent.getPerimeter();
            if (Pagent.getArea() < AreaLowQueue1)
                AreaLowQueue1 = Pagent.getArea();
            if (Pagent.getArea() > AreaHighQueue1)
                AreaHighQueue1 = Pagent.getArea();
            if (Pagent.getXDistance() < ShortestLowQueue1)
                ShortestLowQueue1 = Pagent.getXDistance();
            if (Pagent.getXDistance() > ShortestHighQueue1)
                ShortestHighQueue1 = Pagent.getXDistance();
            if (Pagent.getYDistance() < LongestLowQueue1)
                LongestLowQueue1 = Pagent.getYDistance();
            if (Pagent.getYDistance() > LongestHighQueue1)
                LongestHighQueue1 = Pagent.getYDistance();
            if (EGagent.getEdgeCount() < EdgesLowQueue1)
                EdgesLowQueue1 = EGagent.getEdgeCount();
            if (EGagent.getEdgeCount() > EdgesHighQueue1)
                EdgesHighQueue1 = EGagent.getEdgeCount();
            //  An object was found.
            //  Increment the number of objects found.
            FoundQueue1++;
            // Checks to see whether this is the first image processed.
            if ((i == 0) && (picCurrentEdges.Image == null))
            {
                // It is, so set the image of picCurrentEdges to the
                //  current image's Edge graph and sizes it
                //  appropriately.
                picCurrentEdges.Image = EGagent.getGraph();
                picCurrentEdges.SizeMode =
```

```
                        PictureBoxSizeMode.StretchImage;
                    }
                    //  Assigns all of the Graph Edge strings, generated by the
                    //  EdgeGraphAgent to the ArrayList NewGraphs.
                    NewGraphs = EGagent.getGraphs();
                    //  Adds all of the items in the NewGraphs ArrayList to the
                    //  CurrentGraphsQueue1 ArrayList.
                    foreach(string a in NewGraphs)
                        CurrentGraphsQueue1.Add(a);
                    //  Assigns the newly created Graph image to the temporary
                    //  bitmap variable Current.  The file is then saved to the
                    //  model's "Batch"  directory.
                    BatchGraphs.Add(EGagent.getGraph());
                    Current = EGagent.getGraph();
                    Current.Save(RootDirectory + "\\Batch\\" +
                        System.DateTime.Now.Day.ToString() +
                        System.DateTime.Now.Month.ToString() +
                        System.DateTime.Now.Year.ToString() +
                        System.DateTime.Now.Minute.ToString() +
                        System.DateTime.Now.Second.ToString() +
                        Thread.CurrentThread.Name.ToString() +
                        ".jpg",System.Drawing.Imaging.ImageFormat.Jpeg);
                }
                //  Increment the queue's progress bar, to show that another image
                //  has been processed.
                pbQueue1.Increment(100 / lbBatchQueue1.Items.Count);
                pbQueue1.Refresh();
                //  Increment frmMain's progress bar, to show that another image
                //  has been processed.
                pbProgress.Value = pbProgress.Value + ProgressIncrementer;
                pbProgress.Refresh();
                whiteboard.Restart();
            }
            //  Sets the queue's progress bar to 100%, indicating that processing
            //  is complete.  The progress bar is then hidden.
            pbQueue1.Value = 100;
            pbQueue1.Refresh();
            pbQueue1.Visible = false;
        }
    }
    //  Ends the agents' waiting process and stops the agents.
    whiteboard.RunStartWait = 1;
    MEDthread.Join();
    ICthread.Join();
    TBthread.Join();
    Pthread.Join();
```

```
    EGthread.Join();
}
```

1.6.18 Batch Queue 2

```
//  This subroutine is responsible for processing queue number 2 during
//  Batch File processing.
private void BatchFileQueue2()
{
        //  Integer Variables used to store the highest and lowest x and y
        //  values returned by the PerimeterAgent.
        int topx,topy,bottomx,bottomy;
        topx = topy = bottomx = bottomy = 0;
        //  Bitmap variable used to temporarily store the EdgeGraph generated
        //  by the Batch File process.
        Bitmap Current;
        //  Sets the image of picturebox picQueue2 to the current item in
        //  listbox lbBatchQueue2.
        picQueue2.Image = (Bitmap)
        Bitmap.FromFile(lbBatchQueue2.Items[0].ToString());
        //  Creates a temporary bitmap variable and set it to the current item
        //  in listbox lbBatchQueue2.
        Bitmap temp = (Bitmap)
                Bitmap.FromFile(lbBatchQueue2.Items[0].ToString());
        //  Initializes the total image and object found counters
        TotalQueue2 = FoundQueue2 = 0;
        //  Initializes the low values to their highest possible value and sets the
        //  high values to zero.  This is done to enable the system, to later on, find
        //  the respective lowest and highest values for each variable.
        PerimeterLowQueue2 = AreaLowQueue2 = ShortestLowQueue2 =
           LongestLowQueue2 = EdgesLowQueue2= Int32.MaxValue;
        PerimeterHighQueue2 = AreaHighQueue2 = ShortestHighQueue2 =
           LongestHighQueue2 = EdgesHighQueue2 = 0;
        //  Makes the queue's progress bar visible and initializes it.
        pbQueue2.Visible = true;
        pbQueue2.Value = 0;
        //  Bitmap variable used as temporary storage for the bitmap returned by
        //  the ToBinaryAgent
        Bitmap Intermediate;
        //  ArrayList variable used to store the EdgeGraph ArrayList returned
        //  by the EdgeGraphAgent.
        ArrayList NewGraphs;
        //  Creates the shared Whiteboard object. Then initializes all of the agents
        //  and loads them onto individual threads.
        Whiteboard2 = new Whiteboard((Bitmap) temp.Clone());
        MEDagent2 = new  MainEdgeDetectionAgent(640,480,whiteboard);
```

```
MEDthread2 = new Thread(new ThreadStart(MEDagent.StartWait));
MEDthread2.Start();
ICagent2 = new InvertColoursAgent(640,480,whiteboard);
ICthread2 = new Thread(new ThreadStart(ICagent.StartWait));
ICthread2.Start();
TBagent2 = new ToBinaryAgent(640,480,whiteboard);
TBthread2 = new Thread(new ThreadStart(TBagent.StartWait));
TBthread2.Start();
Pagent2 = new PerimeterAgent(whiteboard,5,5,635,475,1);
Pthread2 = new Thread(new ThreadStart(Pagent.StartWait));
Pthread2.Start();
EGagent2 = new EdgeGraphAgent(whiteboard,5,435,5,635);
EGthread2 = new Thread(new ThreadStart(EGagent.StartWait));
EGthread2.Start();
Whiteboard2.Increment();
//  Process every image int the queue.
for(int i = 0; i < lbBatchQueue2.Items.Count; i++)
{
        //  Increments the total number of images processed in the queue.
        TotalQueue2++;
        //  Update the form.
        this.Refresh();
        //  Sets the image of picturebox picQueue2 to the current item in
        //  listbox lbBatchQueue2.
        picQueue2.Image = (Bitmap)
            Bitmap.FromFile(lbBatchQueue2.Items[i].ToString());
        //  Creates a temporary bitmap variable and set it to the current item
        //  in listbox lbBatchQueue2.
        Bitmap temp = (Bitmap)
            Bitmap.FromFile(lbBatchQueue2.Items[i].ToString());
        //  Show the user that the program is processing.
        System.Windows.Forms.Cursor.Current =
              System.Windows.Forms.Cursors.WaitCursor;
        //  Start the Detection process.
        while (whiteboard.RunStartWait == 0)
    {
      if (whiteboard.Counter > 6)
      {
        if (Pagent.getPerimeter() != 0)
        {
                //  Sets the values for the bounding box.
                topx = Pagent.getLowestX();
                topy = Pagent.getLowestY();
                bottomx = Pagent.getHighestX();
                bottomy = Pagent.getHighestY();
                //  Gets the lowest and highest values from the
```

```
//  PerimeterAgent.
if (Pagent.getPerimeter() < PerimeterLowQueue2)
        PerimeterLowQueue2 = Pagent.getPerimeter();
if (Pagent.getPerimeter() > PerimeterHighQueue2)
        PerimeterHighQueue2 = Pagent.getPerimeter();
if (Pagent.getArea() < AreaLowQueue2)
        AreaLowQueue2 = Pagent.getArea();
if (Pagent.getArea() > AreaHighQueue2)
        AreaHighQueue2 = Pagent.getArea();
if (Pagent.getXDistance() < ShortestLowQueue2)
        ShortestLowQueue2 = Pagent.getXDistance();
if (Pagent.getXDistance() > ShortestHighQueue2)
        ShortestHighQueue2 = Pagent.getXDistance();
if (Pagent.getYDistance() < LongestLowQueue2)
        LongestLowQueue2 = Pagent.getYDistance();
if (Pagent.getYDistance() > LongestHighQueue2)
        LongestHighQueue2 = Pagent.getYDistance();
if (EGagent.getEdgeCount() < EdgesLowQueue2)
        EdgesLowQueue2 = EGagent.getEdgeCount();
if (EGagent.getEdgeCount() > EdgesHighQueue2)
        EdgesHighQueue2 = EGagent.getEdgeCount();
//  An object was found.
//  Increment the number of objects found.
FoundQueue2++;
// Checks to see whether this is the first image processed.
if ((i == 0) && (picCurrentEdges.Image == null))
{
        // It is, so set the image of picCurrentEdges to the
        //  current image's Edge graph and sizes it
        //  appropriately.
        picCurrentEdges.Image = EGagent.getGraph();
        picCurrentEdges.SizeMode =
            PictureBoxSizeMode.StretchImage;
}
//  Assigns all of the Graph Edge strings, generated by the
//  EdgeGraphAgent to the ArrayList NewGraphs.
NewGraphs = EGagent.getGraphs();
//  Adds all of the items in the NewGraphs ArrayList to the
//  CurrentGraphsQueue2 ArrayList.
foreach(string a in NewGraphs)
        CurrentGraphsQueue2.Add(a);
//  Assigns the newly created Graph image to the temporary
 //  bitmap variable Current.  The file is then saved to the
 //  model's "Batch"  directory.
BatchGraphs.Add(EGagent.getGraph());
Current = EGagent.getGraph();
```

```
                    Current.Save(RootDirectory + "\\Batch\\" +
                        System.DateTime.Now.Day.ToString() +
                        System.DateTime.Now.Month.ToString() +
                        System.DateTime.Now.Year.ToString() +
                        System.DateTime.Now.Minute.ToString() +
                        System.DateTime.Now.Second.ToString() +
                        Thread.CurrentThread.Name.ToString() +
                        ".jpg",System.Drawing.Imaging.ImageFormat.Jpeg);
                }
                //  Increment the queue's progress bar, to show that another image
                //  has been processed.
                pbQueue2.Increment(100 / lbBatchQueue2.Items.Count);
                pbQueue2.Refresh();
                //  Increment frmMain's progress bar, to show that another image
                //  has been processed.
                pbProgress.Value = pbProgress.Value + ProgressIncrementer;
                pbProgress.Refresh();
                whiteboard2.Restart();
            }
            //  Sets the queue's progress bar to 100%, indicating that processing
            //  is complete.  The progress bar is then hidden.
            pbQueue2.Value = 100;
            pbQueue2.Refresh();
            pbQueue2.Visible = false;
          }
        }
        //  Ends the agents' waiting process and stops the agents.
        Whiteboard2.RunStartWait = 1;
        MEDthread2.Join();
        ICthread2.Join();
        TBthread2.Join();
        Pthread2.Join();
        EGthread2.Join();
}
```

1.6.19 Batch Queue 3

```
//  This subroutine is responsible for processing queue number 3 during
//  Batch File processing.
private void BatchFileQueue3()
{
        //  Integer Variables used to store the highest and lowest x and y
        //  values returned by the PerimeterAgent.
        int topx,topy,bottomx,bottomy;
        topx = topy = bottomx = bottomy = 0;
        //  Bitmap variable used to temporarily store the EdgeGraph generated
```

```
//  by the Batch File process.
Bitmap Current;
//  Sets the image of picturebox picQueue3 to the current item in
//  listbox lbBatchQueue3.
picQueue3.Image = (Bitmap)
Bitmap.FromFile(lbBatchQueue3.Items[0].ToString());
//  Creates a temporary bitmap variable and set it to the current item
//  in listbox lbBatchQueue3.
Bitmap temp = (Bitmap)
            Bitmap.FromFile(lbBatchQueue3.Items[0].ToString());
//  Initializes the total image and object found counters
TotalQueue3 = FoundQueue3 = 0;
//  Initializes the low values to their highest possible value and sets the
//  high values to zero.  This is done to enable the system, to later on, find
//  the respective lowest and highest values for each variable.
PerimeterLowQueue3 = AreaLowQueue3 = ShortestLowQueue3 =
    LongestLowQueue3 = EdgesLowQueue3= Int32.MaxValue;
PerimeterHighQueue3 = AreaHighQueue3 = ShortestHighQueue3 =
    LongestHighQueue3 = EdgesHighQueue3 = 0;
//  Makes the queue's progress bar visible and initializes it.
pbQueue3.Visible = true;
pbQueue3.Value = 0;
//  Bitmap variable used as temporary storage for the bitmap returned by
//  the ToBinaryAgent
Bitmap Intermediate;
//  ArrayList variable used to store the EdgeGraph ArrayList returned
//  by the EdgeGraphAgent.
ArrayList NewGraphs;
//  Creates the shared Whiteboard object. Then initializes all of the agents
//  and loads them onto individual threads.
Whiteboard3 = new Whiteboard((Bitmap) temp.Clone());
MEDagent3 = new  MainEdgeDetectionAgent(640,480,whiteboard);
MEDthread3 = new Thread(new ThreadStart(MEDagent.StartWait));
MEDthread3.Start();
ICagent3 = new InvertColoursAgent(640,480,whiteboard);
ICthread3 = new Thread(new ThreadStart(ICagent.StartWait));
ICthread3.Start();
TBagent3 = new ToBinaryAgent(640,480,whiteboard);
TBthread3 = new Thread(new ThreadStart(TBagent.StartWait));
TBthread3.Start();
Pagent3 = new PerimeterAgent(whiteboard,5,5,635,475,1);
Pthread3 = new Thread(new ThreadStart(Pagent.StartWait));
Pthread3.Start();
EGagent3 = new EdgeGraphAgent(whiteboard,5,435,5,635);
EGthread3 = new Thread(new ThreadStart(EGagent.StartWait));
EGthread3.Start();
```

```
Whiteboard3.Increment();
//  Process every image int the queue.
for(int i = 0; i < lbBatchQueue3.Items.Count; i++)
{
        //  Increments the total number of images processed in the queue.
        TotalQueue3++;
        //  Update the form.
        this.Refresh();
        //  Sets the image of picturebox picQueue3 to the current item in
        //  listbox lbBatchQueue3.
        picQueue3.Image = (Bitmap)
            Bitmap.FromFile(lbBatchQueue3.Items[i].ToString());
        //  Creates a temporary bitmap variable and set it to the current item
        //  in listbox lbBatchQueue3.
        Bitmap temp = (Bitmap)
            Bitmap.FromFile(lbBatchQueue3.Items[i].ToString());
        //  Show the user that the program is processing.
        System.Windows.Forms.Cursor.Current =
              System.Windows.Forms.Cursors.WaitCursor;
        //  Start the Detection process.
        while (whiteboard.RunStartWait == 0)
    {
      if (whiteboard.Counter > 6)
      {
        if (Pagent.getPerimeter() != 0)
        {
                //  Sets the values for the bounding box.
                topx = Pagent.getLowestX();
                topy = Pagent.getLowestY();
                bottomx = Pagent.getHighestX();
                bottomy = Pagent.getHighestY();
                //  Gets the lowest and highest values from the
                //  PerimeterAgent.
                if (Pagent.getPerimeter() < PerimeterLowQueue3)
                      PerimeterLowQueue3 = Pagent.getPerimeter();
                if (Pagent.getPerimeter() > PerimeterHighQueue3)
                      PerimeterHighQueue3 = Pagent.getPerimeter();
                if (Pagent.getArea() < AreaLowQueue3)
                      AreaLowQueue3 = Pagent.getArea();
                if (Pagent.getArea() > AreaHighQueue3)
                      AreaHighQueue3 = Pagent.getArea();
                if (Pagent.getXDistance() < ShortestLowQueue3)
                      ShortestLowQueue3 = Pagent.getXDistance();
                if (Pagent.getXDistance() > ShortestHighQueue3)
                      ShortestHighQueue3 = Pagent.getXDistance();
                if (Pagent.getYDistance() < LongestLowQueue3)
```

```
                LongestLowQueue3 = Pagent.getYDistance();
            if (Pagent.getYDistance() > LongestHighQueue3)
                LongestHighQueue3 = Pagent.getYDistance();
            if (EGagent.getEdgeCount() < EdgesLowQueue3)
                EdgesLowQueue3 = EGagent.getEdgeCount();
            if (EGagent.getEdgeCount() > EdgesHighQueue3)
                EdgesHighQueue3 = EGagent.getEdgeCount();
            //  An object was found.
            //  Increment the number of objects found.
            FoundQueue3++;
            // Checks to see whether this is the first image processed.
            if ((i == 0) && (picCurrentEdges.Image == null))
            {
                // It is, so set the image of picCurrentEdges to the
                //  current image's Edge graph and sizes it
                //  appropriately.
                picCurrentEdges.Image = EGagent.getGraph();
                picCurrentEdges.SizeMode =
                    PictureBoxSizeMode.StretchImage;
            }
            //  Assigns all of the Graph Edge strings, generated by the
            //  EdgeGraphAgent to the ArrayList NewGraphs.
            NewGraphs = EGagent.getGraphs();
            //  Adds all of the items in the NewGraphs ArrayList to the
            //  CurrentGraphsQueue3 ArrayList.
            foreach(string a in NewGraphs)
                CurrentGraphsQueue3.Add(a);
            //  Assigns the newly created Graph image to the temporary
            //  bitmap variable Current.  The file is then saved to the
            //  model's "Batch"  directory.
            BatchGraphs.Add(EGagent.getGraph());
            Current = EGagent.getGraph();
            Current.Save(RootDirectory + "\\Batch\\" +
                System.DateTime.Now.Day.ToString() +
                System.DateTime.Now.Month.ToString() +
                System.DateTime.Now.Year.ToString() +
                System.DateTime.Now.Minute.ToString() +
                System.DateTime.Now.Second.ToString() +
                Thread.CurrentThread.Name.ToString() +
                ".jpg",System.Drawing.Imaging.ImageFormat.Jpeg);
        }
        //  Increment the queue's progress bar, to show that another image
        //  has been processed.
        pbQueue3.Increment(100 / lbBatchQueue3.Items.Count);
        pbQueue3.Refresh();
        //  Increment frmMain's progress bar, to show that another image
```

```
                //  has been processed.
                pbProgress.Value = pbProgress.Value + ProgressIncrementer;
                pbProgress.Refresh();
                whiteboard3.Restart();
            }
            //  Sets the queue's progress bar to 100%, indicating that processing
            //  is complete.  The progress bar is then hidden.
            pbQueue3.Value = 100;
            pbQueue3.Refresh();
            pbQueue3.Visible = false;
          }
        }
        //  Ends the agents' waiting process and stops the agents.
        Whiteboard3.RunStartWait = 1;
        MEDthread3.Join();
        ICthread3.Join();
       TBthread3.Join();
        Pthread3.Join();
        EGthread3.Join();
}
```

1.6.20 Batch Queue 4

```
//  This subroutine is responsible for processing queue number 4 during
//  Batch File processing.
private void BatchFileQueue4()
{
        //  Integer Variables used to store the highest and lowest x and y
        //  values returned by the PerimeterAgent.
        int topx,topy,bottomx,bottomy;
        topx = topy = bottomx = bottomy = 0;
        //  Bitmap variable used to temporarily store the EdgeGraph generated
        //  by the Batch File process.
        Bitmap Current;
        //  Sets the image of picturebox picQueue4 to the current item in
        //  listbox lbBatchQueue4.
        picQueue4.Image = (Bitmap)
        Bitmap.FromFile(lbBatchQueue4.Items[0].ToString());
        //  Creates a temporary bitmap variable and set it to the current item
        //  in listbox lbBatchQueue4.
        Bitmap temp = (Bitmap)
                Bitmap.FromFile(lbBatchQueue4.Items[0].ToString());
        //  Initializes the total image and object found counters
        TotalQueue4 = FoundQueue4 = 0;
        //  Initializes the low values to their highest possible value and sets the
        //  high values to zero.  This is done to enable the system, to later on, find
```

```
//  the respective lowest and highest values for each variable.
PerimeterLowQueue4 = AreaLowQueue4 = ShortestLowQueue4 =
    LongestLowQueue4 = EdgesLowQueue4= Int32.MaxValue;
PerimeterHighQueue4 = AreaHighQueue4 = ShortestHighQueue4 =
    LongestHighQueue4 = EdgesHighQueue4 = 0;
//  Makes the queue's progress bar visible and initializes it.
pbQueue4.Visible = true;
pbQueue4.Value = 0;
//  Bitmap variable used as temporary storage for the bitmap returned by
//  the ToBinaryAgent
Bitmap Intermediate;
//  ArrayList variable used to store the EdgeGraph ArrayList returned
//  by the EdgeGraphAgent.
ArrayList NewGraphs;
//  Creates the shared Whiteboard object. Then initializes all of the agents
//  and loads them onto individual threads.
Whiteboard4 = new Whiteboard((Bitmap) temp.Clone());
MEDagent4 = new  MainEdgeDetectionAgent(640,480,whiteboard);
MEDthread4 = new Thread(new ThreadStart(MEDagent.StartWait));
MEDthread4.Start();
ICagent4 = new InvertColoursAgent(640,480,whiteboard);
ICthread4 = new Thread(new ThreadStart(ICagent.StartWait));
ICthread4.Start();
TBagent4 = new ToBinaryAgent(640,480,whiteboard);
TBthread4 = new Thread(new ThreadStart(TBagent.StartWait));
TBthread4.Start();
Pagent4 = new PerimeterAgent(whiteboard,5,5,635,475,1);
Pthread4 = new Thread(new ThreadStart(Pagent.StartWait));
Pthread4.Start();
EGagent4 = new EdgeGraphAgent(whiteboard,5,435,5,645);
EGthread4 = new Thread(new ThreadStart(EGagent.StartWait));
EGthread4.Start();
Whiteboard4.Increment();
//  Process every image int the queue.
for(int i = 0; i < lbBatchQueue4.Items.Count; i++)
{
        //  Increments the total number of images processed in the queue.
        TotalQueue4++;
        //  Update the form.
        this.Refresh();
       //  Sets the image of picturebox picQueue4 to the current item in
       //  listbox lbBatchQueue4.
        picQueue4.Image = (Bitmap)
            Bitmap.FromFile(lbBatchQueue4.Items[i].ToString());
       //  Creates a temporary bitmap variable and set it to the current item
       //  in listbox lbBatchQueue4.
```

```
    Bitmap temp = (Bitmap)
        Bitmap.FromFile(lbBatchQueue4.Items[i].ToString());
    //  Show the user that the program is processing.
    System.Windows.Forms.Cursor.Current =
            System.Windows.Forms.Cursors.WaitCursor;
    //  Start the Detection process.
    while (whiteboard.RunStartWait == 0)
{
  if (whiteboard.Counter > 6)
  {
    if (Pagent.getPerimeter() != 0)
    {
            //  Sets the values for the bounding box.
            topx = Pagent.getLowestX();
            topy = Pagent.getLowestY();
            bottomx = Pagent.getHighestX();
            bottomy = Pagent.getHighestY();
            //  Gets the lowest and highest values from the
            //  PerimeterAgent.
            if (Pagent.getPerimeter() < PerimeterLowQueue4)
                    PerimeterLowQueue4 = Pagent.getPerimeter();
            if (Pagent.getPerimeter() > PerimeterHighQueue4)
                    PerimeterHighQueue4 = Pagent.getPerimeter();
            if (Pagent.getArea() < AreaLowQueue4)
                    AreaLowQueue4 = Pagent.getArea();
            if (Pagent.getArea() > AreaHighQueue4)
                    AreaHighQueue4 = Pagent.getArea();
            if (Pagent.getXDistance() < ShortestLowQueue4)
                    ShortestLowQueue4 = Pagent.getXDistance();
            if (Pagent.getXDistance() > ShortestHighQueue4)
                    ShortestHighQueue4 = Pagent.getXDistance();
            if (Pagent.getYDistance() < LongestLowQueue4)
                    LongestLowQueue4 = Pagent.getYDistance();
            if (Pagent.getYDistance() > LongestHighQueue4)
                    LongestHighQueue4 = Pagent.getYDistance();
            if (EGagent.getEdgeCount() < EdgesLowQueue4)
                    EdgesLowQueue4 = EGagent.getEdgeCount();
            if (EGagent.getEdgeCount() > EdgesHighQueue4)
                    EdgesHighQueue4 = EGagent.getEdgeCount();
            //  An object was found.
            //  Increment the number of objects found.
            FoundQueue4++;
            // Checks to see whether this is the first image processed.
            if ((i == 0) && (picCurrentEdges.Image == null))
            {
                    // It is, so set the image of picCurrentEdges to the
```

```
                        //  current image's Edge graph and sizes it
                        //  appropriately.
                        picCurrentEdges.Image = EGagent.getGraph();
                        picCurrentEdges.SizeMode =
                            PictureBoxSizeMode.StretchImage;
                    }
                    //  Assigns all of the Graph Edge strings, generated by the
                    //  EdgeGraphAgent to the ArrayList NewGraphs.
                    NewGraphs = EGagent.getGraphs();
                    //  Adds all of the items in the NewGraphs ArrayList to the
                    //  CurrentGraphsQueue4 ArrayList.
                    foreach(string a in NewGraphs)
                        CurrentGraphsQueue4.Add(a);
                    //  Assigns the newly created Graph image to the temporary
                    //  bitmap variable Current.  The file is then saved to the
                    //  model's "Batch"  directory.
                    BatchGraphs.Add(EGagent.getGraph());
                    Current = EGagent.getGraph();
                    Current.Save(RootDirectory + "\\Batch\\" +
                        System.DateTime.Now.Day.ToString() +
                        System.DateTime.Now.Month.ToString() +
                        System.DateTime.Now.Year.ToString() +
                        System.DateTime.Now.Minute.ToString() +
                        System.DateTime.Now.Second.ToString() +
                        Thread.CurrentThread.Name.ToString() +
                        ".jpg",System.Drawing.Imaging.ImageFormat.Jpeg);
                }
                //  Increment the queue's progress bar, to show that another image
                //  has been processed.
                pbQueue4.Increment(100 / lbBatchQueue4.Items.Count);
                pbQueue4.Refresh();
                //  Increment frmMain's progress bar, to show that another image
                //  has been processed.
                pbProgress.Value = pbProgress.Value + ProgressIncrementer;
                pbProgress.Refresh();
                whiteboard4.Restart();
            }
            //  Sets the queue's progress bar to 100%, indicating that processing
            //  is complete.  The progress bar is then hidden.
            pbQueue4.Value = 100;
            pbQueue4.Refresh();
            pbQueue4.Visible = false;
        }
    }
    //  Ends the agents' waiting process and stops the agents.
    Whiteboard4.RunStartWait = 1;
```

```
      MEDthread4.Join();
      ICthread4.Join();
     TBthread4.Join();
      Pthread4.Join();
      EGthread4.Join();
}
```

1.6.21 Update the Model (Single Image Mode)

```
//  This subroutine is called during Single File Processing mode, to update the
//  model's .mdl-file.
private void UpdateModelFile(int Perimeter, int Area, int Shortest, int Longest, int
                Edges)
{
        // Makes sure no other process is accessing the subroutine.
        lock(this)
        {
                //  TextWriter variable for writing to the model's .mdl-file.
                TextWriter tw;
                tw = new StreamWriter(RootDirectory + "\\" + ModelName + ".mdl");
                //  If the following variables don't have assigned values, they
                //  are assigned default values.
                if(PerimeterLow == 0)
                        PerimeterLow = Int32.MaxValue;
                if(PerimeterHigh == 0)
                        PerimeterHigh = 0;
                if(AreaLow == 0)
                        AreaLow = Int32.MaxValue;
                if(AreaHigh == 0)
                        AreaHigh = 0;
                if(ShortestLow == 0)
                        ShortestLow = Int32.MaxValue;
                if(ShortestHigh == 0)
                        ShortestHigh = 0;
                if(LongestLow == 0)
                        LongestLow = Int32.MaxValue;
                if(LongestHigh == 0)
                        LongestHigh = 0;
                if(EdgesLow == 0)
                        EdgesLow = Int32.MaxValue;
                if(EdgesHigh == 0)
                        EdgesHigh = 0;
                //  Runs through the values generated, to see
                //  if they will replace the current values.
                if (Perimeter < PerimeterLow)
                        PerimeterLow = Perimeter;
```

```
                if (Perimeter > PerimeterHigh)
                        PerimeterHigh = Perimeter;
                if (Area < AreaLow)
                        AreaLow = Area;
                if (Area > AreaHigh)
                        AreaHigh = Area;
                if (Shortest < ShortestLow)
                        ShortestLow = Shortest;
                if (Shortest > ShortestHigh)
                        ShortestHigh = Shortest;
                if (Longest < LongestLow)
                        LongestLow = Longest;
                if (Longest > LongestHigh)
                        LongestHigh = Longest;
                if (Longest < LongestLow)
                        LongestLow = Longest;
                if (Longest > LongestHigh)
                        LongestHigh = Longest;
                if (Edges < EdgesLow)
                        EdgesLow = Edges;
                if (Edges > EdgesHigh)
                        EdgesHigh = Edges;
                //  Write all of the values to the model's .mdl-file.
                tw.WriteLine(PerimeterLow);
                tw.WriteLine(PerimeterHigh);
                tw.WriteLine(AreaLow);
                tw.WriteLine(AreaHigh);
                tw.WriteLine(ShortestLow);
                tw.WriteLine(ShortestHigh);
                tw.WriteLine(LongestLow);
                tw.WriteLine(LongestHigh);
                tw.WriteLine(EdgesLow);
                tw.WriteLine(EdgesHigh);
                //  Closes the .mdl-file.
                tw.Close();
        }
}
```

1.6.22 Adjust Brightness

```
//  This subroutine is called when the 'Adjust Brightness' button is clicked on
//  frmUpdateModel's toolbar.
private void AdjustBrightness()
{
        //  Calls frmBrightness and opens it as a dialog.
        frmBrightness Brightness = new frmBrightness(bitmap,picStudy);
```

```
        Brightness.ShowDialog();
}
```

1.6.23 Adjust Contrast

```
//  This subroutine is called when the 'Adjust Contrast' button is clicked on
//  frmUpdateModel's toolbar.
private void AdjustContrast()
{
        //  Calls frmContrast and opens it as a dialog.
        frmContrast Contrast = new frmContrast(bitmap,picStudy);
        Contrast.ShowDialog();
}
```

1.6.24 Select Camera

```
//  This subroutine is called whenever a capture device needs to be
//  selected.
private void CameraMode()
{
         //  Tries to load a capture device.  If an error is found, an appropriate
        //  message is shown to the user.
        try
        {
                //  Create a new instance of a WiaClass object.
                wiaManager = new WiaClass();
                //  Create a new Select Device Dialog Box.
                WIA.CommonDialogClass class1 = new
                      WIA.CommonDialogClass();
                //  If any other device is loaded, destroy its reference.
                if (CurrentCam != null)
                        wiaVideo.DestroyVideo();
                //  Show the Select Device Dialog Box and assigns the selected
                //  device to CurrentCam.
                CurrentCam = class1.ShowSelectDevice
                                (WIA.WiaDeviceType.UnspecifiedDeviceType,
                                 true, false);
                //  If a device was selected.
                if (CurrentCam != null)
                {
                        //  Assigns the selected device's name to the DeviceID
                        //  string.
                        string DeviceID = CurrentCam.DeviceID.ToString();
                        //  Initialize variables
                        object foundID = null;
                        CollectionClass wiaDevs = null;
```

```
DeviceInfoClass devInfo = null;
int Found = 0;
//  Assigns all of the current WIA devices in the system to
//  wiaDevs.
wiaDevs = wiaManager.Devices as CollectionClass;
//  If there were devices
if( wiaDevs != null )
{
        //  Run thrugh all of the devices.
        foreach( object wiaObj in wiaDevs )
        {
                //  Stop the loop if the selected device was
                //  found.
                if (Found == 1)
                        break;
                //  Wrap the wiaObj in a DeviceInfoClass
                //  wrapper.
                devInfo = (DeviceInfoClass)
                        Marshal.CreateWrapperOfType
                        (wiaObj, typeof(DeviceInfoClass) );
                //  Test whether the current wiaObj matches
                //  the selected device.
                if (devInfo.Id.ToString() == DeviceID)
                {
                        //  Used to stop the loop.
                        Found = 1;
                        //  Assigns the object's ID to foundID
                        //  and releases any resources used by
                        //  the wiaObj and devInfo.
                        foundID = devInfo.Id;
                        Marshal.ReleaseComObject( wiaObj );
                        Marshal.ReleaseComObject( devInfo );
                }
        }
}

// Reinitialize variables
devInfo = null;
foundID = System.Reflection.Missing.Value;
//  Creates a video capture object using the selected device.
wiaCamera = (ItemClass) wiaManager.Create( ref foundID );
//  May sometimes take a while, so inform the user that the
//  system  is loading.
Cursor.Current = Cursors.WaitCursor;
//  Try to load the video stream, if it fails a relevant message
//  is shown to the user.
```

```
try
{
        //  Create a new instance of a WiaVideoClass object.
        wiaVideo = new WiaVideoClass();
        //  MSDN specifies that the ImagesDirectory should
        //  be set  to the WIA_DPV_IMAGES_DIRECTORY.
        //  This is specified as 3587.
        string videoDir = (string) wiaCamera.GetPropById(
                        (WiaItemPropertyId) 3587 );
        wiaVideo.ImagesDirectory = videoDir;
        //  Assigns the ID of the selected device to the
        //  cameraID string.
        string cameraID = wiaCamera.GetPropById(
                (WiaItemPropertyId) WiaDeviceInfoPropertyId.
                DeviceInfoDevId ) as string;
        //  Reinitialize the tabpage.
        ResetPage();
        //  Accesses system pointers (handles) so the code
        //  needs to be declared unsafe in order
        //  for the compiler to process it.
        unsafe
        {
                //  In order to create the video display,
                //  wiaVideo needs the handle of the
                //  componenent on which the video needs to
                //  be displayed.  This reference needs to be in
                //  the format of a
                //  WIAVideoLib._RemotableHandle,
                //  so the the Handle of pnlCam needs to be
                //  cast to this format.
                WIAVIDEOLib._RemotableHandle *handle =
                        (WIAVIDEOLib._RemotableHandle*)
                        pnlCam.Handle.ToPointer();
                wiaVideo.CreateVideoByWiaDevID(cameraID,
                        ref *handle, 1, 0);
        }
}
catch( Exception )
{
        //  If a wiaVideo object was loaded, release it.
        if( wiaVideo != null )
                Marshal.ReleaseComObject( wiaVideo );
        //  Reinitialize the variable.
        wiaVideo = null;
        //  Display an appropriate message to the user.
        MessageBox.Show( this, "Create video stream failed",
```

```
                                                    "WIA", MessageBoxButtons.OK,
                                                    MessageBoxIcon.Stop );

                        }
                        finally
                        {
                                //  If everything was successful, do the following.

                                // Creates the agent communication directory.
                                if (Directory.Exists(Application.StartupPath +
                                  "\\Recognition") == true)
                                   Directory.Delete(Application.StartupPath +
                                       "\\Recognition",true);
                                Directory.CreateDirectory(Application.StartupPath +
                                       "\\Recognition");
                                Cursor.Current = Cursors.Default;
                                //  Adds the tab page that acts as start page for
                                //  Camera Mode
                                tabPictures.TabPages.Add(tabCamVideo);
                                tabCamVideo.ImageIndex = 3;
                                tabCamVideo.Text = "Camera Mode";
                                lblInfo.Text = "Camera Loaded";
                                picCamVideo.Visible = true;
                                gbCamControls.Visible = true;
                                cmdCamPlayPause.ImageIndex = 0;
                                //  Enables all of the toolbar buttons.
                                tbUpdateModel.Buttons[0].Enabled = true;
                                tbUpdateModel.Buttons[1].Enabled = true;
                                tbUpdateModel.Buttons[2].Enabled = true;
                                tbUpdateModel.Buttons[3].Enabled = true;
                                try
                                {
                                        //  Start the capture device stream.
                                        wiaVideo.Play();
                                }
                                catch( Exception ) {}
                        }
                }
        }
        catch( Exception )
        {
                MessageBox.Show( this, "Camera connection failed.", "Message",
                        MessageBoxButtons.OK, MessageBoxIcon.Stop );
        }
}
```

1.6.25 Camera Play/Pause Button

```
//  This subroutine is called when the capture stream's
//  Play/Pause button is clicked.
private void cmdCamPlayPause_Click(object sender, System.EventArgs e)
{
        //  If the stream is played, pause it.  If it's paused
        //  play it.
        if (cmdCamPlayPause.ImageIndex == 1)
        {
                try
                {
                        wiaVideo.Play();
                        cmdCamPlayPause.ImageIndex = 0;
                }
                catch( Exception ) {}
        }
        else
        {
                try
                {
                        wiaVideo.Pause();
                        cmdCamPlayPause.ImageIndex = 1;
                }
                catch( Exception ) {}
        }
}
```

1.6.26 Select Cam/Video Mode

```
//  This subroutine acts as a switchboard to determine which
//  button was clicked on the Video Options toolbar.
private void tbVideoOptions_ButtonClick(object sender,
            System.Windows.Forms.ToolBarButtonClickEventArgs e)
{
        switch (tbVideoOptions.Buttons.IndexOf(e.Button))
        {
                //  Single Frame Mode.
                case 0:  tbVideoOptions.Buttons[0].Pushed = true;
                        tbVideoOptions.Buttons[1].Pushed = false;
                        lblInfo.Text = "Single Image Mode";
                        cmdActivate.Text = "Capture";
                        cmdActivate.Visible = true;
                        picCapturePreview.Visible = true;
                        gbPerimeter.Visible = false;
                        break;
```

```
                // Continuous Mode.
                case 1:  tbVideoOptions.Buttons[0].Pushed = false;
                        tbVideoOptions.Buttons[1].Pushed = true;
                        lblInfo.Text = "Continuous Mode";
                        cmdActivate.Text = "Activate";
                        cmdActivate.Visible = true;
                        picCapturePreview.Visible = true;
                        gbPerimeter.Visible = true;
                        break;
        }

}
```

1.6.27 Activate Button

```
// This subroutine is called when the Activate button is clicked.
private void cmdActivate_Click(object sender, System.EventArgs e)
{
        // Capture the desktop image.
        int hdcSrc = User32.GetWindowDC(User32.GetDesktopWindow()),
                hdcDest = GDI32.CreateCompatibleDC(hdcSrc),
                hBitmap = GDI32.CreateCompatibleBitmap(hdcSrc,
        GDI32.GetDeviceCaps(hdcSrc,8),GDI32.GetDeviceCaps(hdcSrc,10));
                GDI32.SelectObject(hdcDest,hBitmap);
                GDI32.BitBlt(hdcDest,0,0,GDI32.GetDeviceCaps(hdcSrc,8),
                GDI32.GetDeviceCaps(hdcSrc,10),
                hdcSrc,0,0,0x00CC0020);
        // Create the bitmap image and determine which section of
        // the image should be used.
        Bitmap image =
                new Bitmap(Image.FromHbitmap(new IntPtr(hBitmap)),
                Image.FromHbitmap(new IntPtr(hBitmap)).Width,
                Image.FromHbitmap(new IntPtr(hBitmap)).Height);
        RectangleF a = new RectangleF(7,158,640,480);
        bitmap = (new Bitmap((Bitmap)
                    image.Clone(a,PixelFormat.Format24bppRgb))

        // Starts the Single Frame Mode.
        if (tbVideoOptions.Buttons[0].Pushed == true)
        {
                // Creates the shared Whiteboard object. Then initializes all of the
                // agents and loads them onto individual threads.
                whiteboard = new Whiteboard(bitmap);
                MEDagent = new  MainEdgeDetectionAgent(640,480,whiteboard);
                MEDthread = new Thread(new ThreadStart(MEDagent.StartWait));
                MEDthread.Start();
```

```
            ICagent = new InvertColoursAgent(640,480,whiteboard);
            ICthread = new Thread(new ThreadStart(ICagent.StartWait));
            ICthread.Start();
           TBagent = new ToBinaryAgent(640,480,whiteboard);
           TBthread = new Thread(new ThreadStart(TBagent.StartWait));
           TBthread.Start();
            Pagent = new PerimeterAgent(whiteboard,5,5,635,475,1);
            Pthread = new Thread(new ThreadStart(Pagent.StartWait));
            Pthread.Start();
            EGagent = new EdgeGraphAgent(whiteboard,5,435,5,635);
            EGthread = new Thread(new ThreadStart(EGagent.StartWait));
            EGthread.Start();
            //  Tells the the first agent to start processing.
            Whiteboard.Increment()
            StillRecognitionForCamVideo();
            //  Ends the agents' waiting process and stops the agents.
            whiteboard.RunStartWait = 1;
            MEDthread.Join();
             ICthread.Join();
             TBthread.Join();
             Pthread.Join();
              EGthread.Join();
}

//  Starts the Continuous Mode.
if (tbVideoOptions.Buttons[1].Pushed == true)
{
        //  Determines whether the process should be started
        //  or stopped.
        if (cmdActivate.Text == "Activate")
        {
                tbCamVideoType.Buttons[0].Enabled = false;
                tbCamVideoType.Buttons[0].Enabled = false;
                FlagActivate = 1;
                cmdActivate.Text = "Deactivate";
                //  Creates the shared Whiteboard object. Then initializes all
                //  of the agents and loads them onto individual threads.
                whiteboard = new Whiteboard(bitmap);
                MEDagent = new
                                MainEdgeDetectionAgent(640,480,whiteboard);
                MEDthread = new Thread(new
                                        ThreadStart(MEDagent.StartWait));
                 MEDthread.Start();
                 ICagent = new InvertColoursAgent(640,480,whiteboard);
                 ICthread = new Thread(new
                                ThreadStart(ICagent.StartWait));
```

```
                    ICthread.Start();
                    TBagent = new ToBinaryAgent(640,480,whiteboard);
                    TBthread = new Thread(new
                                ThreadStart(TBagent.StartWait));
                   TBthread.Start();
                   Pagent = new PerimeterAgent(whiteboard,5,5,635,475,1);
                   Pthread = new Thread(new ThreadStart(Pagent.StartWait));
                   Pthread.Start();
                   EGagent = new EdgeGraphAgent(whiteboard,5,435,5,635);
                    EGthread = new Thread(new
                                ThreadStart(EGagent.StartWait));
                   EGthread.Start();
                    //  Tells the the first agent to start processing.
                   Whiteboard.Increment()
                    tmCamVideoCapture.Enabled = true;
            }
            else
            {
                    tmCamVideoCapture.Enabled = false;
                   //  Ends the agents' waiting process and stops the agents.
                   whiteboard.RunStartWait = 1;
                   MEDthread.Join();
                   ICthread.Join();
                   TBthread.Join();
                   Pthread.Join();
                  EGthread.Join();
                   tbCamVideoType.Buttons[0].Enabled = true;
                   tbCamVideoType.Buttons[0].Enabled = true;
                   FlagActivate = 0;
                   cmdActivate.Text = "Activate";
            }
        }
}
```

1.6.28 Cam/Video Single Frame Mode

```
//  This subroutine is called whenever video frame processing is done.

private void StillRecognitionForCamVideo()
{
        //  Show the user that the program is processing.
        System.Windows.Forms.Cursor.Current =
                System.Windows.Forms.Cursors.WaitCursor;

        while (whiteboard.RunStartWait == 0)
        {
```

```
    if (whiteboard.Counter > 6)
    {
         //  Display the found object's perimeter.
        if (Pagent.getPerimeter() != 0)
        {
        //  An object was found.
        //  Set up all the components.
        cmdAcceptCamVideo.Visible = true;
        picCapturePreview.Image = whiteboard.picInUseDone;
        picCapturePreview.SizeMode =
            PictureBoxSizeMode.StretchImage;
        picCapturePreview.BackColor = Color.White;
        lblInfo.Text = "Object Found";
        gbCamVideoMeasurements.Visible = true;
        //  Assigns the newly generated images.
        bmpCurrentEdges = EGagent.getGraph();
        picCurrentEdges.SizeMode = PictureBoxSizeMode.StretchImage;
        //  Assigns the newly generated statistics, for the found object,
        //  to the following labels.
        lblCamVideoPerimeter.Text = Pagent.getPerimeter().ToString();
        lblCamVideoArea.Text = Pagent.getArea().ToString();
        lblCamVideoXDistance.Text = Pagent.getXDistance().ToString();
        lblCamVideoYDistance.Text = Pagent.getYDistance().ToString();
        lblCamVideoEdges.Text = EGagent.getEdgeCount().ToString();
        //  Assign the values to global variables to be saved later.
        SinglePerimeter = Pagent.getPerimeter();
        SingleArea = Pagent.getArea();
        SingleXDistance = Pagent.getXDistance();
        SingleYDistance = Pagent.getYDistance();
        SingleEdgeCount = EGagent.getEdgeCount();
        //  Gets a list of all the previously created Graph images.
        filelist = Directory.GetFiles(RootDirectory,"*.jpg");
        //  Sets up the arrangement for frmUpdateModel's
        //  toolbar buttons.
        tbUpdateModel.Buttons[0].Enabled = true;
        tbUpdateModel.Buttons[1].Enabled = true;
        tbUpdateModel.Buttons[2].Enabled = true;
        tbUpdateModel.Buttons[3].Enabled = true;
}
else
{
        //  No object was found.
        lblInfo.Text = "No object found";
        picCapturePreview.Image = null;
        picCapturePreview.BackColor = Color.FromName("Control");
        gbCamVideoMeasurements.Visible = false;
```

```
        }
      }
    }
}

//  This subroutine is called every time the CamVideoCapture
//  timer expires (500)
```

1.6.29 Cam/VideoCapture timer

```
private void tmCamVideoCapture_Tick(object sender, System.EventArgs e)
{
        //  Show the user that the program is processing.
        System.Windows.Forms.Cursor.Current =
                System.Windows.Forms.Cursors.WaitCursor;
         while (whiteboard.RunStartWait == 0)
        {
              if (whiteboard.Counter > 6)
              {
                 //  Display the found object's perimeter.
                if(numMax.Value <= numMin.Value)
                  numMax.Value = numMax.Maximum;
                  //  If the perimeter is in the Perimeter Textbox range; an object
                  //  was found.
                if ((Pagent.getPerimeter() >= numMin.Value) &&
                (Pagent.getPerimeter() <= numMax.Value))
              {
                //  An object was found.
                //  Set up all the components.
                picCapturePreview.Image = whiteboard.picInUseDone;
                picCapturePreview.SizeMode =
                    PictureBoxSizeMode.StretchImage;
                picCapturePreview.BackColor = Color.White;
                lblInfo.Text = "Object Found";
                gbCamVideoMeasurements.Visible = true;
                bmpCurrentEdges = EGagent.getGraph();
                //  Save the newly created EdgeGraph to disc.
                bmpCurrentEdges.Save(RootDirectory + "\\" +
                      System.DateTime.Now.Day.ToString() +
                         System.DateTime.Now.Month.ToString() +
                         System.DateTime.Now.Year.ToString() +
                         System.DateTime.Now.Minute.ToString() +
                         System.DateTime.Now.Second.ToString() +
                         ".jpg",System.Drawing.Imaging.ImageFormat.Jpeg);
                //  Assigns the newly generated statistics, for the found object,
                //  to the following labels.
                lblCamVideoPerimeter.Text = Pagent.getPerimeter().ToString();
```

```
                lblCamVideoArea.Text = Pagent.getArea().ToString();
                lblCamVideoXDistance.Text = Pagent.getXDistance().ToString();
                lblCamVideoYDistance.Text = Pagent.getYDistance().ToString();
                lblCamVideoEdges.Text = EGagent.getEdgeCount().ToString();
                //  Calls the following subroutine to update the model's .mdl-file.
                UpdateModelFile(Pagent.getPerimeter(),
                      Pagent.getArea(),Pagent.getXDistance(),
                      Pagent.getYDistance(), EGagent.getEdgeCount());
                //  Gets a list of all the previously created Graph images.
                filelist = Directory.GetFiles(RootDirectory,"*.jpg");
                //  Sets up the arrangement for frmUpdateModel's
                //  toolbar buttons.
                tbUpdateModel.Buttons[0].Enabled = true;
                tbUpdateModel.Buttons[1].Enabled = true;
                tbUpdateModel.Buttons[2].Enabled = true;
                tbUpdateModel.Buttons[3].Enabled = true;
            }
            else
            {
                //  An object was found.
                lblInfo.Text = "No object found";
                picCapturePreview.Image = null;
                picCapturePreview.BackColor = Color.FromName("Control");
                gbCamVideoMeasurements.Visible = false;
            }
            //  Capture the desktop image.
            int hdcSrc = User32.GetWindowDC(User32.GetDesktopWindow()),
                hdcDest = GDI32.CreateCompatibleDC(hdcSrc),
                hBitmap = GDI32.CreateCompatibleBitmap(hdcSrc,
                GDI32.GetDeviceCaps(hdcSrc,8),
                GDI32.GetDeviceCaps(hdcSrc,10));
            GDI32.SelectObject(hdcDest,hBitmap);
            GDI32.BitBlt(hdcDest,0,0,GDI32.GetDeviceCaps(hdcSrc,8),
            GDI32.GetDeviceCaps(hdcSrc,10),
                        hdcSrc,0,0,0x00CC0020);
            Bitmap image =
                        new Bitmap(Image.FromHbitmap(new IntPtr(hBitmap)),
                        Image.FromHbitmap(new IntPtr(hBitmap)).Width,
                        Image.FromHbitmap(new IntPtr(hBitmap)).Height);
            RectangleF a = new RectangleF(7,163,640,480);
            bitmap = (new Bitmap((Bitmap)
                    image.Clone(a,PixelFormat.Format24bppRgb))
            whiteboard.Reset();
          }
      }
}
```

1.6.30 Select Video

```
//  This subroutine is called when Video Mode is selected.
//  It allows the user to select a video to be displayed.
private void VideoMode()
{
        //  Determine the setting for the Open File Dialog Box
        //  and display it.
        ofdPicture.Filter = "AVI Files (*.avi)|*.avi|MPEG Files (*.mpg)|*.mpg|All
                            Files (*.*)|*.*";
        ofdPicture.RestoreDirectory = true ;
        if(ofdPicture.ShowDialog() == DialogResult.OK)
        {
                //  Variables used to calculate a video's total
                //  runtime.
                int hours, minutes, seconds;
                //  Reset the page's components.
                ResetPage();
                //  Sets the tabpage's settings.
                tabPictures.TabPages.Remove(tabCamVideo);
                tabCamVideo.ImageIndex = 2;
                tabPictures.TabPages.Add(tabCamVideo);
                gbControls.Visible = true;
                picCamVideo.Visible = true;
                tabCamVideo.Text = "Video Mode:  " + ofdPicture.FileName;
                lblInfo.Text = "Video loaded";
                //  Create the video object.
                CurrentVideo = new Video(ofdPicture.FileName);
                CurrentVideo.Owner = picCamVideo;
                picCamVideo.Width = 640;
                picCamVideo.Height = 480;
                //  Calculate and display the video's runtime.
                seconds = (int)CurrentVideo.Duration % 60;
                minutes = (int)CurrentVideo.Duration / 60 % 60;
                hours = (int)CurrentVideo.Duration /60 / 60;
                if (hours < 10)
                        lblVidTotal.Text = "0" + hours.ToString() + ":";
                else
                        lblVidTotal.Text = hours.ToString() + ":";
                if (minutes < 10)
                        lblVidTotal.Text = lblVidTotal.Text + "0" + minutes.ToString()
                                           + ":";
                else
                        lblVidTotal.Text = lblVidTotal.Text + minutes.ToString() + ":";
                if (seconds < 10)
                        lblVidTotal.Text = lblVidTotal.Text + "0" +
```

```
                                        seconds.ToString();
                else
                        lblVidTotal.Text = lblVidTotal.Text + seconds.ToString();
                //  Sets the Position slider's maximum value.
                tbVideo.Maximum = (int) CurrentVideo.Duration;
                //  Play the video and pause it to display the first frame.
                CurrentVideo.Play();
                CurrentVideo.Pause();
                //  Set a variable to determine if the video is played or
                //  paused.
                PlayPause = 2;
                //  Start the video timer.
                tmVideo.Enabled = true;
                cmdPlayPause.ImageIndex = 0;
                //  Set up the toolbar buttons.
                tbUpdateModel.Buttons[0].Enabled = true;
                tbUpdateModel.Buttons[1].Enabled = true;
                tbUpdateModel.Buttons[2].Enabled = true;
                tbUpdateModel.Buttons[3].Enabled = true;
                // Creates the agent communication directory.
                if (Directory.Exists(Application.StartupPath + "\\Recognition")
                    == true)
                   Directory.Delete(Application.StartupPath + "\\Recognition",true);
                Directory.CreateDirectory(Application.StartupPath +
                   "\\Recognition");

        }
}
```

1.6.31 Video Mute checkbox

```
//  This subroutine is called when the Mute checkbox is clicked.
private void chkMute_CheckedChanged(object sender, System.EventArgs e)
{
        //  Mutes the volume.
        if (chkMute.Checked == true)
        {
                CurrentVideo.Audio.Volume = -10000;
                tbVidVolume.Enabled = false;}
                //  Restores the volume.
        if (chkMute.Checked == false)
        {
                CurrentVideo.Audio.Volume = tbVidVolume.Value;
                tbVidVolume.Enabled = true;}
}
```

1.6.32 Video Position scrollbar

```
//  This subroutine is called when the Video Position slider's
//  value changes.
private void tbVideo_Scroll(object sender, System.EventArgs e)
{
        //  Sets the Video Position's value to the current video
        //  position.
        CurrentVideo.CurrentPosition = tbVideo.Value;
        //  Variables used to calculate the current video runtime.
        int hours, minutes, seconds;

        seconds = (int)CurrentVideo.CurrentPosition % 60;
        minutes = (int)CurrentVideo.CurrentPosition / 60 % 60;
        hours = (int)CurrentVideo.CurrentPosition /60 / 60;
        if (hours < 10)
                lblVidCurrent.Text = "0" + hours.ToString() + ":";
        else
                lblVidCurrent.Text = hours.ToString() + ":";
        if (minutes < 10)
                lblVidCurrent.Text = lblVidCurrent.Text + "0" + minutes.ToString() +
                                        ":";
        else
                lblVidCurrent.Text = lblVidCurrent.Text + minutes.ToString() + ":";
        if (seconds < 10)
                lblVidCurrent.Text = lblVidCurrent.Text + "0" + seconds.ToString();
        else
                lblVidCurrent.Text = lblVidCurrent.Text + seconds.ToString();
}
```

1.6.33 Video Volume scrollbar

```
//  This subroutine is called when the Video Volume slider's value
//  changes.  It assigns the slider's value to the video's volume
//  setting.
private void tbVidVolume_Scroll(object sender, System.EventArgs e)
{
        CurrentVideo.Audio.Volume = tbVidVolume.Value;
}
```

1.6.34 Video Stop Button

```
//  This subroutine is called when the Stop button is clicked.
private void cmdStop_Click(object sender, System.EventArgs e)
{
        //  Stops the video and the timer and resets the controls.
```

```
        tmVideo.Enabled = false;
        CurrentVideo.Stop();
        cmdPlayPause.ImageIndex = 0;
        lblVidCurrent.Text = "00:00:00";
        tbVideo.Value = 0;
        PlayPause = 2;
}
```

1.6.35 Video Play/Pause Button

```
//  This subroutine is called when the PlayPause button is clicked.
private void cmdPlayPause_Click(object sender, System.EventArgs e)
{
        //  Make sure the timer is enabled.
        tmVideo.Enabled = true;
        //  If the video is playing, pause it.  If it's paused,
        //  play it.
        if (PlayPause == 1)
        {
                PlayPause = 2;
                CurrentVideo.Pause();
                cmdPlayPause.ImageIndex = 0;
        }
        else
                if (PlayPause == 2)
                {
                        PlayPause = 1;
                        tmVideo.Enabled = true;
                        CurrentVideo.Play();
                        cmdPlayPause.ImageIndex = 1;
                        cmdStop.Enabled = true;
                }
}
```

1.6.36 Video timer

```
//  This subroutine is called when the Video timer expires (100).
private void tmVideo_Tick(object sender, System.EventArgs e)
{
        //  Variables used to calculate the current video runtime.
        int hours, minutes, seconds;
        seconds = (int)CurrentVideo.CurrentPosition % 60;
        minutes = (int)CurrentVideo.CurrentPosition / 60 % 60;
        hours = (int)CurrentVideo.CurrentPosition /60 / 60;
        if (hours < 10)
                lblVidCurrent.Text = "0" + hours.ToString() + ":";
```

```
        else
                lblVidCurrent.Text = hours.ToString() + ":";
        if (minutes < 10)
                lblVidCurrent.Text = lblVidCurrent.Text + "0" + minutes.ToString() +
                                        ":";
        else
                lblVidCurrent.Text = lblVidCurrent.Text + minutes.ToString() + ":";
        if (seconds < 10)
                lblVidCurrent.Text = lblVidCurrent.Text + "0" + seconds.ToString();
        else
                lblVidCurrent.Text = lblVidCurrent.Text + seconds.ToString();
        //  Assign the current video position to the Position slider.
        tbVideo.Value = (int) CurrentVideo.CurrentPosition;
        //  Determine whether the video must be looped.
        if (chkLoop.Checked == true)
        {
                //  Loop it.
                if (lblVidCurrent.Text == lblVidTotal.Text)
                {
                        CurrentVideo.CurrentPosition = 0;
                        tbVideo.Value = 0;
                }
        }
        else
                //  Don't loop it.
                if (lblVidCurrent.Text == lblVidTotal.Text)
        {
                //  Reset the controls.
                PlayPause = 2;
                cmdPlayPause.ImageIndex = 0;
                tmVideo.Enabled = false;
                CurrentVideo.CurrentPosition = 0;
                tbVideo.Value = 0;
                CurrentVideo.Stop();
                lblVidCurrent.Text = "00:00:00";
        }
}
```

1.6.37 PictureAdjustment toolbar

```
//  This subroutine is used as a switchboard to determine which
//  button was clicked on the PictureAdjustment toolbar.
private void tbPictureAdjustment_ButtonClick(object sender,
                System.Windows.Forms.ToolBarButtonClickEventArgs e)
{
        switch(tbPictureAdjustment.Buttons.IndexOf(e.Button))
```

```
        {
                case 0:  AdjustBrightness();
                         break;
                case 1:  AdjustContrast();
                         break;
        }
}
```

1.6.38 PictureProcess Button

```
//  This subroutine is called when the Picture Process button
//  is clicked.
private void cmdPictureProcess_Click(object sender, System.EventArgs e)
{
        //  Disables the toolbar buttons.
        tbPictureAdjustment.Buttons[0].Enabled = false;
        tbPictureAdjustment.Buttons[1].Enabled = false;
        //  Disables the following buttons.
        cmdPictureProcess.Enabled = false;
        gbProcessType.Enabled = false;
        tbPictureAdjustment.Enabled = false;
        //  Set up for Area Mode.
        if (optArea.Checked == true)
        {
                tbUpdateModel.Buttons[0].Enabled = false;
                tbUpdateModel.Buttons[1].Enabled = false;
                tbUpdateModel.Buttons[2].Enabled = false;
                tbUpdateModel.Buttons[3].Enabled = false;
                picStudy.Enabled = true;
                clickcounter = 0;
                lblPictureInfo.Text = "Please draw a selection rectangle around the
                                       object.";
        }
        //  Call Whole Image Mode.
        if (optWhole.Checked == true)
        {
                SingleFileWhole();
        }
}
```

1.6.39 Full Results Button

```
//  This subroutine is called when the Full Results button is clicked.
//  It shows the full results page.
private void cmdFull_Click(object sender, System.EventArgs e)
{
```

```
        tabPictures.TabPages.Remove(tabNormal);
        tabPictures.TabPages.Add(tabRoughEdges);
}
```

1.6.40 Minimal Results Button

```
//  This subroutine is called when the Minimal Results button is clicked.
//  It hides the full results page.
private void cmdMinimal_Click(object sender, System.EventArgs e)
{
        tabPictures.TabPages.Remove(tabRoughEdges);
        tabPictures.TabPages.Add(tabNormal);
}
```

1.6.41 BatchActivate Button

```
//  This subroutine is called when the Batch Activate button is clicked.
private void cmdBatchActivate_Click(object sender, System.EventArgs e)
{
        lblBatchInfo.Text = "Processing...";
        BatchFile();
        CurrentlistIndex = 0;
        lblBatchEdgesNumber.Text = "Edge Graphs: " + (CurrentlistIndex  + 1) +
                " of " + BatchGraphs.Count;
        picBatchGraphs.Image = (Bitmap)BatchGraphs[CurrentlistIndex];
        picBatchGraphs.SizeMode = PictureBoxSizeMode.StretchImage;
        cmdBatchActivate.Enabled = false;
}
```

1.6.42 Accept Button

```
//  This subroutine is called when the Accept Measurements button
//  is clicked.  It saves the current object measurements and EdgeGraph.
private void cmdAccept_Click(object sender, System.EventArgs e)
{
        UpdateModelFile(SinglePerimeter,SingleArea,SingleXDistance,SingleYDis
                        tance,SingleEdgeCount);
        //  Save the newly created EdgeGraph to disc.
        bmpCurrentEdges.Save(RootDirectory + "\\" +
                             System.DateTime.Now.Day.ToString() +
                             System.DateTime.Now.Month.ToString() +
                             System.DateTime.Now.Year.ToString() +
                             System.DateTime.Now.Minute.ToString() +
                             System.DateTime.Now.Second.ToString() +
                             ".jpg",System.Drawing.Imaging.ImageFormat.Jpeg);
        cmdAccept.Visible = false;
        lblPictureInfo.Text = "Model Updated";
```

```
}
```

1.6.43 AcceptBatch Button

```
//  This subroutine is called when the Accept Batch Measurements
//  button is clicked.  It saves the Batch Measurements and EdgeGraphs.
private void cmdAcceptBatch_Click(object sender, System.EventArgs e)
{
        //  TextWriter variable for writing to the model's .mdl-file.
        TextWriter tw;
        tw = new StreamWriter(RootDirectory + "\\" + ModelName + ".mdl");
        //  Write all of the values to the model's .mdl-file.
        tw.WriteLine(PerimeterLow);
        tw.WriteLine(PerimeterHigh);
        tw.WriteLine(AreaLow);
        tw.WriteLine(AreaHigh);
        tw.WriteLine(ShortestLow);
        tw.WriteLine(ShortestHigh);
        tw.WriteLine(LongestLow);
        tw.WriteLine(LongestHigh);
        tw.WriteLine(EdgesLow);
        tw.WriteLine(EdgesHigh);
        //  Closes the .mdl-file.
        tw.Close();
        //  Save the EdgeGraphs
        string[] files = Directory.GetFiles(RootDirectory + "\\Batch\\","*.jpg");
         foreach(string i in files)
        {
                File.Copy(i,RootDirectory + "\\" + i.Substring(RootDirectory.Length
                        + 9));
        }
        cmdAcceptBatch.Visible = false;
        lblBatchInfo.Text = "Model Updated";
}
```

1.6.44 AcceptCamVideo Button

```
//  This subroutine is called when the Accept Cam/Video Measurements
//  button is clicked.  It saves the current object's measurements and
//  EdgeGraph.
private void cmdAcceptCamVideo_Click(object sender, System.EventArgs e)
{
        UpdateModelFile(SinglePerimeter,SingleArea,SingleXDistance,
                SingleYDistance,SingleEdgeCount);
        //  Save the newly created EdgeGraph to disc.
        bmpCurrentEdges.Save(RootDirectory + "\\" +
```

```
                System.DateTime.Now.Day.ToString() +
                System.DateTime.Now.Month.ToString() +
                System.DateTime.Now.Year.ToString() +
                System.DateTime.Now.Minute.ToString() +
                 System.DateTime.Now.Second.ToString() +
                ".jpg",System.Drawing.Imaging.ImageFormat.Jpeg);
        cmdAcceptCamVideo.Visible = false;
        lblInfo.Text = "Model Updated";
}
```

1.6.45 Next Button

```
//  This subroutine is called when the 'Next' button is clicked.
//  It displays the next image in the filelist array.
private void cmdNext_Click(object sender, System.EventArgs e)
{
        //  Checks whether there are still images left in the filelist array
        if (CurrentlistIndex < (BatchGraphs.Count - 1))
        {
                //  Sets the image of the picturebox picModelImages to the next
                //  value stored in the filelist array and sizes it accordingly.
                picBatchGraphs.Image = (Bitmap)BatchGraphs[CurrentlistIndex];
                picBatchGraphs.SizeMode = PictureBoxSizeMode.StretchImage;
                CurrentlistIndex++;
                //  Sets the value of the label lblBatchEdgesNumber
                lblBatchEdgesNumber.Text = "Edge Graphs: " + (CurrentlistIndex
                        + 1) +  " of " + BatchGraphs.Count;
                //  Make sure the user will be able to go back to previous images.
                cmdPrevious.Enabled = true;
        }
        //  Checks whether the end of the filelist array has been reached.
        if (CurrentlistIndex == (BatchGraphs.Count - 1))
        {
                //  Make sure the user won't be able to go to a next image, because
                //  the last image in the array has been reached.
                cmdNext.Enabled = false;
                //  Make sure the user will be able to go back to previous images.
                if (CurrentlistIndex > 0)
                        cmdPrevious.Enabled = true;
                //  Sets the value of the label lblBatchEdgesNumber
                lblBatchEdgesNumber.Text = "Edge Graphs: " + (CurrentlistIndex
                        + 1) +  " of " + BatchGraphs.Count;
        }
}
```

1.6.46 Previous Button

```
//  This subroutine is called when the 'Previous' button is clicked.
//  It displays the previous image in the filelist array.
private void cmdPrevious_Click(object sender, System.EventArgs e)
{
        //  Checks whether there are still images left in the filelist array
        if (CurrentlistIndex > 0)
        {
                //  Sets the image of the picturebox picModelImages to the previous
                //  value stored in the filelist array and sizes it accordingly.
                CurrentlistIndex--;
                picBatchGraphs.Image = (Bitmap)BatchGraphs[CurrentlistIndex];
                picBatchGraphs.SizeMode = PictureBoxSizeMode.StretchImage;
                //  Sets the value of the label lblBatchEdgesNumber
                lblBatchEdgesNumber.Text = "Edge Graphs: " + (CurrentlistIndex
                      + 1) + " of " + BatchGraphs.Count;
                //  Make sure the user will be able to go to the next image.
                cmdNext.Enabled = true;
        }
        //  Checks whether the start of the filelist array has been reached.
        if (CurrentlistIndex == 0)
        {
                //  Make sure the user won't be able to go to a previous image,
                //  cause the first image in the array has been reached.
                cmdPrevious.Enabled = false;
                //  Make sure the user will be able to go to the next image.
                if (BatchGraphs.Count > 1)
                        cmdNext.Enabled = true;
                //  Sets the value of the label lblBatchEdgesNumber
                lblBatchEdgesNumber.Text = "Edge Graphs: " + (CurrentlistIndex +
                      1) + " of " + BatchGraphs.Count;
        }
}
```

1.6.47 Queue List Manipulators

```
//  This subroutine is called when the user clicks on an item in listbox
//   lbBatchQueue1.
private void lbBatchQueue1_SelectedIndexChanged(object sender,
             System.EventArgs e)
{
        //  Sets the filename label to the path of the currently selected file
        //  and sets the picturebox to display its image.
        picQueue1.Image = (Bitmap)
            Bitmap.FromFile(lbBatchQueue1.SelectedItem.ToString());
```

```
}

//  This subroutine is called when the user clicks on queue 1's 'Next' button.
private void cmdNextQueue1_Click(object sender, System.EventArgs e)
{
        //  Checks to see whether there are items left in the queue.
        //  If there is, increment the Queue1Current value, which
        //  acts as an index.
        if (Queue1Current < lbBatchQueue1.Items.Count - 1)
                Queue1Current++;
        //  Sets the filename label to the path of the currently selected file
        //  and sets the picturebox to display its image.
        picQueue1.Image = (Bitmap)  Bitmap.FromFile(lbBatchQueue1.Items
                                [Queue1Current].ToString());
}

//  This subroutine is called when the user clicks on queue 1's 'Previous' button.
private void cmdPrevQueue1_Click(object sender, System.EventArgs e)
{
        //  Checks to see whether there are items left in the queue.
        //  If there is, decrement the Queue1Current value, which
        //  acts as an index.
        if (Queue1Current > 0)
                Queue1Current--;
        //  Sets the filename label to the path of the currently selected file
        //  and sets the picturebox to display its image.
        picQueue1.Image = (Bitmap) Bitmap.FromFile(lbBatchQueue1.Items
                               [Queue1Current].ToString());
}

//  This subroutine is called when the user clicks on an item in listbox
//  lbBatchQueue2.
private void lbBatchQueue2_SelectedIndexChanged(object sender,
                System.EventArgs e)
{
        //  Sets the filename label to the path of the currently selected file
        //  and sets the picturebox to display its image.
        picQueue2.Image = (Bitmap)
             Bitmap.FromFile(lbBatchQueue2.SelectedItem.ToString());
}

//  This subroutine is called when the user clicks on queue 2's 'Next' button.
private void cmdNextQueue2_Click(object sender, System.EventArgs e)
{
        //  Checks to see whether there are items left in the queue.
        //  If there is, increment the Queue2Current value, which
```

```
        //  acts as an index.
        if (Queue2Current < lbBatchQueue2.Items.Count - 1)
                Queue2Current++;
        //  Sets the filename label to the path of the currently selected file
        //  and sets the picturebox to display its image.
        picQueue2.Image = (Bitmap) Bitmap.FromFile(lbBatchQueue2.Items
                                [Queue2Current].ToString());
}

//  This subroutine is called when the user clicks on queue 2's 'Previous' button.
private void cmdPrevQueue2_Click(object sender, System.EventArgs e)
{
        //  Checks to see whether there are items left in the queue.
        //  If there is, decrement the Queue2Current value, which
        //  acts as an index.
        if (Queue2Current > 0)
                Queue2Current--;
        //  Sets the filename label to the path of the currently selected file
        //  and sets the picturebox to display its image.
        picQueue2.Image = (Bitmap)
            Bitmap.FromFile(lbBatchQueue2.Items[Queue2Current].ToString());
}

//  This subroutine is called when the user clicks on an item in listbox
//  lbBatchQueue3.
private void lbBatchQueue3_SelectedIndexChanged(object sender,
                System.EventArgs e)
{
        //  Sets the filename label to the path of the currently selected file
        //  and sets the picturebox to display its image.
        picQueue3.Image = (Bitmap)
              Bitmap.FromFile(lbBatchQueue3.SelectedItem.ToString());
}

//  This subroutine is called when the user clicks on queue 3's 'Next' button.
private void cmdNextQueue3_Click(object sender, System.EventArgs e)
{
        //  Checks to see whether there are items left in the queue.
        //  If there is, increment the Queue3Current value, which
        //  acts as an index.
        if (Queue3Current < lbBatchQueue3.Items.Count - 1)
                Queue3Current++;
        //  Sets the filename label to the path of the currently selected file
        //  and sets the picturebox to display its image.
        picQueue3.Image = (Bitmap) Bitmap.FromFile(lbBatchQueue3.Items
                               [Queue3Current].ToString());}
```

```
//  This subroutine is called when the user clicks on queue 3's 'Previous' button.
private void cmdPrevQueue3_Click(object sender, System.EventArgs e)
{
        //  Checks to see whether there are items left in the queue.
        //  If there is, decrement the Queue3Current value, which
        //  acts as an index.
        if (Queue3Current > 0)
                Queue3Current--;
        //  Sets the filename label to the path of the currently selected file
        //  and sets the picturebox to display its image.
        picQueue3.Image = (Bitmap)
                Bitmap.FromFile(lbBatchQueue3.Items[Queue3Current].ToString());
}

//  This subroutine is called when the user clicks on an item in listbox
//  lbBatchQueue4.
private void lbBatchQueue4_SelectedIndexChanged(object sender,
                System.EventArgs e)
{
        //  Sets the filename label to the path of the currently selected file
        //  and sets the picturebox to display its image.
        picQueue4.Image = (Bitmap)
                Bitmap.FromFile(lbBatchQueue4.SelectedItem.ToString());
}

//  This subroutine is called when the user clicks on queue 4's 'Next' button.
private void cmdNextQueue4_Click(object sender, System.EventArgs e)
{
        //  Checks to see whether there are items left in the queue.
        //  If there is, increment the Queue4Current value, which
        //  acts as an index.
        if (Queue4Current < lbBatchQueue4.Items.Count - 1)
                Queue4Current++;
        //  Sets the filename label to the path of the currently selected file
        //  and sets the picturebox to display its image.
        picQueue4.Image = (Bitmap) Bitmap.FromFile(lbBatchQueue4.Items
                                [Queue4Current].ToString());
}

//  This subroutine is called when the user clicks on queue 4's 'Previous' button.
private void cmdPrevQueue4_Click(object sender, System.EventArgs e)
{
        //  Checks to see whether there are items left in the queue.
        //  If there is, decrement the Queue4Current value, which
        //  acts as an index.
        if (Queue4Current > 0)
```

```
            Queue4Current--;
        //  Sets the filename label to the path of the currently selected file
        //  and sets the picturebox to display its image.
        picQueue4.Image = (Bitmap)
              Bitmap.FromFile(lbBatchQueue4.Items[Queue4Current].ToString());
}
```

1.6.48 Context Menu Options

```
//  This subroutine is called when the user clicks on picCurrentEdges.
private void picCurrentEdges_MouseDown(object sender,
            System.Windows.Forms.MouseEventArgs e)
{
        //  Checks to make sure the right mouse button was clicked.
        if (e.Button == System.Windows.Forms.MouseButtons.Right)
        {
                //  It was, so save it's position.
                System.Drawing.Point CurrentMouse = new
                     System.Drawing.Point();
                CurrentMouse.X = e.X;
                CurrentMouse.Y = e.Y;
                //  Call the relative context menu.
                picCurrentEdgesMenu.Show(picCurrentEdges, CurrentMouse);
        }
}

//  This subroutine is called when the 'Open in own window' option
//  is selected in the relative context menu.
private void mnuCurrentEdges_Window_Click(object sender, System.EventArgs
                                                    e)
{
        frmPicDisplay1 PicDisplay = new
            frmPicDisplay1((Bitmap)picCurrentEdges.Image);
        PicDisplay.Show();
}

//  This subroutine is called when the 'Open in own window' option
//  is selected in the relative context menu.
private void mnuPicModelEdgesOwnWindow_Click(object sender,
            System.EventArgs e)
{
        frmPicDisplay1 PicDisplay = new
            frmPicDisplay1((Bitmap)picBatchGraphs.Image);
        PicDisplay.Show();
}
```

```
//  This subroutine is called when the 'Save As...' option
//  is selected in the relative context menu.
private void mnuBoth_SaveAs_Click(object sender, System.EventArgs e)
{
        //  Sets the save file dialog's filter and properties.
        sfdBoth.Filter = "JPEG Files (*.jpg)|*.jpg";
        sfdBoth.FileName = "";
        //  Opens the save file dialog and saves the file.
        if(sfdBoth.ShowDialog() == DialogResult.OK)
        {
                bmpBoth.Save(sfdBoth.FileName,
                    System.Drawing.Imaging.ImageFormat.Jpeg);
        }
}

//  This subroutine is called when the 'Open in own window' option
//  is selected in the relative context menu.
private void mnuBoth_Window_Click(object sender, System.EventArgs e)
{
        frmPicDisplay1 PicDisplay = new frmPicDisplay1(bmpBoth);
        PicDisplay.Show();
}

//  This subroutine is called when the user clicks on picRoughEdges.
private void picRoughEdges_MouseDown(object sender,
                System.Windows.Forms.MouseEventArgs e)
{
        //  Checks to make sure the right mouse button was clicked.
        if (e.Button == System.Windows.Forms.MouseButtons.Right)
        {
                //  It was, so save it's position.
                System.Drawing.Point CurrentMouse = new
                System.Drawing.Point();
                CurrentMouse.X = e.X;
                CurrentMouse.Y = e.Y;
                //  Call the relative context menu.
                picRoughEdgesMenu.Show(picRoughEdges, CurrentMouse);
        }
}

//  This subroutine is called when the 'Save As...' option
//  is selected in the relative context menu.
private void mnuRoughEdges_SaveAs_Click(object sender, System.EventArgs
                                                        e)
{
        //  Sets the save file dialog's filter and properties.
```

```
        sfdRoughEdges.Filter = "JPEG Files (*.jpg)|*.jpg";
        sfdRoughEdges.FileName = "";
        //  Opens the save file dialog and saves the file.
        if(sfdRoughEdges.ShowDialog() == DialogResult.OK)
        {
            picRoughOutlineBitmap.Save
                (sfdRoughEdges.FileName,
                 System.Drawing.Imaging.ImageFormat.Jpeg);
        }
}

//  This subroutine is called when the 'Open in own window' option
//  is selected in the relative context menu.
private void mnuRoughEdges_Window_Click(object sender, System.EventArgs
                                                    e)
{
        frmPicDisplay1 PicDisplay = new frmPicDisplay1(picRoughOutlineBitmap);
        PicDisplay.Show();
}

//  This subroutine is called when the user clicks on picOriginalAndOutline.
private void picOriginalAndOutline_MouseDown(object sender,
            System.Windows.Forms.MouseEventArgs e)
{
        //  Checks to make sure the right mouse button was clicked.
        if (e.Button == System.Windows.Forms.MouseButtons.Right)
        {
            //  It was, so save it's position.
            System.Drawing.Point CurrentMouse = new
                System.Drawing.Point();
            CurrentMouse.X = e.X;
            CurrentMouse.Y = e.Y;
            //  Call the relative context menu.
            OriginalAndOutlineMenu.Show(picOriginalAndOutline,
                 CurrentMouse);
        }
}

//  This subroutine is called when the 'Save As...' option
//  is selected in the relative context menu.
private void mnuOriginalAndOutline_SaveAs_Click(object sender,
            System.EventArgs e)
{
        //  Sets the save file dialog's filter and properties.
        sfdOriginalAndOutline.Filter = "JPEG Files (*.jpg)|*.jpg";
        sfdOriginalAndOutline.FileName = "";
```

```
        //  Opens the save file dialog and saves the file.
        if(sfdOriginalAndOutline.ShowDialog() == DialogResult.OK)
        {
            bmpOriginalAndOutline.Save
            (sfdOriginalAndOutline.FileName,
            System.Drawing.Imaging.ImageFormat.Jpeg);
        }
}

//  This subroutine is called when the 'Open in own window' option
//  is selected in the relative context menu.
private void mnuOriginalAndOutline_Window_Click(object sender,
              System.EventArgs e)
{
        frmPicDisplay1 PicDisplay = new frmPicDisplay1(bmpOriginalAndOutline);
        PicDisplay.Show();
}

//  This subroutine is called when the 'Save As...' option
//  is selected in the relative context menu.
private void mnuOriginalAndBoth_SaveAs_Click(object sender,
              System.EventArgs e)
{
        //  Sets the save file dialog's filter and properties.
        sfdOriginalAndBoth.Filter = "JPEG Files (*.jpg)|*.jpg";
        sfdOriginalAndBoth.FileName = "";
        //  Opens the save file dialog and saves the file.
        if(sfdOriginalAndBoth.ShowDialog() == DialogResult.OK)
        {
            bmpOriginalAndBoth.Save
            (sfdOriginalAndBoth.FileName,
              System.Drawing.Imaging.ImageFormat.Jpeg);
        }
}

//  This subroutine is called when the 'Open in own window' option
//  is selected in the relative context menu.
private void mnuOriginalAndBoth_Window_Click(object sender,
              System.EventArgs e)
{
        frmPicDisplay1 PicDisplay = new frmPicDisplay1(bmpOriginalAndBoth);
        PicDisplay.Show();
}
```

```
//  This subroutine is called when the user clicks on picOutline.
private void picOutline_MouseDown(object sender,
                System.Windows.Forms.MouseEventArgs e)
{
        //  Checks to make sure the right mouse button was clicked.
        if (e.Button == System.Windows.Forms.MouseButtons.Right)
        {
                //  It was, so save it's position.
                System.Drawing.Point CurrentMouse = new
                    System.Drawing.Point();
                CurrentMouse.X = e.X;
                CurrentMouse.Y = e.Y;
                //  Call the relative context menu.
                OutlineMenu.Show(picOutline, CurrentMouse);
        }
}

//  This subroutine is called when the 'Save As...' option
//  is selected in the relative context menu.
private void mnuOutline_SaveAs_Click(object sender, System.EventArgs e)
{
        //  Sets the save file dialog's filter and properties.
        sfdOutline.Filter = "JPEG Files (*.jpg)|*.jpg";
        sfdOutline.FileName = "";
        //  Opens the save file dialog and saves the file.
        if(sfdOutline.ShowDialog() == DialogResult.OK)
        {
                bmpOutline.Save(sfdOutline.FileName,
                   System.Drawing.Imaging.ImageFormat.Jpeg);
        }
}

//  This subroutine is called when the 'Open in own window' option
//  is selected in the relative context menu.
private void mnuOutline_Window_Click(object sender, System.EventArgs e)
{
        frmPicDisplay1 PicDisplay = new frmPicDisplay1(bmpOutline);
        PicDisplay.Show();
}

//  This subroutine is called when the user clicks on picOriginalAndEdges.
private void picOriginalAndEdges_MouseDown(object sender,
                System.Windows.Forms.MouseEventArgs e)
{
        //  Checks to make sure the right mouse button was clicked.
        if (e.Button == System.Windows.Forms.MouseButtons.Right)
```

```
        {
                //  It was, so save it's position.
                System.Drawing.Point CurrentMouse = new
                    System.Drawing.Point();
                CurrentMouse.X = e.X;
                CurrentMouse.Y = e.Y;
                //  Call the relative context menu.
                OriginalAndEdgesMenu.Show(picOriginalAndEdges,
                    CurrentMouse);
        }
}

//  This subroutine is called when the 'Save As...' option
//  is selected in the relative context menu.
private void mnuOriginalAndEdges_SaveAs_Click(object sender,
                System.EventArgs e)
{
        //  Sets the save file dialog's filter and properties.
        sfdOriginalAndEdges.Filter = "JPEG Files (*.jpg)|*.jpg";
        sfdOriginalAndEdges.FileName = "";
        //  Opens the save file dialog and saves the file.
        if(sfdOriginalAndEdges.ShowDialog() == DialogResult.OK)
        {
                bmpOriginalAndEdges.Save
                    (sfdOriginalAndEdges.FileName,
                     System.Drawing.Imaging.ImageFormat.Jpeg);
        }
}

//  This subroutine is called when the 'Open in own window' option
//  is selected in the relative context menu.
private void mnuOrginalAndEdges_Window_Click(object sender,
System.EventArgs e)
{
        frmPicDisplay1 PicDisplay = new frmPicDisplay1(bmpOriginalAndEdges);
        PicDisplay.Show();
}

//  This subroutine is called when the Show in Own Window option
//  is selected in any picture's context menu.
private void miPictureShow_Click(object sender, System.EventArgs e)
{
        if (picPicturePreview.Image != null)
        {
                frmPicDisplay1 PicDisplay = new
                    frmPicDisplay1((Bitmap)picPicturePreview.Image);
```

```
                PicDisplay.Show();
            }
            else
                lblPictureInfo.Text = "No image to display";
    }


    }
    }
```

1.7 View a Model's Measurements (frmViewModel)

This form is called whenever the user wants to view an existing Model's measurements. The form displays all of the measurements of the Model as well as all of the Edge Graphs which were created during the process of updating the model

1.7.1 System References

```
//  Declaration of References
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.IO;
```

1.7.2 Form Items and Global Variables

```
namespace AntsNest
{
/// <summary>
/// Summary description for frmViewModel.
/// </summary>
public class frmViewModel : System.Windows.Forms.Form
{
        private System.Windows.Forms.Button cmdOK;
        private System.Windows.Forms.GroupBox groupBox1;
        private System.Windows.Forms.Label lblModelName;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.Label lblModelYDistance;
        private System.Windows.Forms.Label lblModelXDistance;
        private System.Windows.Forms.Label lblModelArea;
        private System.Windows.Forms.Label lblModelPerimeter;
        private System.Windows.Forms.Label label17;
        private System.Windows.Forms.Label label18;
        private System.Windows.Forms.Label label20;
        private System.Windows.Forms.Label label22;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.Button cmdPrevious;
        private System.Windows.Forms.Button cmdNext;
        private System.Windows.Forms.PictureBox picModelImages;
        private System.Windows.Forms.Panel panel1;
```

```
/// <summary>
/// Required designer variable.
/// </summary>
        private System.ComponentModel.Container components = null;
        // String array used to store the list of graph files in the
        // specific model's folder.
        private string[] filelist;
        //  String variable used to pass the location of the Models
        //  directory to the rest of the program
        private string RootDirectory;
        //  Integer variable used to keep track of the graph
        //  currently displayed in picturebox picModelImages.
        private int filelistIndex;
```

1.7.3 Form Initialization

```
//  Startup subroutine for frmViewModel.
//  Takes one argument, consisting of a string variable,
//  defining which model's statistics to be display.
public frmViewModel(string ModelName)
{
        InitializeComponent();
        //  Sets the value of RootDirectory to the path of the Models
        //  directory in the directory where the executable file of
        //  the application resides.
        RootDirectory = Application.StartupPath + "\\Models\\" + ModelName;
        //  Create a new TextReader object, to be used for reading
        //  the values stored in the model's .mdl-file.
        TextReader tr;
        //  String variable used to temporarily store the value
        //  read off the .mdl-file.
        string compare = "";
        //  Disables the 'Next' and 'Previous' buttons.  In cae
        //  there is only one graph.
        cmdNext.Enabled = false;
        cmdPrevious.Enabled = false;
        //  Assigns the list of graph files to the filelist string array
        filelist = Directory.GetFiles(RootDirectory,"*.jpg");
        //  Sets the filelistIndex to 0, indicating the first item in the
        //  filelist array.
        filelistIndex = 0;
        //  Sets the image of the picturebox picModelImages to the first
        //  value stored in the filelist array and sizes it accordingly.
        picModelImages.Image = (Bitmap)Bitmap.FromFile(filelist[filelistIndex]);
        picModelImages.SizeMode = PictureBoxSizeMode.StretchImage;
```

```
//  If there is more than one graph, enable the next button.
if (filelist.Length > 1)
{
        cmdNext.Enabled = true;
}

//  Initializes the temporary storage variables.
PerimeterLow = PerimeterHigh = AreaLow = AreaHigh = 0;
ShortestLow = ShortestHigh = LongestLow = LongestHigh = 0;

//  Checks whether the specific model's Directory exists
if (Directory.Exists(RootDirectory))
{
        //  The model's directory exists.

        //  Checks whether the specific model's .mdl-file exists.
        if (File.Exists(RootDirectory + "\\" + ModelName + ".mdl"))
        {
        //  The .mdl-file exists.

        //  Read the .mdl file and store the values in the
        //  temporary variables.
                tr = new StreamReader(RootDirectory + "\\" + ModelName
                                        + ".mdl");

                for (int i = 0; i < 8; i++)
                {
                        compare = tr.ReadLine();
                        if (i == 0)
                                if (compare == null)
                                        PerimeterLow = 0;
                                else
                                        PerimeterLow = Int32.Parse(compare);
                        if (i == 1)
                                if (compare == null)
                                        PerimeterHigh = 0;
                                else
                                        PerimeterHigh = Int32.Parse(compare);
                        if (i == 2)
                                if (compare == null)
                                        AreaLow = 0;
                                else
                                        AreaLow = Int32.Parse(compare);
                        if (i == 3)
                                if (compare == null)
                                        AreaHigh = 0;
```

```
                        else
                                AreaHigh = Int32.Parse(compare);
                    if (i == 4)
                        if (compare == null)
                                ShortestLow = 0;
                        else
                                ShortestLow = Int32.Parse(compare);
                    if (i == 5)
                        if (compare == null)
                                ShortestHigh = 0;
                        else
                                ShortestHigh = Int32.Parse(compare);
                    if (i == 6)
                        if (compare == null)
                                LongestLow = 0;
                        else
                                LongestLow = Int32.Parse(compare);
                    if (i == 7)
                        if (compare == null)
                                LongestHigh = 0;
                        else
                                LongestHigh = Int32.Parse(compare);
                }
                // Closes the .mdl-file.
                tr.Close();
        }
        else
        {
                //  The .mdl file does not exist, so it is created.
                File.Create(RootDirectory + "\\" + ModelName + ".mdl");
        }
}
else
{
        //  The directory doesn't exist, so it and the .mdl-file
        //  is created.
        Directory.CreateDirectory(RootDirectory);
        File.Create(RootDirectory + "\\" + ModelName + ".mdl");
}

 //  Assign the values to the display labels.
lblModelName.Text = ModelName;
lblModelPerimeter.Text = PerimeterLow.ToString() + "-" +
                         PerimeterHigh.ToString();
lblModelArea.Text = AreaLow.ToString() + "-" +
                    AreaHigh.ToString();
```

```
        lblModelXDistance.Text = ShortestLow.ToString() + "-" +
                                 ShortestHigh.ToString();
        lblModelYDistance.Text = LongestLow.ToString() + "-" +
                                 LongestHigh.ToString();
        //  Sets the value of the label lblModelEdgesNumber
        lblModelEdgesNumber.Text = "Model Edge Graphs: " + (filelistIndex  + 1)
                                   +  " of " + filelist.Length;
}
```

1.7.4 Form Designer Generated Code

```
/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
        if( disposing )
        {
                if(components != null)
                {
                        components.Dispose();
                }
        }
        base.Dispose( disposing );
}

#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
        //  Component code omitted.
}
#endregion
```

1.7.5 Next Button

```
//  This subroutine is called when the 'Next' button is clicked.
//  It displays the next image in the filelist array.
private void cmdNext_Click(object sender, System.EventArgs e)
{
        //  Checks whether there are still images left in the filelist array
        if (filelistIndex < (filelist.Length))
        {
```

```
                // Sets the image of the picturebox picModelImages to the
                // next value stored in the filelist array and sizes it
                // accordingly.
                picModelImages.Image =
                        (Bitmap)Bitmap.FromFile(filelist[filelistIndex]);
                picModelImages.SizeMode =
                        PictureBoxSizeMode.StretchImage;
                filelistIndex++;
                // Sets the value of the label lblModelEdgesNumber
                lblModelEdgesNumber.Text = "Model Edge Graphs: " +
                                (filelistIndex  + 1) + " of " + filelist.Length;
                // Make sure the user will be able to go back to previous
                // images.
                cmdPrevious.Enabled = true;
        }

        // Checks whether the end of the filelist array has been reached.
        if (filelistIndex == (filelist.Length))
        {
                // Make sure the user won't be able to go to a next image, because
                // the last image in the array has been reached.
                cmdNext.Enabled = false;
                // Make sure the user will be able to go back to previous images.
                if (filelistIndex > 0)
                        cmdPrevious.Enabled = true;
                                // Sets the value of the label lblModelEdgesNumber
                lblModelEdgesNumber.Text = "Model Edge Graphs: " +
                                (filelistIndex  + 1) + " of " + filelist.Length;
        }
}
```

1.7.6 Previous Button

```
// This subroutine is called when the 'Previous' button is clicked.
// It displays the previous image in the filelist array.
private void cmdPrevious_Click(object sender, System.EventArgs e)
{
        // Checks whether there are still images left in the filelist array
        if (filelistIndex > 0)
        {
                // Sets the image of the picturebox picModelImages to the previous
                // value stored in the filelist array and sizes it accordingly.
                filelistIndex--;
                picModelImages.Image =
                                (Bitmap)Bitmap.FromFile(filelist[filelistIndex]);
                picModelImages.SizeMode = PictureBoxSizeMode.StretchImage;
```

```
                // Sets the value of the label lblModelEdgesNumber
                lblModelEdgesNumber.Text = "Model Edge Graphs: " +
                                (filelistIndex  + 1) + " of " + filelist.Length;
                // Make sure the user will be able to go to the next image.
                cmdNext.Enabled = true;
        }
        // Checks whether the start of the filelist array has been reached.
        if (filelistIndex == 0)
        {
                // Make sure the user won't be able to go to a previous image,
                // because the first image in the array has been reached.
                cmdPrevious.Enabled = false;
                // Make sure the user will be able to go to the next image.
                if (filelist.Length > 1)
                        cmdNext.Enabled = true;
                // Sets the value of the label lblModelEdgesNumber
                lblModelEdgesNumber.Text = "Model Edge Graphs: " +
                                (filelistIndex  + 1) + " of " + filelist.Length;

        }
}
```

1.7.7 OK Button

```
// This subroutine is called when the 'OK' button is clicked.
// It closes frmViewModel.
private void cmdOK_Click(object sender, System.EventArgs e)
{
        this.Close();
}


}
}
```

1.8 Set Conveyer Dimensions (frmDimensions)

This form is called when the user needs to set the dimensions of the conveyer belt that is being used. This is crucial if the recognition phase is to return accurate real-world coordinates.

1.8.1 System References

```
//  Declaration of Referemces
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.IO;
using System.Drawing.Imaging;
using System.Runtime.InteropServices;
using System.Threading;
using Microsoft.DirectX.AudioVideoPlayback;
using WIALib;
using WIAVIDEOLib;
```

1.8.2 Form Items and Global Variables

```
namespace AntsNest
{
/// <summary>
/// Summary description for frmDimensions.
/// </summary>
public class frmDimensions : System.Windows.Forms.Form
{
        private System.Windows.Forms.Panel pnlCam;
        private System.Windows.Forms.Button cmdUpdate;
        private System.Windows.Forms.TextBox txtYDistance;
        private System.Windows.Forms.TextBox txtXDistance;
        private System.Windows.Forms.Label lblCurrentX;
        private System.Windows.Forms.Label lblCurrentY;
        private System.Windows.Forms.Label lblInfo;
        private System.Windows.Forms.GroupBox groupBox1;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.GroupBox groupBox2;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.Label label4;
```

```
        private System.Windows.Forms.GroupBox groupBox3;
        private System.Windows.Forms.RadioButton optUpwards;
        private System.Windows.Forms.RadioButton optDownwards;
        private System.Windows.Forms.RadioButton optLeft;
        private System.Windows.Forms.RadioButton optRight;
        private System.Windows.Forms.GroupBox groupBox4;
        private System.Windows.Forms.TextBox txtSpeed;
        private System.Windows.Forms.Label label5;
        private System.Windows.Forms.Label label6;
        private System.Windows.Forms.Label lblCurrentSpeed;
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.Container components = null;
        //  Windows Image Aquisition Device variable used
        //  to store the currently loaded Capture Device.
        private WIA.Device CurrentCam;
        //  WIA manager COM object used to store the current system WIA
        //  information.
        private WiaClass            wiaManager;
        // WIA ItemClass object used to store the currently selected capture
        //  device.
        private ItemClass           wiaCamera;
        // WIAVideoClass object used to store the current video stream.
        private WiaVideoClass       wiaVideo;
```

1.8.3 Form Initialization

```
//  Startup subroutine for frmDimensions.
public frmDimensions()
{
        //  Initialize form items.
        InitializeComponent();
        //  Make sure no camera is currently loaded.
        CurrentCam = null;
        //  Load a capture device
        CameraMode();
        //  Initializes the variable used to temporarily
        //  store values read off the Dimensions.cfg file.
        string temp = "";
        //  Sets labels to "0" in case no values are stored in
        //  the config file or the file does not exist.
        lblCurrentX.Text = "0";
        lblCurrentY.Text = "0";
        lblCurrentSpeed.Text = "0";
         //  Read the values off the config file and place them
```

```
//  in the relevant textboxes.
try
{
        if (File.Exists(Application.StartupPath + "\\Dimensions.cfg"))
        {
                //  The file exists, so read its values.
                lblInfo.Text = "File Found";
                StreamReader sr = new
                        StreamReader(Application.StartupPath +
                        "\\Dimensions.cfg");
                temp = sr.ReadLine();
                if (temp != null)
                        lblCurrentX.Text = temp;
                else
                        lblCurrentX.Text = "0";
                temp = sr.ReadLine();
                if (temp != null)
                        lblCurrentY.Text = temp;
                else
                        lblCurrentY.Text = temp;
                temp = sr.ReadLine();
                if (temp != null)
                {
                        double CurrentSpeed = Double.Parse(temp) / 1000;
                        lblCurrentSpeed.Text = CurrentSpeed.ToString();
                }
                else
                        lblCurrentSpeed.Text = "0";
                temp = sr.ReadLine();
                if (temp == "0")
                        optUpwards.Checked = true;
                else
                if (temp == "1")
                        optDownwards.Checked = true;
                else
                if (temp == "2")
                        optLeft.Checked = true;
                else
                if (temp == "3")
                        optRight.Checked = true;
                else
                        optUpwards.Checked = true;
                sr.Close();
        }
        else
        {
```

```
                    //  No file exists, so set the values to "0" and "Upwards"
                    //  The file is not actually created here.  The file is only
                     //  created when the Update button is clicked.
                    lblInfo.Text = "File Created";
                    lblCurrentX.Text = "0";
                    lblCurrentY.Text = "0";
                    lblCurrentSpeed.Text = "0";
                    optUpwards.Checked = true;
                }
        }
        catch (Exception exception)
        {
                //  In case there is an error in the Dimensions.cfg file.
                lblInfo.Text = "Error in Dimensions file.  Setting values to defaults.";
                lblCurrentX.Text = "0";
                lblCurrentY.Text = "0";
                lblCurrentSpeed.Text = "0";
                optUpwards.Checked = true;
        }
}
```

1.8.4 Form Designer Generated Code

```
/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
        if( disposing )
        {
                //      DisposePicure();
                if( wiaVideo != null )
                        // release any COM instances
                        Marshal.ReleaseComObject( wiaVideo ); wiaVideo = null;
                if( wiaCamera != null )
                        Marshal.ReleaseComObject( wiaCamera );
                 wiaCamera = null;
                if( wiaManager != null )
                        Marshal.ReleaseComObject( wiaManager );
                wiaManager = null;
                if (components != null)
                {
                        components.Dispose();
                }
        }
        base.Dispose( disposing );
```

```
}

#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
        //  Component code omitted.
}
#endregion
```

1.8.5 Select Camera

```
//  This subroutine is called whenever a capture device needs to be
//  selected.
private void CameraMode()
{
         //  Tries to load a capture device.  If an error is found, an appropriate
        //  message is shown to the user.
        try
        {
                //  Create a new instance of a WiaClass object.
                wiaManager = new WiaClass();
                //  Create a new Select Device Dialog Box.
                WIA.CommonDialogClass class1 = new
                      WIA.CommonDialogClass();
                //  If any other device is loaded, destroy its reference.
                if (CurrentCam != null)
                        wiaVideo.DestroyVideo();
                //  Show the Select Device Dialog Box and assigns the selected
                //  device to CurrentCam.
                CurrentCam = class1.ShowSelectDevice
                                (WIA.WiaDeviceType.UnspecifiedDeviceType,
                                 true, false);
                //  If a device was selected.
                if (CurrentCam != null)
                {
                        //  Assigns the selected device's name to the DeviceID
                        //  string.
                        string DeviceID = CurrentCam.DeviceID.ToString();
                        //  Initialize variables
                        object foundID = null;
                        CollectionClass wiaDevs = null;
                        DeviceInfoClass devInfo = null;
```

```
int Found = 0;
//  Assigns all of the current WIA devices in the system to
//  wiaDevs.
wiaDevs = wiaManager.Devices as CollectionClass;
//  If there were devices
if( wiaDevs != null )
{
        //  Run thrugh all of the devices.
        foreach( object wiaObj in wiaDevs )
        {
                //  Stop the loop if the selected device was
                //  found.
                if (Found == 1)
                        break;
                //  Wrap the wiaObj in a DeviceInfoClass
                //  wrapper.
                devInfo = (DeviceInfoClass)
                        Marshal.CreateWrapperOfType
                        (wiaObj, typeof(DeviceInfoClass) );
                //  Test whether the current wiaObj matches
                //  the selected device.
                if (devInfo.Id.ToString() == DeviceID)
                {
                        //  Used to stop the loop.
                        Found = 1;
                        //  Assigns the object's ID to foundID
                        //  and releases any resources used by
                        //  the wiaObj and devInfo.
                        foundID = devInfo.Id;
                        Marshal.ReleaseComObject( wiaObj );
                        Marshal.ReleaseComObject( devInfo );
                }
        }
}

// Reinitialize variables
devInfo = null;
foundID = System.Reflection.Missing.Value;
//  Creates a video capture object using the selected device.
wiaCamera = (ItemClass) wiaManager.Create( ref foundID );
//  May sometimes take a while, so inform the user that the
//  system  is loading.
Cursor.Current = Cursors.WaitCursor;
//  Try to load the video stream, if it fails a relevant message
//  is shown to the user.
try
```

```
{
        //  Create a new instance of a WiaVideoClass object.
        wiaVideo = new WiaVideoClass();
        //  MSDN specifies that the ImagesDirectory should
        //  be set  to the WIA_DPV_IMAGES_DIRECTORY.
        //  This is specified as 3587.
        string videoDir = (string) wiaCamera.GetPropById(
                        (WiaItemPropertyId) 3587 );
        wiaVideo.ImagesDirectory = videoDir;
        //  Assigns the ID of the selected device to the
        //  cameraID string.
        string cameraID = wiaCamera.GetPropById(
                (WiaItemPropertyId) WiaDeviceInfoPropertyId.
                DeviceInfoDevId ) as string;
        //  Accesses system pointers (handles) so the code
        //  needs to be declared unsafe in order
        //  for the compiler to process it.
        unsafe
        {
                //  In order to create the video display,
                //  wiaVideo needs the handle of the
                //  componenent on which the video needs to
                //  be displayed.  This reference needs to be in
                //  the format of a
                //  WIAVideoLib._RemotableHandle,
                //  so the the Handle of pnlCam needs to be
                //  cast to this format.
                WIAVIDEOLib._RemotableHandle *handle =
                        (WIAVIDEOLib._RemotableHandle*)
                        pnlCam.Handle.ToPointer();
                wiaVideo.CreateVideoByWiaDevID(cameraID,
                        ref *handle, 1, 0);
        }
}
catch( Exception )
{
        //  If a wiaVideo object was loaded, release it.
        if( wiaVideo != null )
                Marshal.ReleaseComObject( wiaVideo );
        //  Reinitialize the variable.
        wiaVideo = null;
        //  Display an appropriate message to the user.
        MessageBox.Show( this, "Create video stream failed",
                        "WIA", MessageBoxButtons.OK,
                        MessageBoxIcon.Stop );
```

```
                }
                finally
                {
                        //  If everything was successful, do the following.
                        try
                        {
                                //  Start the video stream and display a
                                //  message to the user.
                                wiaVideo.Play();
                                lblInfo.Text = "Camera Loaded";
                        }
                        catch( Exception ) {}
                }
        }
    }
    catch( Exception )
    {
        //  Display an appropriate message to the user.
        MessageBox.Show( this, "Camera connection failed.", "Message",
                         MessageBoxButtons.OK, MessageBoxIcon.Stop );
    }
}
```

1.8.6 Update Button

```
//  This subroutine is called whenever the Update Button is clicked.
//  It saves the values in the textbox to the Dimensions.cfg file.
private void cmdUpdate_Click(object sender, System.EventArgs e)
{
        //  Variables used in determining where if any
        //  .'s or ,'s were typed in the Speed textbox.
        int Check, currentchar;
         char[] current;
        //  Write the variable to the Dimensions.cfg file.
        StreamWriter sw = new StreamWriter(Application.StartupPath +
                          "\\Dimensions.cfg");
        //  If there is text in the XDistance textbox, write it to file,
        //  else write the text of the XDistance label.
        if (txtXDistance.Text != "")
        {
                sw.WriteLine(txtXDistance.Text);
                lblCurrentX.Text = txtXDistance.Text;
                lblInfo.Text = "Dimensions updated";
        }
        else
                sw.WriteLine(lblCurrentX.Text);
```

```
//  If there is text in the YDistance textbox, write it to file,
//  else write the text of the YDistance label.
if (txtYDistance.Text != "")
{
        sw.WriteLine(txtYDistance.Text);
        lblCurrentY.Text = txtYDistance.Text;
        lblInfo.Text = "Dimensions updated";
}
else
        sw.WriteLine(lblCurrentY.Text);
        //  Initialize the variables
Check = 0;
currentchar = 0;
//  Run through the string in the Speed textbox to find any .'s or ,'s.
for (int i = 0; (i < txtSpeed.Text.Length) && (Check == 0); i++)
{
        current = txtSpeed.Text.Substring(i,1).ToCharArray();
        if (!char.IsNumber(current[0]))
        {
                if (current[0] == '.')
                {
                        Check = 1;
                        currentchar = i;
                }
                else
                {
                        if (current[0] == ',')
                        {
                                Check = 1;
                                currentchar = i;
                        }
                        else
                        {
                                MessageBox.Show("An illegal character was
                                        typed in the Speed Box.",
                                        "Message", MessageBoxButtons.OK,
                                        MessageBoxIcon.Information);
                                txtSpeed.Text = "";
                                sw.WriteLine(lblCurrentSpeed.Text);
                        }
                }
        }
}

//  If there is text in the Speed textbox, write it to file,
//  else write the text of the Speed label.
```

```
        if (txtSpeed.Text != "")
        {
                if (Check == 1)
                {
                        int temp;
                        //  Make sure the Speed value starts off with a digit.
                        try
                        {
                                temp = Int32.Parse(txtSpeed.Text.Substring
                                        (0,currentchar)) * 1000;
                        }
                        catch (Exception exception)
                        {
                                temp = 0;
                        }
                        int speed = (int)( temp +
                                        ((Int32.Parse(txtSpeed.Text.Substring
                                        (currentchar + 1))) / Math.Pow
                                        (10, txtSpeed.Text.Substring
                                        (currentchar + 1).Length) * 1000));
                        sw.WriteLine(speed);
                        lblCurrentSpeed.Text = txtSpeed.Text;
                }
                else
                {
                        sw.WriteLine(Int32.Parse(txtSpeed.Text) * 1000);
                        lblCurrentSpeed.Text = txtSpeed.Text;
                }
        }
         //  Write a value to the Dimensions.cfg file depending on
        //  which Direction option was selected.
        if (optUpwards.Checked == true)
                sw.WriteLine(0);
        if (optDownwards.Checked == true)
                sw.WriteLine(1);
        if (optLeft.Checked == true)
                sw.WriteLine(2);
        if (optRight.Checked == true)
                sw.WriteLine(3);
        //  Clears the textboxes and closes the Dimensions.cfg file.
        txtXDistance.Text = "";
        txtYDistance.Text = "";
        txtSpeed.Text = "";
        sw.Close();
}
```

1.8.7 X-Axis text changed

```
//  This subroutine is called whenever the text changes in the XDistance textbox.
//  It is used to make sure that only numeric text is entered.
private void txtXDistance_TextChanged(object sender, System.EventArgs e)
{
        if (txtXDistance.Text != "")
        {
                char[] temp = txtXDistance.Text.ToCharArray
                                (0,txtXDistance.Text.Length);
                if (char.IsNumber(temp[txtXDistance.Text.Length - 1]))
                        lblInfo.Text = "Updating X-Axis measurement";
                else
                        txtXDistance.Text = txtXDistance.Text.Substring
                                        (0,txtXDistance.Text.Length - 1);
        }
}
```

1.8.8 Y-Axis text changed

```
//  This subroutine is called whenever the text changes in the YDistance textbox.
//  It is used to make sure that only numeric text is entered.
private void txtYDistance_TextChanged(object sender, System.EventArgs e)
{
        if (txtYDistance.Text != "")
        {
                char[] temp = txtYDistance.Text.ToCharArray
                                (0,txtYDistance.Text.Length);
                if (char.IsNumber(temp[txtYDistance.Text.Length - 1]))
                        lblInfo.Text = "Updating Y-Axis measurement";
                else
                        txtYDistance.Text = txtYDistance.Text.Substring
                                        (0,txtYDistance.Text.Length - 1);
        }
}

}
}
```

1.9 Perform Recognition (frmRecognition)

This form is called whenever the user wants to perform object recognition. This is done by selecting 1 or more of the Models in the system and then providing still images, a video file or a device stream for the program to perform recognition on. Recognition is performed by comparing the newly generated object values to those of the chosen Models.

1.9.1 System References

```
//  Declaration of References
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.IO;
using System.Drawing.Imaging;
using System.Runtime.InteropServices;
using System.Threading;
using WIALib;
using WIAVIDEOLib;
using Microsoft.DirectX.AudioVideoPlayback;
```

1.9.2 Form Items and Global Variables

```
namespace AntsNest
{
/// <summary>
/// Summary description for frmStillImage.
/// </summary>
public class frmRecognition : System.Windows.Forms.Form
{
        private System.Windows.Forms.ToolBar tbStillImage;
        private System.ComponentModel.IContainer components;
        private System.Windows.Forms.OpenFileDialog ofdPicture;
        private System.Windows.Forms.ImageList ilButtons;
        private System.Windows.Forms.ToolBarButton tbbVideo;
        private System.Windows.Forms.ToolBarButton tbbCamera;
        private System.Windows.Forms.ToolBarButton tbbImage;
        private System.Windows.Forms.SaveFileDialog sfdPicture;
        private Microsoft.VisualBasic.Compatibility.VB6.DirListBox dirListBox1;
        private System.Windows.Forms.TabControl tabRecognition;
        private System.Windows.Forms.PictureBox picCapturePreview;
```

```
private System.Windows.Forms.Button cmdCamPlayPause;
private System.Windows.Forms.CheckBox chkLoop;
private System.Windows.Forms.CheckBox chkMute;
private System.Windows.Forms.GroupBox gbMatches;
private System.Windows.Forms.Label lblYDistance;
private System.Windows.Forms.Label lblXDistance;
private System.Windows.Forms.Label lblArea;
private System.Windows.Forms.Label lblPerimeter;
private System.Windows.Forms.Panel panel3;
private System.Windows.Forms.Label label9;
private System.Windows.Forms.Label label8;
private System.Windows.Forms.Label lblTime;
private System.Windows.Forms.Label lblLabelTime;
private System.Windows.Forms.Panel panel1;
private System.Windows.Forms.PictureBox picMatchY;
private System.Windows.Forms.PictureBox picMatchX;
private System.Windows.Forms.PictureBox picMatchArea;
private System.Windows.Forms.PictureBox picMatchPerimeter;
private System.Windows.Forms.Label label1;
private System.Windows.Forms.Label label2;
private System.Windows.Forms.Label label3;
private System.Windows.Forms.Label label4;
private System.Windows.Forms.Button cmdActivate;
private System.Windows.Forms.ToolBar tbRecognitionPicAdjustments;
private System.Windows.Forms.ToolBarButton tbbPictureBrightness;
private System.Windows.Forms.ToolBarButton tbbPictureContrast;
private System.Windows.Forms.GroupBox gbCamVideoControls;
private System.Windows.Forms.Button cmdStop;
private System.Windows.Forms.Button cmdPlayPause;
private System.Windows.Forms.TrackBar tbVidVolume;
private System.Windows.Forms.GroupBox gbVidTimes;
private System.Windows.Forms.Label lblCurrentVidTimeLabel;
private System.Windows.Forms.Label lblVidCurrent;
private System.Windows.Forms.Label lblTotalVidTimeLabel;
private System.Windows.Forms.Label lblVidTotal;
private System.Windows.Forms.TrackBar tbVideo;
private System.Windows.Forms.Label lblInfo;
private System.Windows.Forms.TabPage tabPicture;
private System.Windows.Forms.TabPage tabCamVideo;
private System.Windows.Forms.PictureBox picCam;
private System.Windows.Forms.PictureBox picStudy;
private System.Windows.Forms.GroupBox groupBox1;
private System.Windows.Forms.Panel panel4;
private System.Windows.Forms.Label label14;
private System.Windows.Forms.Label label15;
private System.Windows.Forms.Label label17;
```

```
private System.Windows.Forms.Panel panel6;
private System.Windows.Forms.Label label20;
private System.Windows.Forms.Label label21;
private System.Windows.Forms.Label label22;
private System.Windows.Forms.Label label23;
private System.Windows.Forms.GroupBox gbProcessType;
private System.Windows.Forms.RadioButton optArea;
private System.Windows.Forms.RadioButton optWhole;
private System.Windows.Forms.Label lblPictureInfo;
private System.Windows.Forms.Button cmdPictureProcess;
private System.Windows.Forms.PictureBox picPicturePreview;
private System.Windows.Forms.PictureBox picPictureMatchY;
private System.Windows.Forms.PictureBox picPictureMatchX;
private System.Windows.Forms.PictureBox picPictureMatchArea;
private System.Windows.Forms.PictureBox picPictureMatchPerimeter;
private System.Windows.Forms.Label lblPictureTime;
private System.Windows.Forms.Label lblPictureYDistance;
private System.Windows.Forms.Label lblPictureXDistance;
private System.Windows.Forms.Label lblPictureArea;
private System.Windows.Forms.Label lblPicturePerimeter;
private System.Windows.Forms.ContextMenu cmPicturePreview;
private System.Windows.Forms.MenuItem mnuPictureShow;
private System.Windows.Forms.ToolBarButton tbbSingleCapture;
private System.Windows.Forms.ToolBarButton tbbContinuousCapture;
private System.Windows.Forms.ToolBar tbCamVideoType;
private System.Windows.Forms.Timer tmContinuousMode;
private System.Windows.Forms.Label lblPictureEdges;
private System.Windows.Forms.Label label5;
private System.Windows.Forms.PictureBox picPictureMatchEdges;
private System.Windows.Forms.PictureBox picMatchEdges;
private System.Windows.Forms.Label lblEdges;
private System.Windows.Forms.Label label7;
private System.Windows.Forms.ImageList ilBrightnessContrast;
private System.Windows.Forms.GroupBox groupBox2;
private System.Windows.Forms.RadioButton optNever;
private System.Windows.Forms.RadioButton optObject;
private System.Windows.Forms.RadioButton optNoMatch;
private System.Windows.Forms.RadioButton optMatch;
private System.Windows.Forms.Timer tmVideo;
private System.Windows.Forms.ImageList ilControls;
//  Bitmap variable used to store the bitmap currently
//  displayed in Still Image mode.
private Bitmap bitmap;
//  Integer Variables used to store the individual values
//  values read off of the model's .mdl file.
int PerimeterLow, PerimeterHigh;
```

```
int AreaLow, AreaHigh;
int XDistanceLow, XDistanceHigh;
int YDistanceLow, YDistanceHigh;
int EdgesLow, EdgesHigh;
//  String variable used to store the name of the current
//  model.
string ModelName;
//  String variable used to pass the location of the Models
//  directory to the rest of the program
private string RootDirectory;
// String variable used to store the name of the current file.
private string filename;
//  Arraylist variable used to store all of the Edge graph strings
//  read off of the model's .mdl-file.
private ArrayList PreviousGraphs = new ArrayList();
//  Arraylist variable used to store all of the Edge graph strings
//  created by the EdgeGraphAgent during Single File Processing.
private ArrayList CurrentGraphs = new ArrayList();
//  Time variables used to store the recognition start and end times.
private System.DateTime StartTime;
private System.TimeSpan TotalTime;
//  Windows Image Aquisition Device variable used
//  to store the currently loaded Capture Device.
private WIA.Device CurrentCam;
//  WIA manager COM object used to store the current system WIA
//  information.
private WiaClass            wiaManager;
// WIA ItemClass object used to store the currently selected capture
//  device.
private ItemClass           wiaCamera;
// WIAVideoClass object used to store the current video stream.
private WiaVideoClass       wiaVideo;
//  Video variable used to store the current video object.
private Video CurrentVideo;
//  Integer variable used to store the current state of the
//  video stream.
private int PlayPause;
//  Integer variable used to keep track of the number of times the user has
//  clicked on picStudy, after he or she has selected Single File Mode.
private int clickcounter;
//  Integer variable used to keep track of the points of the rectangle drawn
//  on picStudy during the Single File Mode image selection process.
private int TopLeftX, TopLeftY;
private int BottomLeftX, BottomLeftY;
private int BottomRightX, BottomRightY;
private int TopRightX, TopRightY;
```

```
private int CentreX, CentreY;
//  Point variables used to keep track of where on picStudy the user
//  started drawing a rectangle.
private Point ptOriginal = new Point();
private Point Corner1 = new Point();
//  Point variables used to keep track of where on picStudy the user
//  stopped drawing a rectangle.
private Point ptLast = new Point();
private Point Corner2 = new Point();
//  Boolean variable used to keep track of whether the Mouse is down
//  (true) or up (false).
private Boolean bHaveMouse;
//  Integer variable used to keep track of whether the Continuous Mode
//  has to be started or stopped.
int FlagActivate;
//  Variables used to keep track of conveyer measurements and current
//  object position.
private double XMeasurement, YMeasurement, Speed, Direction;
public double CurrentConveyerX, CurrentConveyerY;
//  Model struct array used to keep track of all the currently loaded
//  Models' measurements.
private Model[] LoadedModels;
//  Integer variables used to keep track of individual Models' Match
//  Percentage.
private int MatchPercentage1, MatchPercentage2, MatchPercentage3,
        MatchPercentage4;
//  MatchStruct struct array used to keep track of all the currently loaded
//  Models' specific field matches.
private MatchStruct[] Matches;
//  Integer variables used to store the values returned from the agents.
private int gotPerimeter, gotArea, gotXDistance, gotYDistance, gotEdges;
//  Integer variable used to calculate the minimum match perimeter.
private int MatchPerimeter;
//  Stream used to write the Log File.
private StreamWriter LogWriter;
//  Variables used to maintain and activate the system's agents.
private Whiteboard whiteboard;
private MainEdgeDetectionAgent MEDagent;
private Thread MEDthread;
private InvertColoursAgent ICagent;
private Thread ICthread;
private ToBinaryAgent TBagent;
private Thread TBthread;
private PerimeterAgent Pagent;
private Thread Pthread;
private EdgeGraphAgent EGagent;
```

```
        private Thread EGthread;
```

1.9.3 Form Initialization

```
//  Startup subroutine for frmRecognition.
//  Takes 3 arguments:  - A reference to the current model's directory.
//                      - A reference to the model's name.
//                      - A reference to the string of models selected.
public frmRecognition(string root, string modelname, string[] ModelList)
{
        InitializeComponent();
        //  Assigned passed variables to their local counterparts.
        RootDirectory = root;
        ModelName = modelname;
        //  Create a new Model struct of the passed variable's
        //  size.
        LoadedModels = new Model[ModelList.Length];
        //  Create a new MatchStruct struct of the passed variable's
        //  size.
        Matches = new MatchStruct[ModelList.Length];
        //  Open up the file to write the Recognition Log entries.
         LogWriter = new StreamWriter(Application.StartupPath +
                     "\\RecognitionLog.txt");
        //  String used to store the form's header text.
        string HeaderText = "";
        //  Initialize the variable used to calculate the
        //  minimum object perimeter.
        double Calculator = 0;
        //  Create and assign the form's header.
        if (ModelList.Length == 1)
            HeaderText = "Perform recognition using the " + ModelList[0] + "
             model";
        if (ModelList.Length == 2)
            HeaderText = "Perform recognition using the " + ModelList[0] + " and " +
                         ModelList[1] + " models";
        if (ModelList.Length == 3)
            HeaderText = "Perform recognition using the " + ModelList[0] + ", " +
                         ModelList[1] + " and " + ModelList[2] +" models";
        if (ModelList.Length == 4)
            HeaderText = "Perform recognition using the " + ModelList[0] + ", " +
                         ModelList[1] + ", " + ModelList[2] + " and " + ModelList[3]
                         +" models";
        this.Text = HeaderText;
        // Initialize form variables.
        clickcounter = 2;
        FlagActivate = 0;
```

```
picStudy.Enabled = false;
//  Variable used to store values read off the config
//  files
string compare = "";
//  Initialize variables.
PerimeterLow = PerimeterHigh = AreaLow = AreaHigh = 0;
XDistanceLow = XDistanceHigh = YDistanceLow = YDistanceHigh = 0;
MatchPerimeter = Int32.MaxValue;
// Stream Reader variable used to read model config files.
TextReader tr;
//  Read the values off the config files.
if (Directory.Exists(RootDirectory))
{
        for (int j = 0; j < ModelList.Length; j++)
        {
                if (File.Exists(RootDirectory + "\\" + ModelList[j] + "\\" +
                    ModelList[j] + ".mdl"))
                {
                        LoadedModels[j].Name = ModelList[j];
                        tr = new StreamReader(RootDirectory + "\\" +
                              ModelList[j] + "\\" + ModelList[j] + ".mdl");
                        for (int i = 0; i < 10; i++)
                        {
                                compare = tr.ReadLine();
                                if (i == 0)
                                {
                                        if (compare == null)
                                        LoadedModels[j].PerimeterLow = 0;
                                        else
                                        LoadedModels[j].PerimeterLow =
                                             Int32.Parse(compare);
                                        if (LoadedModels[j].PerimeterLow <
                                           MatchPerimeter)
                                            MatchPerimeter =
                                            LoadedModels[j].PerimeterLow;
                                }
                                if (i == 1)
                                        if (compare == null)
                                        LoadedModels[j].PerimeterHigh = 0;
                                        else
                                        LoadedModels[j].PerimeterHigh =
                                            Int32.Parse(compare);
                                if (i == 2)
                                        if (compare == null)
                                        LoadedModels[j].AreaLow = 0;
                                        else
```

```
                    LoadedModels[j].AreaLow =
                        Int32.Parse(compare);
                if (i == 3)
                    if (compare == null)
                    LoadedModels[j].AreaHigh = 0;
                    else
                    LoadedModels[j].AreaHigh =
                        Int32.Parse(compare);
                if (i == 4)
                    if (compare == null)
                    LoadedModels[j].XDistanceLow = 0;
                    else
                    LoadedModels[j].XDistanceLow =
                         Int32.Parse(compare);
                if (i == 5)
                    if (compare == null)
                    LoadedModels[j].XDistanceHigh = 0;
                    else
                    LoadedModels[j].XDistanceHigh =
                       Int32.Parse(compare);
                if (i == 6)
                    if (compare == null)
                    LoadedModels[j].YDistanceLow = 0;
                    else
                    LoadedModels[j].YDistanceLow =
                         Int32.Parse(compare);
                if (i == 7)
                    if (compare == null)
                    LoadedModels[j].YDistanceHigh = 0;
                    else
                    LoadedModels[j].YDistanceHigh =
                       Int32.Parse(compare);
                if (i == 8)
                    if (compare == null)
                    LoadedModels[j].EdgesLow = 0;
                    else
                    LoadedModels[j].EdgesLow =
                      Int32.Parse(compare);
                if (i == 9)
                    if (compare == null)
                    LoadedModels[j].EdgesHigh = 0;
                    else
                    LoadedModels[j].EdgesHigh =
                       Int32.Parse(compare);
            }
            while ((compare = tr.ReadLine()) != null)
```

```
                        {
                                PreviousGraphs.Add(compare);
                        }
                        tr.Close();
                }
                else
                {
                        LoadedModels[j].Name = "No File";
                        File.Create(RootDirectory + "\\" + ModelList[j] + "\\" +
                              ModelList[j] + ".mdl");
                }
        }
}
else
{
        for (int i = 0; i < ModelList.Length; i++)
        {
                Directory.CreateDirectory(RootDirectory + "\\" + ModelList[i]);
                File.Create(RootDirectory + "\\" + ModelList[i] + "\\" +
                     ModelList[i] + ".mdl");
        }
}
//  Calculate the minimum perimeter.
Calculator = (double) MatchPerimeter;
MatchPerimeter =  (int)(Calculator * 75 / 100);

//  Read the values off the Conveyer Dimensions config file.
try
{
        if (File.Exists(Application.StartupPath + "\\Dimensions.cfg"))
        {
                StreamReader sr = new
                    StreamReader(Application.StartupPath +
                    "\\Dimensions.cfg");
                XMeasurement = Int32.Parse(sr.ReadLine());
                YMeasurement = Int32.Parse(sr.ReadLine());
                Speed = Int32.Parse(sr.ReadLine());
                Direction = Int32.Parse(sr.ReadLine());
                if ((XMeasurement == 0) || (YMeasurement == 0) || (Speed
                    == 0))
                MessageBox.Show("Conveyer dimensions incomplete;
                        reported measurements may be
                        inaccurate.","Message",
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Information);
                sr.Close();
```

```
        }
        else
        {
                MessageBox.Show("No data found on conveyer dimensions;
                        reported measurements may be
                        inaccurate.","Message",MessageBoxButtons.OK,
                        MessageBoxIcon.Information);
                XMeasurement = 0;
                YMeasurement = 0;
                Speed = 0;
                Direction = 0;
        }
}
catch (Exception exception)
{
        MessageBox.Show("Error in conveyer dimensions; reported
                        measurements may be inaccurate.",
                        "Message",MessageBoxButtons.OK,
                         MessageBoxIcon.Information);
        XMeasurement = 0;
        YMeasurement = 0;
        Speed = 0;
        Direction = 0;
}

//  Set up the toolbar buttons.

tbStillImage.Buttons[0].Enabled = true;
tbStillImage.Buttons[1].Enabled = true;
tbStillImage.Buttons[2].Enabled = true;
//  Hide the Match boxes..
picMatchPerimeter.Visible = false;
picMatchArea.Visible = false;
picMatchX.Visible = false;
picMatchY.Visible = false;
picMatchEdges.Visible = false;
//  Hide the video controls.
cmdPlayPause.Visible = false;
cmdStop.Visible = false;
//  Set the Video to playmode
PlayPause = 2;
//  Hide the camera controls.
cmdCamPlayPause.Visible = false;
cmdCamPlayPause.Enabled = false;
//  Make sure no capture device is loaded.
CurrentCam = null;
```

```
        //  Make sure no tabpages are loaded.
        tabRecognition.TabPages.Remove(tabPicture);
        tabRecognition.TabPages.Remove(tabCamVideo);
 }
```

1.9.4 Form Closing

```
//  This subroutine is called when frmRecognition closes.
//  It closes the Recognition Log File writer.
private void frmRecognition_Closing(object sender,
                System.ComponentModel.CancelEventArgs e)
{
        LogWriter.Close();
}
```

1.9.5 Mouse Down on Single File Mode image

```
//  This subroutine is called when the left mouse button is pressed.
public void MyMouseDown( Object sender, MouseEventArgs e )
{
        //  Checks whether Single File Mode has been selected.
        if (clickcounter < 1)
        {
                //  Assigns the current position to a point.
                Point A = new Point(e.X,e.Y);
                Corner1 = A;
                // Mark the mouse as being down.
                bHaveMouse = true;
                // Store the starting point for drawing the rectangle.
                ptOriginal.X = e.X;
                ptOriginal.Y = e.Y;
                // Initialize the end points of the rectangle.
                ptLast.X = -1;
                ptLast.Y = -1;
                //  Increment the clickcounter.
                clickcounter++;
        }
}
```

1.9.6 Mouse Up on Single File Mode Image

```
//  This subroutine is called when the left mouse button is released.
public void MyMouseUp( Object sender, MouseEventArgs e )
{
        //  Assigns the current position to a point.
        Point A = new Point(e.X,e.Y);
```

```
Corner2 = A;
// Mark the mouse as being up.
bHaveMouse = false;
//  Draw the final dragging rectangle
if( ptLast.X != -1 )
{
        //  Assigns the current position to a point and
        //  calls the subroutine to erase the previous
        //  rectangle
        Point ptCurrent = new Point( e.X, e.Y );
        MyDrawReversibleRectangle( ptOriginal, ptLast );
}
// Re-initialize the end and start points of the rectangle.
ptLast.X = -1;
ptLast.Y = -1;
ptOriginal.X = -1;
ptOriginal.Y = -1;
//  Make sure which point is top left and which is
//  bottom right.
if (Corner2.Y > Corner1.Y)
{
        TopLeftY = Corner1.Y;
        BottomRightY = Corner2.Y;
}
else
{
        TopLeftY = Corner2.Y;
        BottomRightY = Corner1.Y;
}
if (Corner2.X > Corner1.X)
{
        TopLeftX = Corner1.X;
        BottomRightX = Corner2.X;
}
else
{
        TopLeftX = Corner2.X;
        BottomRightX = Corner1.X;
}
//  Create a temporary bitmap value and assign it a reference
//  to the bitmap currently displayed in picturebox
//  picStudy.
Bitmap Main;
Main = (Bitmap) bitmap.Clone();
//  Set the cloned bitmap up as a Graphics object, so that it
//  can be drawn on.
```

```
            Graphics g = Graphics.FromImage(Main);
            //  Create a new pen, with which to draw.
            Pen myPen = new Pen(Color.Red);
            myPen.Width = 3;
            //  Temporary storage value used in calculating
            //  relative corner values for the drawing
            //  rectangle.
            double newvalue = 0;
            //  Calculate the relative values, so that the
            //  rectangle will be correctly aligned on
            //  picturebox pixStudy.
            newvalue = TopLeftX * (double)(Main.Width /640.0);
            BottomLeftX = TopLeftX = (int) newvalue;
            newvalue = 0;
            newvalue = BottomRightY * (double)(Main.Height /480.0);
            BottomLeftY = BottomRightY = (int) newvalue;
            newvalue = 0;
            newvalue = BottomRightX * (double)(Main.Width /640.0);
            TopRightX = BottomRightX = (int) newvalue;
            newvalue = 0;
            newvalue = TopLeftY * (double)(Main.Height /480.0);
            TopRightY = TopLeftY = (int) newvalue;
            newvalue = 0;
            CentreX = TopLeftX + ((TopRightX - TopLeftX) / 2);
            CentreY = TopLeftY + ((BottomLeftY - TopLeftY) / 2);
            //  Draw the rectangle
            g.DrawRectangle(myPen,TopLeftX,TopLeftY,TopRightX –
                    TopLeftX,BottomLeftY - TopLeftY);
            //  Set the picturebox picStudy's image to the newly
            //  drawn on bitmap and size it appropriately.
            picStudy.Image = Main;
            picStudy.SizeMode = PictureBoxSizeMode.StretchImage;
            //  Sets up the arrangement for frmUpdateModel's
            //  toolbar buttons.
            tbStillImage.Buttons[0].Enabled = true;
            tbStillImage.Buttons[1].Enabled = true;
            tbStillImage.Buttons[2].Enabled = true;
            //  Calls the Still Image Area Recognition subroutine.
            StillAreaRecognition();
}
```

1.9.7 Mouse Move on Single File Mode Image

```
// This subroutine is called when the mouse is moved.
public void MyMouseMove( Object sender, MouseEventArgs e )
{
	//  Assigns the current position to a point.
	Point ptCurrent = new Point( e.X, e.Y );
	// Checks whether the mouse is down.
	if( bHaveMouse )
	{
		//  The mouse is down
		//  Checks whether this is the first time this
		//  subroutine has run.
		if( ptLast.X != -1 )
		{
			//  Not the first time
			//  Calls the subroutine to erase the previous
			//  rectangle.
			MyDrawReversibleRectangle( ptOriginal, ptLast);
		}
		// Update the last point.
		ptLast = ptCurrent;
		// Draw the new rectangle.
		MyDrawReversibleRectangle( ptOriginal, ptLast);
	}
}
```

1.9.8 Draw Rectangle on Single File Mode image

```
//  This subroutine draws the dragging rectangle.
private void MyDrawReversibleRectangle( Point p1, Point p2 )
{
	//  Create a new rectangle, to be used for drawing a
	//  rectangle on screen.
	Rectangle rc = new Rectangle();
	//  Convert the picStudy coordinates to screen coordinates.
	p1 = picStudy.PointToScreen( p1 );
	p2 = picStudy.PointToScreen( p2 );
	//  Make sure which point is top left and which is
	//  bottom right.  Then calculate the height and
	//  width of the rectangle.
	if( p1.X < p2.X )
	{
		rc.X = p1.X;
		rc.Width = p2.X - p1.X;
	}
```

```
        else
        {
                rc.X = p2.X;
                rc.Width = p1.X - p2.X;
        }
        if( p1.Y < p2.Y )
        {
                rc.Y = p1.Y;
                rc.Height = p2.Y - p1.Y;
        }
        else
        {
                rc.Y = p2.Y;
                rc.Height = p1.Y - p2.Y;
        }
        //  Draw/Erase the temporary rectangle.
        ControlPaint.DrawReversibleFrame(rc,Color.Red, FrameStyle.Dashed);
}
```

1.9.9 Override Mouse Functions

```
// This subroutine defines the overloads for the mouse events.
protected override void OnLoad(System.EventArgs e)
{
        picStudy.MouseDown += new MouseEventHandler( MyMouseDown );
        picStudy.MouseUp += new MouseEventHandler( MyMouseUp );
        picStudy.MouseMove += new MouseEventHandler( MyMouseMove );
        bHaveMouse = false;
}
```

1.9.10 Form Designer Generated Code

```
/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
        if( disposing )
        {
                //        DisposePicure();
                if( wiaVideo != null )
                   // release any COM instances
                   Marshal.ReleaseComObject( wiaVideo );
                wiaVideo = null;
                if( wiaCamera != null )
                     Marshal.ReleaseComObject( wiaCamera );
```

```
                wiaCamera = null;
                if( wiaManager != null )
                    Marshal.ReleaseComObject( wiaManager );
                wiaManager = null;
                if (components != null)
                {
                        components.Dispose();
                }
        }
        base.Dispose( disposing );
}

#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
        //  Component code omitted.
}
#endregion
```

1.9.11 StillImage toolbar

```
//  This subroutine acts as a switchboard to determine which
//  button on the Still Image toolbar was clicked.
private void tbStillImage_ButtonClick(object sender,
                System.Windows.Forms.ToolBarButtonClickEventArgs e)
{
        switch (tbStillImage.Buttons.IndexOf(e.Button))
        {
          //  Still Image Mode.
          case 0: SelectImage();
                        break;
          //  Video Mode.
          case 1: SelectVideo();
                        break;
          //  Camera Mode.
          case 2: CameraMode();
                        break;
        }
}
```

1.9.12 Reset Page

```
//  This subroutine is called when the page needs to be reset and all the
//  variables re-initialized.
private void ResetPage()
{
        //  Reset form variables.
        clickcounter = 2;
        FlagActivate = 0;
        PlayPause = 2;
        //  Reset components.
       picStudy.Image = null;
        lblTime.Text = "";
        picPicturePreview.Image = null;
        picPicturePreview.BackColor = Color.FromName("Control");
        lblPicturePerimeter.Text = null;
        lblPictureArea.Text = null;
        lblPictureXDistance.Text = null;
        lblPictureYDistance.Text = null;
        lblPictureEdges.Text = null;
        lblPictureTime.Text = null;
        optWhole.Checked = true;
        lblVidCurrent.Text = null;
        lblVidTotal.Text = null;
        chkMute.Checked = false;
        chkLoop.Checked = false;
        tbVideo.Value = 0;
        tbVidVolume.Value = -1;
        tmVideo.Enabled = false;tbCamVideoType.Buttons[0].Pushed = true;
        tbCamVideoType.Buttons[1].Pushed = false;
        lblInfo.Text = "";
        cmdPlayPause.ImageIndex = 0;
        //  Hide the following components.
        picPictureMatchPerimeter.Visible = false;
        picPictureMatchArea.Visible = false;
        picPictureMatchX.Visible = false;
        picPictureMatchY.Visible = false;
        picPictureMatchEdges.Visible = false;
        picMatchPerimeter.Visible = false;
        picMatchArea.Visible = false;
        picMatchX.Visible = false;
        picMatchY.Visible = false;
        picMatchEdges.Visible = false;
        cmdPlayPause.Visible = false;
        cmdStop.Visible = false;
        gbVidTimes.Visible = false;
```

```
        tbVideo.Visible = false;
        tbVidVolume.Visible = false;
        chkMute.Visible = false;
        chkLoop.Visible = false;
        //  Disable the following components.
        picStudy.Enabled = false;
        tbStillImage.Buttons[0].Enabled = true;
        tbStillImage.Buttons[1].Enabled = true;
        tbStillImage.Buttons[2].Enabled = true;
        tmVideo.Enabled = false;
        tmContinuousMode.Enabled = false;
        tbRecognitionPicAdjustments.Enabled = true;
        gbProcessType.Enabled = true;
        cmdPictureProcess.Enabled = true;
        tbRecognitionPicAdjustments.Buttons[0].Enabled = true;
        tbRecognitionPicAdjustments.Buttons[1].Enabled = true;
        //  Remove the tabpages.
        tabRecognition.TabPages.Remove(tabPicture);
        tabRecognition.TabPages.Remove(tabCamVideo);
        //  Remove the currently playing video (if any)
        if (CurrentVideo != null)
        {
                CurrentVideo.Stop();
        }
        CurrentVideo = null;
}
```

1.9.13 Select Image for Single Image Mode

```
//  This subroutine is called when the user needs to
//  load an image file for Single Image Mode.
private void SelectImage()
{
        //  Set up and display the Open File Dialog Box.
        ofdPicture.Filter = "JPEG Files (*.jpg)|*.jpg|Bitmap Files (*.bmp)|*.bmp|GIF
                Files (*.gif)|*.gif|All Files (*.*)|*.*)";
        ofdPicture.RestoreDirectory = true ;
        if(ofdPicture.ShowDialog() == DialogResult.OK)
        {
                //  Reset the Page.
                ResetPage();
                //  Load the image file.
                filename = ofdPicture.FileName;
                bitmap = (Bitmap)Bitmap.FromFile(filename,false);
                //  Refresh the page.
                this.Invalidate();
```

```
                //  Assign the image to the picturebox.
                picStudy.Image = bitmap;
                picStudy.SizeMode = PictureBoxSizeMode.StretchImage;
                //  Add the Still Image tabpage and set its properties.
                tabRecognition.TabPages.Add(tabPicture);
                lblPictureInfo.Text = "Picture Loaded";
                tabPicture.Text = "Still Image Mode:  " + ofdPicture.FileName;
                // Creates the agent communication directory.
                if (Directory.Exists(Application.StartupPath + "\\Recognition")
                    == true)
                  Directory.Delete(Application.StartupPath + "\\Recognition",true);
                Directory.CreateDirectory(Application.StartupPath +
                  "\\Recognition");

        }
}
```

1.9.14 Select Video for Video Mode

```
//  This subroutine is called when the user needs to
//  load a video file for Video Mode.
private void SelectVideo()
{
        //  Set up and display the Open File Dialog Box.
        ofdPicture.Filter = "AVI Files (*.avi)|*.avi|MPEG Files (*.mpg)|*.mpg|All
                            Files (*.*)|*.*";
        ofdPicture.RestoreDirectory = true ;
        if(ofdPicture.ShowDialog() == DialogResult.OK)
          {
                //  Variables used to calculate the video's total
                //  runtime.
                int hours, minutes, seconds;
                //  Reset the page.
                ResetPage();
                //  Make the video controls visible.
                cmdPlayPause.Visible = true;
                cmdStop.Visible = true;
                picCam.Visible = true;
                gbVidTimes.Visible = true;
                tbVideo.Visible = true;
                tbVidVolume.Visible = true;
                chkMute.Visible = true;
                chkLoop.Visible = true;
                //  Set up the toolbar's buttons.
                tbStillImage.Buttons[0].Enabled = true;
                tbStillImage.Buttons[1].Enabled = true;
```

```
    tbStillImage.Buttons[2].Enabled = true;
    //  Create and set up the video object.
    filename = ofdPicture.FileName;
    CurrentVideo = new Video(ofdPicture.FileName);
    CurrentVideo.Owner = picCam;
    picCam.Width = 640;
    picCam.Height = 480;
    //  Play and pause the video to display its first
    //  frame.
    CurrentVideo.Play();
    CurrentVideo.Pause();
    // Assigns the current video time position to
    //  the label.
    lblVidCurrent.Text = "00:00:00";
    //  Calculate and display the total video
    //  runtime.
    seconds = (int)CurrentVideo.Duration % 60;
    minutes = (int)CurrentVideo.Duration / 60 % 60;
    hours = (int)CurrentVideo.Duration /60 / 60;
    if (hours < 10)
        lblVidTotal.Text = "0" + hours.ToString() + ":";
    else
        lblVidTotal.Text = hours.ToString() + ":";
    if (minutes < 10)
         lblVidTotal.Text = lblVidTotal.Text + "0" + minutes.ToString() +
                                ":";
    else
         lblVidTotal.Text = lblVidTotal.Text + minutes.ToString() + ":";
    if (seconds < 10)
         lblVidTotal.Text = lblVidTotal.Text + "0" + seconds.ToString();
    else
         lblVidTotal.Text = lblVidTotal.Text + seconds.ToString();
    //  Set the Video Position slider's Maximum value
    //  to the video's runtime.
    tbVideo.Maximum = (int) CurrentVideo.Duration;
    //  Add the Cam/Video tabpage and set its properties.
    tabRecognition.TabPages.Add(tabCamVideo);
    tabCamVideo.ImageIndex = 1;
    tabCamVideo.Text = "Video Mode:  " + ofdPicture.FileName;
  // Creates the agent communication directory.
  if (Directory.Exists(Application.StartupPath + "\\Recognition")
      == true)
    Directory.Delete(Application.StartupPath + "\\Recognition",true);
  Directory.CreateDirectory(Application.StartupPath +
    "\\Recognition");
```

```
        }
}
```

1.9.15 Single Image Whole Image Mode

```
//  This subroutine is called when Recognition needs to be
//  done on a Still Image as a whole.
private void StillRecognition()
{
        //  variables used to calculate the bounding box.
        int topx, topy, bottomx, bottomy;
        //  Variables used to determine which Model matched.
        int MainMatch, HighestMatch;
        //  Variable used to store the log line.
        string Log = "";
        //  Add to Log File line.
        Log = "Still;";
         //  Creates the shared Whiteboard object. Then initializes all of the agents
        //  and loads them onto individual threads.
        whiteboard = new Whiteboard((Bitmap) bitmap.Clone());
        MEDagent = new  MainEdgeDetectionAgent(640,480,whiteboard);
        MEDthread = new Thread(new ThreadStart(MEDagent.StartWait));
        MEDthread.Start();
        ICagent = new InvertColoursAgent(640,480,whiteboard);
        ICthread = new Thread(new ThreadStart(ICagent.StartWait));
        ICthread.Start();
        TBagent = new ToBinaryAgent(640,480,whiteboard);
        TBthread = new Thread(new ThreadStart(TBagent.StartWait));
        TBthread.Start();
        Pagent = new PerimeterAgent(whiteboard,5,5,635,475,2);
        Pthread = new Thread(new ThreadStart(Pagent.StartWait));
        Pthread.Start();
        EGagent = new EdgeGraphAgent(whiteboard,5,435,5,635);
        EGthread = new Thread(new ThreadStart(EGagent.StartWait));
        EGthread.Start();
        //  Add to Log File line.
        Log = "Still;";
        if (optWhole.Checked == true)
           Log = Log + "Whole;" + filename + ";" +
           DateTime.Now.ToShortDateString() + ";"  +
           DateTime.Now.ToLongTimeString() + ";";
       else
           Log = Log + "Area;" + filename + ";" +
           DateTime.Now.ToShortDateString() + ";" +
           DateTime.Now.ToLongTimeString() + ";";
        //  Set start time for recognition process.
```

```
StartTime = System.DateTime.Now;
//  Initialize the bounding box.
topx = bitmap.Width - 1;
topy = bitmap.Height - 1;
bottomx = bottomy = 0;
//  Show the user that the program is processing.
System.Windows.Forms.Cursor.Current =
          System.Windows.Forms.Cursors.WaitCursor;
//  Increment the counter to let the agents know that they may start
//  processing.
whiteboard.Increment();
//  Halts all of the threads when they have finished processing.
while (whiteboard.RunStartWait == 0)
{
          if (whiteboard.Counter > 4)
          {
             //  Display the found object's perimeter.
            picPicturePreview.BackColor = Color.White;
            picPicturePreview.Image = whiteboard.picInUseDone;
            picPicturePreview.SizeMode = PictureBoxSizeMode.StretchImage;
            //  Get the perimeter value and determine if its higher than the
            //  minimum value.
           gotPerimeter = Pagent.getPerimeter();
           if (gotPerimeter >= MatchPerimeter)
           {
             //  Object Found.
             //  Add to Log File line.
             Log = Log + "Y;";
             //  Set the bounding box's values.
             topx = Pagent.getLowestX();
             topy = Pagent.getLowestY();
             bottomx = Pagent.getHighestX();
             bottomy = Pagent.getHighestY();
             //  Set the global variables to the calculated
             //  values.
             gotArea = Pagent.getArea();
             gotXDistance = Pagent.getXDistance();
             gotYDistance = Pagent.getYDistance();
             //  Display the calculated values in the labels.
             lblPicturePerimeter.Text = gotPerimeter.ToString();
             lblPictureArea.Text = gotArea.ToString();
             lblPictureXDistance.Text = gotXDistance.ToString();
             lblPictureYDistance.Text = gotYDistance.ToString();
             lblPictureEdges.Text = "Not Tested";
             //  Initialize the search variables.
             HighestMatch = 0;
```

```
MatchPercentage1 = MatchPercentage2 = MatchPercentage3 =
                        MatchPercentage4 = 0;
//  See if there's a match on the first Model.
MatchModelOne();
MainMatch = 0;
HighestMatch = MatchPercentage1;
//  See if there's a higher match on the second model.
if ((HighestMatch != 80) && (LoadedModels.Length == 2))
{
        MatchModelTwo();
        if (MatchPercentage2 > HighestMatch)
        {
                MainMatch = 1;
                HighestMatch = MatchPercentage2;
        }
}
//  See if there's a higher match on the third model.
if ((HighestMatch != 80) && (LoadedModels.Length == 3))
{
        MatchModelThree();
        if (MatchPercentage3 > HighestMatch)
        {
                MainMatch = 2;
                HighestMatch = MatchPercentage3;
        }
}
//  See if there's a higher match on the fourth model.
if ((HighestMatch != 80) && (LoadedModels.Length == 4))
{
        MatchModelFour();
        if (MatchPercentage4 > HighestMatch)
        {
                MainMatch = 3;
                HighestMatch = MatchPercentage4;
        }
}
//  Set the match pictures according to the best matched Model.
picPictureMatchPerimeter.Visible =
      Matches[MainMatch].Perimeter;
picPictureMatchArea.Visible = Matches[MainMatch].Area;
picPictureMatchX.Visible = Matches[MainMatch].XDistance;
picPictureMatchY.Visible = Matches[MainMatch].YDistance;
//  If the highest match was 60, see if the EdgeGraphs match.
if (HighestMatch == 60)
{
        whiteboard.Increment();
```

```
            int check = 0;
            while (check == 0)
            {
                If (whiteboard.Counter == 7)
                {
                   check = 1;
                  gotEdges = EGagent.getEdgeCount();
                  if ((gotEdges >=
                    LoadedModels[MainMatch].EdgesLow) &&
                    (gotEdges <=
                    LoadedModels[MainMatch].EdgesHigh))
                     {
                        HighestMatch = HighestMatch + 20;
                        picPictureMatchEdges.Visible = true;
                        lblPictureEdges.Text = gotEdges.ToString();
                        Matches[MainMatch].Log =
                        Matches[MainMatch].Log + "Y;";
                     }
                 else
                    Matches[MainMatch].Log =
                       Matches[MainMatch].Log + "N;";
                }
            }
}
else
        Matches[MainMatch].Log = Matches[MainMatch].Log + " ;";
 //  Ends the agents' waiting process and stops the agents.
 whiteboard.RunStartWait = 1;
 MEDthread.Join();
  ICthread.Join();
 TBthread.Join();
  Pthread.Join();
 EGthread.Join();
 //  Set end time for recognition process.
 TotalTime = System.DateTime.Now.Subtract(StartTime);
  //  Draw the bounding box.
 for (int x = topx; x <= bottomx; x++)
        bitmap.SetPixel(x,topy,Color.Red);
 for (int x = topx; x <= bottomx; x++)
        bitmap.SetPixel(x,bottomy,Color.Red);
 for (int y = topy; y <= bottomy; y++)
        bitmap.SetPixel(topx,y,Color.Red);
 for (int y = topy; y <= bottomy; y++)
        bitmap.SetPixel(bottomx,y,Color.Red);
 //  Refresh the image.
 picStudy.Refresh();
```

```
//  Determine if a match was made or not.
if (HighestMatch >= 80)
{
        //  Match made.
        //  Add to Log File line.
        Log = Log + "Y;" + TotalTime.Milliseconds.ToString() + ";" +
        HighestMatch.ToString() + "% match found on " +
                LoadedModels[MainMatch].Name + ";" +
                gotPerimeter.ToString() + ";" + gotArea.ToString() +
                ";" + gotXDistance.ToString() + ";" +
                gotYDistance.ToString() + ";";
        if (gotEdges == 0)
                Log = Log +  " ;";
        else
                Log = Log + gotEdges.ToString() + ";";
        Log = Log + Matches[MainMatch].Log;
        //  Display Match information to the user.
        lblPictureTime.Text = TotalTime.Seconds.ToString() + " s "
             + TotalTime.Milliseconds.ToString() + " ms";
             picStudy.Image = bitmap;
        picStudy.SizeMode = PictureBoxSizeMode.StretchImage;
        System.Windows.Forms.Cursor.Current =
                System.Windows.Forms.Cursors.Arrow;
        lblPictureInfo.Text = HighestMatch.ToString() + "% match
               found on " + LoadedModels[MainMatch].Name;
}
else
{
        //  No Match.
       //  Add to Log File line.
      Log = Log + "N;" + TotalTime.Milliseconds.ToString() + ";" +
                        "Insufficient Match Criteria (" +
                        HighestMatch.ToString() + "% match found on
                        " + LoadedModels[MainMatch].Name + ");" +
                        gotPerimeter.ToString() + ";" +
                        gotArea.ToString() + ";" +
                        gotXDistance.ToString() + ";" +
                        gotYDistance.ToString() + ";";
        if (gotEdges == 0)
                Log = Log +  " ;";
        else
                Log = Log + gotEdges.ToString() + ";";
        Log = Log + Matches[MainMatch].Log;

        //  Display Match information to the user.
        System.Windows.Forms.Cursor.Current =
```

```
                        System.Windows.Forms.Cursors.Arrow;
                    lblPictureInfo.Text = "Insufficient Match Criteria (" +
                        HighestMatch.ToString() + "% Match)";
                }
            }
            else
            {
                Log = Log + "N; ; ;Object couldn't be found; ; ; ; ; ; ; ; ; ;";
                lblPictureInfo.Text = "Object couldn't be found";
            }
            //  Set up the StillImage toolbar buttons.
            tbStillImage.Buttons[0].Enabled = true;
            tbStillImage.Buttons[1].Enabled = true;
            tbStillImage.Buttons[2].Enabled = true;
            //  Make sure the user can't click on picStudy.
            picStudy.Enabled = false;
            //  Write the Log File line.
            LogWriter.WriteLine(Log);
           }
        }
}
```

1.9.16 Single Image Area Mode

```
//  This subroutine is called when Recognition needs to be
//  done on an Area of a Still Image.
private void StillAreaRecognition()
        //  variables used to calculate the bounding box.
        int topx, topy, bottomx, bottomy;
        //  Variables used to determine which Model matched.
        int MainMatch, HighestMatch;
        //  Variable used to store the log line.
        string Log = "";
        //  Add to Log File line.
        Log = "Still;";
         //  Creates the shared Whiteboard object. Then initializes all of the agents
        //  and loads them onto individual threads.
        whiteboard = new Whiteboard((Bitmap) bitmap.Clone());
        MEDagent = new  MainEdgeDetectionAgent(640,480,whiteboard);
        MEDthread = new Thread(new ThreadStart(MEDagent.StartWait));
        MEDthread.Start();
        ICagent = new InvertColoursAgent(640,480,whiteboard);
        ICthread = new Thread(new ThreadStart(ICagent.StartWait));
        ICthread.Start();
        TBagent = new ToBinaryAgent(640,480,whiteboard);
        TBthread = new Thread(new ThreadStart(TBagent.StartWait));
```

```
TBthread.Start();
Pagent = new PerimeterAgent(whiteboard,
            TopLeftX,TopLeftY,BottomRightX,BottomRightY,2);
Pthread = new Thread(new ThreadStart(Pagent.StartWait));
Pthread.Start();
EGagent = new EdgeGraphAgent(whiteboard,
              TopLeftX,TopLeftY,BottomRightX,BottomRightY);
EGthread = new Thread(new ThreadStart(EGagent.StartWait));
EGthread.Start();
//  Add to Log File line.
Log = "Still;";
if (optWhole.Checked == true)
   Log = Log + "Whole;" + filename + ";" +
   DateTime.Now.ToShortDateString() + ";"  +
   DateTime.Now.ToLongTimeString() + ";";
else
   Log = Log + "Area;" + filename + ";" +
   DateTime.Now.ToShortDateString() + ";" +
   DateTime.Now.ToLongTimeString() + ";";
//  Set start time for recognition process.
StartTime = System.DateTime.Now;
//  Initialize the bounding box.
topx = bitmap.Width - 1;
topy = bitmap.Height - 1;
bottomx = bottomy = 0;
//  Show the user that the program is processing.
System.Windows.Forms.Cursor.Current =
          System.Windows.Forms.Cursors.WaitCursor;
//  Increment the counter to let the agents know that they may start
//  processing.
whiteboard.Increment();
//  Halts all of the threads when they have finished processing.
while (whiteboard.RunStartWait == 0)
{
         if (whiteboard.Counter > 4)
         {
            //  Display the found object's perimeter.
           picPicturePreview.BackColor = Color.White;
           picPicturePreview.Image = whiteboard.picInUseDone;
           picPicturePreview.SizeMode = PictureBoxSizeMode.StretchImage;
           //  Get the perimeter value and determine if its higher than the
           //  minimum value.
          gotPerimeter = Pagent.getPerimeter();
          if (gotPerimeter >= MatchPerimeter)
          {
            //  Object Found.
```

```
//  Add to Log File line.
Log = Log + "Y;";
//  Set the bounding box's values.
topx = Pagent.getLowestX();
topy = Pagent.getLowestY();
bottomx = Pagent.getHighestX();
bottomy = Pagent.getHighestY();
//  Set the global variables to the calculated
//  values.
gotArea = Pagent.getArea();
gotXDistance = Pagent.getXDistance();
gotYDistance = Pagent.getYDistance();
//  Display the calculated values in the labels.
lblPicturePerimeter.Text = gotPerimeter.ToString();
lblPictureArea.Text = gotArea.ToString();
lblPictureXDistance.Text = gotXDistance.ToString();
lblPictureYDistance.Text = gotYDistance.ToString();
lblPictureEdges.Text = "Not Tested";
//  Initialize the search variables.
HighestMatch = 0;
MatchPercentage1 = MatchPercentage2 = MatchPercentage3 =
                        MatchPercentage4 = 0;
//  See if there's a match on the first Model.
MatchModelOne();
MainMatch = 0;
HighestMatch = MatchPercentage1;
//  See if there's a higher match on the second model.
if ((HighestMatch != 80) && (LoadedModels.Length == 2))
{
        MatchModelTwo();
        if (MatchPercentage2 > HighestMatch)
        {
                MainMatch = 1;
                HighestMatch = MatchPercentage2;
        }
}
//  See if there's a higher match on the third model.
if ((HighestMatch != 80) && (LoadedModels.Length == 3))
{
        MatchModelThree();
        if (MatchPercentage3 > HighestMatch)
        {
                MainMatch = 2;
                HighestMatch = MatchPercentage3;
        }
}
```

```
//  See if there's a higher match on the fourth model.
if ((HighestMatch != 80) && (LoadedModels.Length == 4))
{
        MatchModelFour();
        if (MatchPercentage4 > HighestMatch)
        {
                MainMatch = 3;
                HighestMatch = MatchPercentage4;
        }
}
//  Set the match pictures according to the best matched Model.
picPictureMatchPerimeter.Visible =
      Matches[MainMatch].Perimeter;
picPictureMatchArea.Visible = Matches[MainMatch].Area;
picPictureMatchX.Visible = Matches[MainMatch].XDistance;
picPictureMatchY.Visible = Matches[MainMatch].YDistance;
//  If the highest match was 60, see if the EdgeGraphs match.
if (HighestMatch == 60)
{
        whiteboard.Increment();
        int check = 0;
        while (check == 0)
        {
            If (whiteboard.Counter == 7)
            {
              check = 1;
              gotEdges = EGagent.getEdgeCount();
              if ((gotEdges >=
                LoadedModels[MainMatch].EdgesLow) &&
                (gotEdges <=
                LoadedModels[MainMatch].EdgesHigh))
                 {
                    HighestMatch = HighestMatch + 20;
                    picPictureMatchEdges.Visible = true;
                    lblPictureEdges.Text = gotEdges.ToString();
                    Matches[MainMatch].Log =
                    Matches[MainMatch].Log + "Y;";
                 }
             else
                Matches[MainMatch].Log =
                   Matches[MainMatch].Log + "N;";
           }
       }
}
else
        Matches[MainMatch].Log = Matches[MainMatch].Log + " ;";
```

```
// Ends the agents' waiting process and stops the agents.
whiteboard.RunStartWait = 1;
MEDthread.Join();
ICthread.Join();
TBthread.Join();
Pthread.Join();
EGthread.Join();
// Set end time for recognition process.
TotalTime = System.DateTime.Now.Subtract(StartTime);
// Draw the bounding box.
for (int x = topx; x <= bottomx; x++)
        bitmap.SetPixel(x,topy,Color.Red);
for (int x = topx; x <= bottomx; x++)
        bitmap.SetPixel(x,bottomy,Color.Red);
for (int y = topy; y <= bottomy; y++)
        bitmap.SetPixel(topx,y,Color.Red);
for (int y = topy; y <= bottomy; y++)
        bitmap.SetPixel(bottomx,y,Color.Red);
// Refresh the image.
picStudy.Refresh();
// Determine if a match was made or not.
if (HighestMatch >= 80)
{
        // Match made.
        // Add to Log File line.
        Log = Log + "Y;" + TotalTime.Milliseconds.ToString() + ";" +
        HighestMatch.ToString() + "% match found on " +
                LoadedModels[MainMatch].Name + ";" +
                gotPerimeter.ToString() + ";" + gotArea.ToString() +
                ";" + gotXDistance.ToString() + ";" +
                gotYDistance.ToString() + ";";
        if (gotEdges == 0)
                Log = Log +  " ;";
        else
                Log = Log + gotEdges.ToString() + ";";
        Log = Log + Matches[MainMatch].Log;
        // Display Match information to the user.
        lblPictureTime.Text = TotalTime.Seconds.ToString() + " s "
             + TotalTime.Milliseconds.ToString() + " ms";
             picStudy.Image = bitmap;
        picStudy.SizeMode = PictureBoxSizeMode.StretchImage;
        System.Windows.Forms.Cursor.Current =
                System.Windows.Forms.Cursors.Arrow;
        lblPictureInfo.Text = HighestMatch.ToString() + "% match
                found on " + LoadedModels[MainMatch].Name;
}
```

```
                else
                {
                        //  No Match.
                        //  Add to Log File line.
                        Log = Log + "N;" + TotalTime.Milliseconds.ToString() + ";" +
                                "Insufficient Match Criteria (" +
                                HighestMatch.ToString() + "% match found on
                                " + LoadedModels[MainMatch].Name + ");" +
                                gotPerimeter.ToString() + ";" +
                                gotArea.ToString() + ";" +
                                gotXDistance.ToString() + ";" +
                                gotYDistance.ToString() + ";";
                        if (gotEdges == 0)
                                Log = Log +  " ;";
                        else
                                Log = Log + gotEdges.ToString() + ";";
                        Log = Log + Matches[MainMatch].Log;

                        //  Display Match information to the user.
                        System.Windows.Forms.Cursor.Current =
                                System.Windows.Forms.Cursors.Arrow;
                        lblPictureInfo.Text = "Insufficient Match Criteria (" +
                                HighestMatch.ToString() + "% Match)";
                }
        }
        else
        {
                Log = Log + "N; ; ;Object couldn't be found; ; ; ; ; ; ; ; ; ;";
                lblPictureInfo.Text = "Object couldn't be found";
        }
        //  Set up the StillImage toolbar buttons.
        tbStillImage.Buttons[0].Enabled = true;
        tbStillImage.Buttons[1].Enabled = true;
        tbStillImage.Buttons[2].Enabled = true;
        //  Make sure the user can't click on picStudy.
        picStudy.Enabled = false;
        //  Write the Log File line.
        LogWriter.WriteLine(Log);
       }
    }
}
```

1.9.17 Match Models to Measurements

```
//  This subroutine determines whether the first 4 fields of the first
//  Model are a match and correspondingly adds to the Log File line.
private void MatchModelOne()
{
        Matches[0].Log = "";
        if ((gotPerimeter >= LoadedModels[0].PerimeterLow) && (gotPerimeter <=
            LoadedModels[0].PerimeterHigh))
        {
                MatchPercentage1 = MatchPercentage1 + 20;
                Matches[0].Perimeter = true;
                Matches[0].Log = Matches[0].Log + "Y;";
        }
        else
        {
                Matches[0].Perimeter = false;
                Matches[0].Log = Matches[0].Log + "N;";
        }
        if ((gotArea >= LoadedModels[0].AreaLow) && (gotArea <=
             LoadedModels[0].AreaHigh))
        {
                MatchPercentage1 = MatchPercentage1 + 20;
                Matches[0].Area = true;
                Matches[0].Log = Matches[0].Log + "Y;";
        }
        else
        {
                Matches[0].Area = false;
                Matches[0].Log = Matches[0].Log + "N;";
        }
        if ((gotXDistance >= LoadedModels[0].XDistanceLow) && (gotXDistance
              <= LoadedModels[0].XDistanceHigh))
        {
                MatchPercentage1 = MatchPercentage1 + 20;
                Matches[0].XDistance = true;
                Matches[0].Log = Matches[0].Log + "Y;";
        }
        else
        {
                Matches[0].XDistance = false;
                Matches[0].Log = Matches[0].Log + "N;";
        }
        if ((gotYDistance >= LoadedModels[0].YDistanceLow) && (gotYDistance
            <= LoadedModels[0].YDistanceHigh))
        {
```

```
                MatchPercentage1 = MatchPercentage1 + 20;
                Matches[0].YDistance = true;
                Matches[0].Log = Matches[0].Log + "Y;";
        }
        else
        {
                Matches[0].YDistance = false;
                Matches[0].Log = Matches[0].Log + "N;";
        }
}


//  This subroutine determines whether the first 4 fields of the second
//  Model are a match and correspondingly adds to the Log File line.
private void MatchModelTwo()
{
        Matches[1].Log = "";
        if ((gotPerimeter >= LoadedModels[1].PerimeterLow) && (gotPerimeter <=
            LoadedModels[1].PerimeterHigh))
        {
                MatchPercentage2 = MatchPercentage2 + 20;
                Matches[1].Perimeter = true;
                Matches[1].Log = Matches[1].Log + "Y;";
        }
        else
        {
                Matches[1].Perimeter = false;
                Matches[1].Log = Matches[1].Log + "N;";
        }
        if ((gotArea >= LoadedModels[1].AreaLow) && (gotArea <=
            LoadedModels[1].AreaHigh))
        {
                MatchPercentage2 = MatchPercentage2 + 20;
                Matches[1].Area = true;
                Matches[1].Log = Matches[1].Log + "Y;";
        }
        else
        {
                Matches[1].Area = false;
                Matches[1].Log = Matches[1].Log + "N;";
        }
        if ((gotXDistance >= LoadedModels[1].XDistanceLow) && (gotXDistance
              <= LoadedModels[1].XDistanceHigh))
        {
                MatchPercentage2 = MatchPercentage2 + 20;
                Matches[1].XDistance = true;
```

```
                Matches[1].Log = Matches[1].Log + "Y;";
        }
        else
        {
                Matches[1].XDistance = false;
                Matches[1].Log = Matches[1].Log + "N;";
        }
        if ((gotYDistance >= LoadedModels[1].YDistanceLow) && (gotYDistance
            <= LoadedModels[1].YDistanceHigh))
        {
                MatchPercentage2 = MatchPercentage2 + 20;
                Matches[1].YDistance = true;
                Matches[1].Log = Matches[1].Log + "Y;";
        }
        else
        {
                Matches[1].YDistance = false;
                Matches[1].Log = Matches[1].Log + "N;";
        }
}

//  This subroutine determines whether the first 4 fields of the third
//  Model are a match and correspondingly adds to the Log File line.
private void MatchModelThree()
{
        Matches[2].Log = "";
        if ((gotPerimeter >= LoadedModels[2].PerimeterLow) && (gotPerimeter <=
            LoadedModels[2].PerimeterHigh))
        {
                MatchPercentage3 = MatchPercentage3 + 20;
                Matches[2].Perimeter = true;
                Matches[2].Log = Matches[2].Log + "Y;";
        }
        else
        {
                Matches[2].Perimeter = false;
                Matches[2].Log = Matches[2].Log + "N;";
        }
        if ((gotArea >= LoadedModels[2].AreaLow) && (gotArea <=
            LoadedModels[2].AreaHigh))
        {
                MatchPercentage3 = MatchPercentage3 + 20;
                Matches[2].Area = true;
                Matches[2].Log = Matches[2].Log + "Y;";
        }
        else
```

```
        {
                Matches[2].Area = false;
                Matches[2].Log = Matches[2].Log + "N;";
        }
        if ((gotXDistance >= LoadedModels[2].XDistanceLow) && (gotXDistance
            <= LoadedModels[2].XDistanceHigh))
        {
                MatchPercentage3 = MatchPercentage3 + 20;
                Matches[2].XDistance = true;
                Matches[2].Log = Matches[2].Log + "Y;";
        }
        else
        {
                Matches[2].XDistance = false;
                Matches[2].Log = Matches[2].Log + "N;";
        }
        if ((gotYDistance >= LoadedModels[2].YDistanceLow) &&
           (gotYDistance <= LoadedModels[2].YDistanceHigh))
        {
                MatchPercentage3 = MatchPercentage3 + 20;
                Matches[2].YDistance = true;
                Matches[2].Log = Matches[2].Log + "Y;";
        }
        else
        {
                Matches[2].YDistance = false;
                Matches[2].Log = Matches[2].Log + "N;";
        }
}

//  This subroutine determines whether the first 4 fields of the fourth
//  Model are a match and correspondingly adds to the Log File line.
private void MatchModelFour()
{
        Matches[3].Log = "";
        if ((gotPerimeter >= LoadedModels[3].PerimeterLow) && (gotPerimeter <=
            LoadedModels[3].PerimeterHigh))
        {
                MatchPercentage4 = MatchPercentage4 + 20;
                Matches[3].Perimeter = true;
                Matches[3].Log = Matches[3].Log + "Y;";
        }
        else
        {
                Matches[3].Perimeter = false;
                Matches[3].Log = Matches[3].Log + "N;";
```

```
        }
        if ((gotArea >= LoadedModels[3].AreaLow) && (gotArea <=
           LoadedModels[3].AreaHigh))
        {
                MatchPercentage4 = MatchPercentage4 + 20;
                Matches[3].Area = true;
                Matches[3].Log = Matches[3].Log + "Y;";
        }
        else
        {
                Matches[3].Area = false;
                Matches[3].Log = Matches[3].Log + "N;";
        }
        if ((gotXDistance >= LoadedModels[3].XDistanceLow) && (gotXDistance
            <= LoadedModels[3].XDistanceHigh))
        {
                MatchPercentage4 = MatchPercentage4 + 20;
                Matches[3].XDistance = true;
                Matches[3].Log = Matches[3].Log + "Y;";
        }
        else
        {
                Matches[3].XDistance = false;
                Matches[3].Log = Matches[3].Log + "N;";
        }
        if ((gotYDistance >= LoadedModels[3].YDistanceLow) && (gotYDistance
            <= LoadedModels[3].YDistanceHigh))
        {
                MatchPercentage4 = MatchPercentage4 + 20;
                Matches[3].YDistance = true;
                Matches[3].Log = Matches[3].Log + "Y;";
        }
        else
        {
                Matches[3].YDistance = false;
                Matches[3].Log = Matches[3].Log + "N;";
        }
}
```

1.9.18 Adjust Contrast

```
//  This subroutine is called when the 'Adjust Brightness' button is clicked on
//  frmUpdateModel's toolbar.
private void AdjustBrightness()
{
        //  Calls frmBrightness and opens it as a dialog.
        frmBrightness Brightness = new frmBrightness(bitmap,picStudy);
        Brightness.ShowDialog();
}
```

1.9.19 Adjust Contrast

```
//  This subroutine is called when the 'Adjust Contrast' button is clicked on
//  frmUpdateModel's toolbar.
private void AdjustContrast()
{
        //  Calls frmContrast and opens it as a dialog.
        frmContrast Contrast = new frmContrast(bitmap,picStudy);
        Contrast.ShowDialog();
}
```

1.9.20 Video Play/Pause Button

```
//  This subroutine is called when the PlayPause button is clicked.
private void cmdPlayPause_Click(object sender, System.EventArgs e)
{
        //  Make sure the Video timer is enabled.
        tmVideo.Enabled = true;
        //  If the video is playing, pause it.  If it's paused, play it.
        if (PlayPause == 1)
        {
                PlayPause = 2;
                CurrentVideo.Pause();
                cmdPlayPause.ImageIndex = 0;

        }
       else
        if (PlayPause == 2)
        {
                PlayPause = 1;
                CurrentVideo.Play();
                cmdPlayPause.ImageIndex = 1;
                cmdStop.Enabled = true;
        }
}
```

1.9.21 Video Stop Button

```
//  This subroutine is called when the Stop button is clicked.
private void cmdStop_Click(object sender, System.EventArgs e)
{
        //  Stop the video and reset the video controls.
        tmVideo.Enabled = false;
        CurrentVideo.Stop();
        cmdPlayPause.ImageIndex = 0;
        PlayPause = 2;
        lblVidCurrent.Text = "00:00:00";
        tbVideo.Value = 0;
        //  Hide the match pictureboxes and reset the recognition time
        //  label.
        picMatchPerimeter.Visible = false;
        picMatchArea.Visible = false;
        picMatchX.Visible = false;
        picMatchY.Visible = false;
        picMatchEdges.Visible = false;
        lblTime.Text = "";
        //  Set up the StillImage toolbar's buttons.
        tbStillImage.Buttons[0].Enabled = true;
        tbStillImage.Buttons[1].Enabled = true;
        tbStillImage.Buttons[2].Enabled = true;
}
```

1.9.22 Video timer

```
//  This subroutine is called whenever the Video timer expires (100).
private void tmVideo_Tick(object sender, System.EventArgs e)
{
        //  Variables used to calculate the current video runtime
        //  position.
        int hours, minutes, seconds;
        seconds = (int)CurrentVideo.CurrentPosition % 60;
        minutes = (int)CurrentVideo.CurrentPosition / 60 % 60;
        hours = (int)CurrentVideo.CurrentPosition /60 / 60;
        if (hours < 10)
                lblVidCurrent.Text = "0" + hours.ToString() + ":";
        else
                lblVidCurrent.Text = hours.ToString() + ":";
        if (minutes < 10)
                lblVidCurrent.Text = lblVidCurrent.Text + "0" + minutes.ToString() +
                ":";
        else
                lblVidCurrent.Text = lblVidCurrent.Text + minutes.ToString() + ":";
```

```
        if (seconds < 10)
                lblVidCurrent.Text = lblVidCurrent.Text + "0" + seconds.ToString();
        else
                lblVidCurrent.Text = lblVidCurrent.Text + seconds.ToString();
         //  Assign the current runtime value to the Video Position slider.
        tbVideo.Value = (int) CurrentVideo.CurrentPosition;
        //  Determine whether the video must be looped or not.
        if (chkLoop.Checked == true)
        {
                //  Loop it.
                if (lblVidCurrent.Text == lblVidTotal.Text)
                {
                        CurrentVideo.CurrentPosition = 0;
                        tbVideo.Value = 0;
                }
        }
        else
                //  Don't loop it.
                if (lblVidCurrent.Text == lblVidTotal.Text)
                {
                   //  Reset the video controls.
                        PlayPause = 2;
                        cmdPlayPause.ImageIndex = 0;
                        tmVideo.Enabled = false;
                        CurrentVideo.CurrentPosition = 0;
                        tbVideo.Value = 0;
                       CurrentVideo.Stop();
                        lblVidCurrent.Text = "00:00:00";
                }
}
```

1.9.23 Video Position scrollbar

```
//  This subroutine is called when the Video Position scrollbar's
//  value changes.
private void tbVideo_Scroll(object sender, System.EventArgs e)
{
        //  Sets the video's current position to that
        //  of the slider.
        CurrentVideo.CurrentPosition = tbVideo.Value;
        //  Variables used to calculate the current video runtime
        //  position.
        int hours, minutes, seconds;
        seconds = (int)CurrentVideo.CurrentPosition % 60;
        minutes = (int)CurrentVideo.CurrentPosition / 60 % 60;
        hours = (int)CurrentVideo.CurrentPosition /60 / 60;
```

```
        if (hours < 10)
                lblVidCurrent.Text = "0" + hours.ToString() + ":";
        else
                lblVidCurrent.Text = hours.ToString() + ":";
        if (minutes < 10)
                lblVidCurrent.Text = lblVidCurrent.Text + "0" + minutes.ToString() +
                                        ":";
        else
                lblVidCurrent.Text = lblVidCurrent.Text + minutes.ToString() + ":";
        if (seconds < 10)
                lblVidCurrent.Text = lblVidCurrent.Text + "0" + seconds.ToString();
        else
                lblVidCurrent.Text = lblVidCurrent.Text + seconds.ToString();
}
```

1.9.24 Video Volume scrollbar

```
//  This subroutine is called when the Video Volume slider's value
//  changes.  It assigns the slider's value to the video's volume
//  setting.
private void tbVidVolume_Scroll(object sender, System.EventArgs e)
{
   CurrentVideo.Audio.Volume = tbVidVolume.Value;
}
```

1.9.25 Mute checkbox

```
//  This subroutine is called when the Mute checkbox is clicked.
private void chkMute_CheckedChanged(object sender, System.EventArgs e)
{
        //  Mute the volume.
        if (chkMute.Checked == true)
        {
                CurrentVideo.Audio.Volume = -10000;
                tbVidVolume.Enabled = false;
        }
       //  Restore the volume.
        if (chkMute.Checked == false)
        {
                CurrentVideo.Audio.Volume = tbVidVolume.Value;
                tbVidVolume.Enabled = true;
        }
}
```

1.9.26 Select Camera

```
//  This subroutine is called whenever a capture device needs to be
//  selected.
private void CameraMode()
{
        //  Tries to load a capture device.  If an error is found, an appropriate
        //  message is shown to the user.
        try
        {
                //  Create a new instance of a WiaClass object.
                wiaManager = new WiaClass();
                //  Create a new Select Device Dialog Box.
                WIA.CommonDialogClass class1 = new
                        WIA.CommonDialogClass();
                //  If any other device is loaded, destroy its reference.
                if (CurrentCam != null)
                        wiaVideo.DestroyVideo();
                //  Show the Select Device Dialog Box and assigns the selected
                //  device to CurrentCam.
                CurrentCam = class1.ShowSelectDevice
                                (WIA.WiaDeviceType.UnspecifiedDeviceType,
                                 true, false);
                //  If a device was selected.
                if (CurrentCam != null)
                {
                        //  Assigns the selected device's name to the DeviceID
                        //  string.
                        string DeviceID = CurrentCam.DeviceID.ToString();
                        //  Initialize variables
                        object foundID = null;
                        CollectionClass wiaDevs = null;
                        DeviceInfoClass devInfo = null;
                        int Found = 0;
                        //  Assigns all of the current WIA devices in the system to
                        //  wiaDevs.
                        wiaDevs = wiaManager.Devices as CollectionClass;
                        //  If there were devices
                        if( wiaDevs != null )
                        {
                                //  Run thrugh all of the devices.
                                foreach( object wiaObj in wiaDevs )
                                {
                                        //  Stop the loop if the selected device was
                                        //  found.
                                        if (Found == 1)
```

```
                    break;
                // Wrap the wiaObj in a DeviceInfoClass
                // wrapper.
                devInfo = (DeviceInfoClass)
                        Marshal.CreateWrapperOfType
                        (wiaObj, typeof(DeviceInfoClass) );
                // Test whether the current wiaObj matches
                // the selected device.
                if (devInfo.Id.ToString() == DeviceID)
                {
                    // Used to stop the loop.
                    Found = 1;
                    // Assigns the object's ID to foundID
                    // and releases any resources used by
                    // the wiaObj and devInfo.
                    foundID = devInfo.Id;
                    Marshal.ReleaseComObject( wiaObj );
                    Marshal.ReleaseComObject( devInfo );
                }
            }
        }

        // Reinitialize variables
        devInfo = null;
        foundID = System.Reflection.Missing.Value;
        // Creates a video capture object using the selected device.
        wiaCamera = (ItemClass) wiaManager.Create( ref foundID );
        // May sometimes take a while, so inform the user that the
        // system is loading.
        Cursor.Current = Cursors.WaitCursor;
        // Try to load the video stream, if it fails a relevant message
        // is shown to the user.
        try
        {
            // Create a new instance of a WiaVideoClass object.
            wiaVideo = new WiaVideoClass();
            // MSDN specifies that the ImagesDirectory should
            // be set to the WIA_DPV_IMAGES_DIRECTORY.
            // This is specified as 3587.
            string videoDir = (string) wiaCamera.GetPropById(
                        (WiaItemPropertyId) 3587 );
            wiaVideo.ImagesDirectory = videoDir;
            // Assigns the ID of the selected device to the
            // cameraID string.
            string cameraID = wiaCamera.GetPropById(
                (WiaItemPropertyId) WiaDeviceInfoPropertyId.
```

```
                DeviceInfoDevId ) as string;
            //  Accesses system pointers (handles) so the code
            //  needs to be declared unsafe in order
            //  for the compiler to process it.
            unsafe
            {
                //  In order to create the video display,
                //  wiaVideo needs the handle of the
                //  componenent on which the video needs to
                //  be displayed.  This reference needs to be in
                //  the format of a
                //  WIAVideoLib._RemotableHandle,
                //  so the the Handle of pnlCam needs to be
                //  cast to this format.
                WIAVIDEOLib._RemotableHandle *handle =
                    (WIAVIDEOLib._RemotableHandle*)
                    picCam.Handle.ToPointer();
                wiaVideo.CreateVideoByWiaDevID(cameraID,
                    ref *handle, 1, 0);
            }
        }
        catch( Exception )
        {
            //  If a wiaVideo object was loaded, release it.
            if( wiaVideo != null )
                Marshal.ReleaseComObject( wiaVideo );
            //  Reinitialize the variable.
            wiaVideo = null;
            //  Display an appropriate message to the user.
            MessageBox.Show( this, "Create video stream failed",
                                "WIA", MessageBoxButtons.OK,
                                MessageBoxIcon.Stop );

        }
        finally
        {
            //  If everything was successful, do the following.
            try
            {
                // Creates the agent communication directory.
                if (Directory.Exists(Application.StartupPath +
                   "\\Recognition") == true)
                   Directory.Delete(Application.StartupPath +
                        "\\Recognition",true);
                Directory.CreateDirectory
                    (Application.StartupPath +
```

```
                                        "\\Recognition");
                                    //  Play the camera stream and set the camera
                                    //  controls.
                                    wiaVideo.Play();
                                    Cursor.Current = Cursors.Default;
                                    cmdCamPlayPause.Visible = true;
                                    cmdCamPlayPause.Enabled = true;
                                    cmdCamPlayPause.ImageIndex = 1;
                                    picCam.Visible = true;
                                    //  Add the Cam/Video tabpage and set its
                                   //  properties.
                                    tabCamVideo.ImageIndex = 2;
                                    tabRecognition.TabPages.Add(tabCamVideo);
                                    tabCamVideo.Text = "Camera Mode";
                                }
                                catch( Exception ) {}
                            }
                        }
                }
                catch( Exception )
                {
                        MessageBox.Show( this, "Camera connection failed.", "Message",
                                MessageBoxButtons.OK, MessageBoxIcon.Stop );
                }
}
```

1.9.27 Camera PlayPause Button

```
//  This subroutine is called when the CamPlayPause button is clicked.
private void cmdCamPlayPause_Click(object sender, System.EventArgs e)
{
        //  If the stream is played, pause it.  If it's paused
        //  play it.
        if (cmdCamPlayPause.ImageIndex == 0)
        {
                try
                {
                        wiaVideo.Play();
                        cmdCamPlayPause.ImageIndex = 1;
                }
                catch( Exception ) {}
        }
        else
        {
                try
                {
```

```
                wiaVideo.Pause();
                cmdCamPlayPause.ImageIndex = 0;
            }
            catch( Exception ) {}
        }
}
```

1.9.28 RecognitionPicAdjustments toolbar

```
//  This subroutine acts as a switchboard to determine which button was
//  clicked on the RecognitionPicAdjustment toolbar.
private void tbRecognitionPicAdjustments_ButtonClick(object sender,
            System.Windows.Forms.ToolBarButtonClickEventArgs e)
{
        switch(tbRecognitionPicAdjustments.Buttons.IndexOf(e.Button))
        {
            //  Adjust the Brightness.
            case 0:  AdjustBrightness();
                break;
            //  Adjust the Contrast.
            case 1:  AdjustContrast();
                break;
        }
}
```

1.9.29 PictureProcess Button

```
//  This subroutine is called when the PictureProcess button is clicked.
private void cmdPictureProcess_Click(object sender, System.EventArgs e)
{
        //  Disable the toolbar buttons.
        tbRecognitionPicAdjustments.Buttons[0].Enabled = false;
        tbRecognitionPicAdjustments.Buttons[1].Enabled = false;
        //  Disable the following buttons.
        gbProcessType.Enabled = false;
        cmdPictureProcess.Enabled = false;
        picStudy.Enabled = true;
        //  Set up for Whole Image Mode.
        if (optWhole.Checked == true)
           StillRecognition();
        //  Set up for Area of Image Mode.
        if (optArea.Checked == true)
        {
            tbStillImage.Buttons[0].Enabled = false;
            tbStillImage.Buttons[1].Enabled = false;
            tbStillImage.Buttons[2].Enabled = false;
```

```
                lblPictureInfo.Text = "Please draw a selection rectangle around the
                                        object.";
                clickcounter = 0;
        }
}
```

1.9.30 CamVideoType toolbar

```
//  This subroutine acts as a switchboard to determine which button was
//  clicked on the CamVideoType toolbar.
private void tbCamVideoType_ButtonClick(object sender,
                System.Windows.Forms.ToolBarButtonClickEventArgs e)
{
        //  Make the following buttons visible.
        cmdActivate.Visible = true;
        picCapturePreview.Visible = true;
        gbMatches.Visible = true;
        switch(tbCamVideoType.Buttons.IndexOf(e.Button))
        {
                //  Single Frame Mode.
                case 0: tbCamVideoType.Buttons[0].Pushed = true;
                        tbCamVideoType.Buttons[1].Pushed = false;
                        lblInfo.Text = "Single Image Mode";
                        cmdActivate.Text = "Capture and Process";
                        break;
               //  Continuous Mode.
                case 1: tbCamVideoType.Buttons[0].Pushed = false;
                        tbCamVideoType.Buttons[1].Pushed = true;
                        lblInfo.Text = "Continuous Mode";
                        cmdActivate.Text = "Activate";
                        FlagActivate = 0;
                        break;
        }
}
```

1.9.31 Activate Button

```
//  This subroutine is called when the Activate button is clicked.
private void cmdActivate_Click(object sender, System.EventArgs e)
{
        //  Capture the desktop image.
        int hdcSrc = User32.GetWindowDC(User32.GetDesktopWindow()),
                hdcDest = GDI32.CreateCompatibleDC(hdcSrc),
                hBitmap = GDI32.CreateCompatibleBitmap(hdcSrc,
        GDI32.GetDeviceCaps(hdcSrc,8),GDI32.GetDeviceCaps(hdcSrc,10));
                GDI32.SelectObject(hdcDest,hBitmap);
```

```
        GDI32.BitBlt(hdcDest,0,0,GDI32.GetDeviceCaps(hdcSrc,8),
        GDI32.GetDeviceCaps(hdcSrc,10),
        hdcSrc,0,0,0x00CC0020);
// Create the bitmap image and determine which section of
// the image should be used.
Bitmap image =
        new Bitmap(Image.FromHbitmap(new IntPtr(hBitmap)),
        Image.FromHbitmap(new IntPtr(hBitmap)).Width,
        Image.FromHbitmap(new IntPtr(hBitmap)).Height);
RectangleF a = new RectangleF(7,158,640,480);
bitmap = (new Bitmap((Bitmap)
            image.Clone(a,PixelFormat.Format24bppRgb))
// Single Frame Mode.
if (tbCamVideoType.Buttons[0].Pushed == true)
{
        // Creates the shared Whiteboard object. Then initializes all of the
        // agents and loads them onto individual threads.
        whiteboard = new Whiteboard(bitmap);
        MEDagent = new  MainEdgeDetectionAgent(640,480,whiteboard);
        MEDthread = new Thread(new ThreadStart(MEDagent.StartWait));
        MEDthread.Start();
        ICagent = new InvertColoursAgent(640,480,whiteboard);
        ICthread = new Thread(new ThreadStart(ICagent.StartWait));
        ICthread.Start();
       TBagent = new ToBinaryAgent(640,480,whiteboard);
       TBthread = new Thread(new ThreadStart(TBagent.StartWait));
       TBthread.Start();
        Pagent = new PerimeterAgent(whiteboard,5,5,635,475,2);
        Pthread = new Thread(new ThreadStart(Pagent.StartWait));
        Pthread.Start();
        EGagent = new EdgeGraphAgent(whiteboard,5,435,5,635);
        EGthread = new Thread(new ThreadStart(EGagent.StartWait));
        EGthread.Start();
        // Tells the the first agent to start processing.
        Whiteboard.Increment()
        StillRecognitionForCamVideo();
        // Ends the agents' waiting process and stops the agents.
        whiteboard.RunStartWait = 1;
        MEDthread.Join();
        ICthread.Join();
        TBthread.Join();
        Pthread.Join();
        EGthread.Join();
}
```

```
//  Continuous Mode.
if (tbCamVideoType.Buttons[1].Pushed == true)
{
        if (FlagActivate == 0)
        {
                tbCamVideoType.Buttons[0].Enabled = false;
                tbCamVideoType.Buttons[0].Enabled = false;
                FlagActivate = 1;
                cmdActivate.Text = "Deactivate";
                //  Creates the shared Whiteboard object. Then initializes all
                //  of the agents and loads them onto individual threads.
                whiteboard = new Whiteboard(bitmap);
                MEDagent = new
                                MainEdgeDetectionAgent(640,480,whiteboard);
                MEDthread = new Thread(new
                                        ThreadStart(MEDagent.StartWait));
                MEDthread.Start();
                ICagent = new InvertColoursAgent(640,480,whiteboard);
                ICthread = new Thread(new
                                ThreadStart(ICagent.StartWait));
                ICthread.Start();
                TBagent = new ToBinaryAgent(640,480,whiteboard);
                TBthread = new Thread(new
                                ThreadStart(TBagent.StartWait));
                TBthread.Start();
                Pagent = new PerimeterAgent(whiteboard,5,5,635,475,2);
                Pthread = new Thread(new ThreadStart(Pagent.StartWait));
                Pthread.Start();
                EGagent = new EdgeGraphAgent(whiteboard,5,435,5,635);
                EGthread = new Thread(new
                                ThreadStart(EGagent.StartWait));
                EGthread.Start();
                //  Tells the the first agent to start processing.
                Whiteboard.Increment()
                tmContinuousMode.Enabled = true;
        }
        else
        {
                tmContinuousMode.Enabled = false;
                //  Ends the agents' waiting process and stops the agents.
                whiteboard.RunStartWait = 1;
                MEDthread.Join();
                ICthread.Join();
                TBthread.Join();
```

```
                    Pthread.Join();
                    EGthread.Join();
                    tbCamVideoType.Buttons[0].Enabled = true;
                    tbCamVideoType.Buttons[0].Enabled = true;
                    FlagActivate = 0;
                    cmdActivate.Text = "Activate";
                }
            }
}
```

1.9.32 Perform Recogniton on Frame

```
//  This subroutine is called to capture a frame from a video
//  or device source and then perform recognition on it.
private void StillRecognitionForCamVideo()
{
        //  Variables used to determine the bounding box.
        int topx, topy, bottomx, bottomy;
        //  Variables used to determine which Model has the best match.
        int MainMatch, HighestMatch;
        //  Initialize the Log File line.
        string Log = "";
        //  Add to the Log File line.
        if (tabCamVideo.Text == "Camera Mode")
          Log = "Camera;";
        else
          Log = "Video;";
        if (cmdActivate.Text == "Capture and Process")
          Log = Log + "Single Image;" + filename + ";" +
          DateTime.Now.ToShortDateString() + ";" +
          DateTime.Now.ToLongTimeString() + ";";
        else
          Log = Log + "Continuous;" + filename + ";" +
          DateTime.Now.ToShortDateString() + ";" +
          DateTime.Now.ToLongTimeString() + ";";
        //  Set start time for recognition process.
        StartTime = System.DateTime.Now;
        //  Clear all of the labels and pictureboxes.
        lblPerimeter.Text = null;
        lblArea.Text = null;
        lblXDistance.Text = null;
        lblYDistance.Text = null;
        lblEdges.Text = null;
        lblTime.Text = null;
        picMatchPerimeter.Visible = false;
        picMatchArea.Visible = false;
```

```
picMatchX.Visible = false;
picMatchY.Visible = false;
picMatchEdges.Visible = false;
picCapturePreview.Image = null;
picCapturePreview.BackColor = Color.FromName("Control");
//  Show the user that the program is processing.
System.Windows.Forms.Cursor.Current =
        System.Windows.Forms.Cursors.WaitCursor;
       //  Halts all of the threads when they have finished processing.
while (whiteboard.RunStartWait == 0)
{
       if (whiteboard.Counter > 4)
       {
          //  Display the found object's perimeter.
         picPicturePreview.BackColor = Color.White;
         picPicturePreview.Image = whiteboard.picInUseDone;
         picPicturePreview.SizeMode = PictureBoxSizeMode.StretchImage;
         //  Get the perimeter value and determine if its higher than the
         //  minimum value.
        gotPerimeter = Pagent.getPerimeter();
        if (gotPerimeter >= MatchPerimeter)
        {

        //  Match Found.
        //  Add to the Log File line.
         Log = Log + "Y;";
         //  Display the found perimeter image to the picture box
         picCapturePreview.Image = Pagent.getpicInUse();
         picCapturePreview.SizeMode =
                             PictureBoxSizeMode.StretchImage;
         picCapturePreview.BackColor = Color.White;
         //  Set the bounding box.
         topx = Pagent.getLowestX();
         topy = Pagent.getLowestY();
         bottomx = Pagent.getHighestX();
         bottomy = Pagent.getHighestY();
         //  Assign the calculated values to the global variables.
         gotPerimeter = Pagent.getPerimeter();
         gotArea = Pagent.getArea();
         gotXDistance = Pagent.getXDistance();
         gotYDistance = Pagent.getYDistance();
         //  Assign the calculated values to the labels.
         lblPerimeter.Text = gotPerimeter.ToString();
         lblArea.Text = gotArea.ToString();
         lblXDistance.Text = gotXDistance.ToString();
         lblYDistance.Text = gotYDistance.ToString();
```

```
lblEdges.Text = "Not Tested";
//  Pause video / device if Pause On Object was selected.
if (optObject.Checked == true)
{
        lblInfo.Text = "Paused for Object Found";
        if (cmdPlayPause.Visible == true)
        {
                PlayPause = 2;
                CurrentVideo.Pause();
                cmdPlayPause.ImageIndex = 0;
        }
        if (cmdCamPlayPause.Visible == true)
        {
                try
                {
                        wiaVideo.Pause();
                        cmdCamPlayPause.ImageIndex = 0;
                }
                catch( Exception ) {}
        }
}
//  Initialize variables to determine which Model is the best match.
HighestMatch = 0;
MatchPercentage1 = MatchPercentage2 = MatchPercentage3 =
                        MatchPercentage4 = 0;
//  See if there's a match on the first Model.
MatchModelOne();
MainMatch = 0;
HighestMatch = MatchPercentage1;
//  See if there's a higher match on the second Model.
if ((HighestMatch != 80) && (LoadedModels.Length == 2))
{
        MatchModelTwo();
        if (MatchPercentage2 > HighestMatch)
        {
                MainMatch = 1;
                HighestMatch = MatchPercentage2;
        }
}
//  See if there's a higher match on the third Model.
if ((HighestMatch != 80) && (LoadedModels.Length == 3))
{
        MatchModelThree();
        if (MatchPercentage3 > HighestMatch)
        {
                MainMatch = 2;
```

```
                HighestMatch = MatchPercentage3;
            }
        }
        //  See if there's a higher match on the fourth Model.
        if ((HighestMatch != 80) && (LoadedModels.Length == 4))
        {
            MatchModelFour();
            if (MatchPercentage4 > HighestMatch)
            {
                MainMatch = 3;
                HighestMatch = MatchPercentage4;
            }
        }
        //  Set the match pictures according to the best matched Model.
        picMatchPerimeter.Visible = Matches[MainMatch].Perimeter;
        picMatchArea.Visible = Matches[MainMatch].Area;
        picMatchX.Visible = Matches[MainMatch].XDistance;
        picMatchY.Visible = Matches[MainMatch].YDistance;
        //  If the highest match was 60, see if the EdgeGraphs match.
        if (HighestMatch == 60)
        {
            whiteboard.Increment();
            int check = 0;
            while (check == 0)
            {
                If (whiteboard.Counter == 7)
                {
                    check = 1;
                    gotEdges = EGagent.getEdgeCount();
                    If ((gotEdges >=
                      LoadedModels[MainMatch].EdgesLow) &&
                     (gotEdges <=
                     LoadedModels[MainMatch].EdgesHigh))
                     {
                         HighestMatch = HighestMatch + 20;
                         picMatchEdges.Visible = true;
                         lblEdges.Text = gotEdges.ToString();
                         Matches[MainMatch].Log =
                             Matches[MainMatch].Log + "Y;";
                     }
                    else
                         Matches[MainMatch].Log =
                             Matches[MainMatch].Log + "N;";
            }
        }
        else
```

```
            Matches[MainMatch].Log = Matches[MainMatch].Log + " ;";
        //  Set end time for recognition process.
        TotalTime = System.DateTime.Now.Subtract(StartTime);
        //  Refresh picCam.
        picCam.Refresh();
        //  Determine if a Match was found.
        if (HighestMatch >= 80)
        {
                //  Match found.
                //  Add to the Log File line.
                Log = Log + "Y;" + TotalTime.Milliseconds.ToString() + ";" +
                        HighestMatch.ToString() + "% match found on " +
                        LoadedModels[MainMatch].Name + ";" +
                        gotPerimeter.ToString() + ";" + gotArea.ToString() +
                        ";" + gotXDistance.ToString() + ";" +
                        gotYDistance.ToString() + ";";

                if (gotEdges == 0)
                        Log = Log +  " ;";
                else
                        Log = Log + gotEdges.ToString() + ";";
                Log = Log + Matches[MainMatch].Log;
                //  Display match findings to the user.
                lblTime.Text = TotalTime.Seconds.ToString() + " s " +
                                TotalTime.Milliseconds.ToString() + " ms";
                  if (optObject.Checked == false)
                    lblInfo.Text = HighestMatch.ToString() + "% match found
                                on " + LoadedModels[MainMatch].Name;
                 //  Determine the current object position.
                CurrentConveyerX = (topx + ((bottomx - topx) / 2)) *
                                (XMeasurement / 640);
                CurrentConveyerY = (topy + ((bottomy - topy) / 2)) *
                                (YMeasurement / 480);
                if (Direction == 0)
                  CurrentConveyerY = (CurrentConveyerY) - (Speed *
                                TotalTime.Milliseconds / 1000);
                if (Direction == 1)
                  CurrentConveyerY = (CurrentConveyerY) + (Speed *
                                TotalTime.Milliseconds / 1000);
                if (Direction == 2)
                  CurrentConveyerX = (CurrentConveyerX) - (Speed *
                                TotalTime.Milliseconds / 1000);
                if (Direction == 3)
                  CurrentConveyerX = (CurrentConveyerX) + (Speed *
                                TotalTime.Milliseconds / 1000);
                 //  Pause video / device if Pause On Match was selected.
```

```
            if (optMatch.Checked == true)
            {
                lblInfo.Text = "Paused for " + HighestMatch.ToString()
                               + "% match found on " +
                               LoadedModels[MainMatch].Name;;
                if (cmdPlayPause.Visible == true)
                {
                    PlayPause = 2;
                    CurrentVideo.Pause();
                    cmdPlayPause.ImageIndex = 0;
                }
                if (cmdCamPlayPause.Visible == true)
                {
                    try
                    {
                        wiaVideo.Pause();
                        cmdCamPlayPause.ImageIndex = 0;
                    }
                    catch( Exception ) {}
                }
            }
        }
        else
        {
            //  No match found.
            //  Add to the Log File Line.
            Log = Log + "N;" + TotalTime.Milliseconds.ToString() + ";" +
                        "Insufficient Match Criteria (" +
                        HighestMatch.ToString() + "% match found on
                        " + LoadedModels[MainMatch].Name + ");" +
                        gotPerimeter.ToString() + ";" +
                        gotArea.ToString() + ";" +
                        gotXDistance.ToString() + ";" +
                        gotYDistance.ToString() + ";";
            if (gotEdges == 0)
                Log = Log +  " ;";
            else
                Log = Log + gotEdges.ToString() + ";";
            Log = Log + Matches[MainMatch].Log;
            //  Display match findings to the user.
            lblTime.Text = TotalTime.Seconds.ToString() + " s " +
                           TotalTime.Milliseconds.ToString() + " ms";
              if (optObject.Checked == false)
                lblInfo.Text = HighestMatch.ToString() + "% match found
                               on " + LoadedModels[MainMatch].Name;
              //  Determine the current object position.
```

```
                CurrentConveyerX = (topx + ((bottomx - topx) / 2)) *
                                   (XMeasurement / 640);
                CurrentConveyerY = (topy + ((bottomy - topy) / 2)) *
                                   (YMeasurement / 480);
                if (Direction == 0)
                  CurrentConveyerY = (CurrentConveyerY) - (Speed *
                                     TotalTime.Milliseconds / 1000);
                if (Direction == 1)
                  CurrentConveyerY = (CurrentConveyerY) + (Speed *
                                     TotalTime.Milliseconds / 1000);
                if (Direction == 2)
                  CurrentConveyerX = (CurrentConveyerX) - (Speed *
                                     TotalTime.Milliseconds / 1000);
                if (Direction == 3)
                  CurrentConveyerX = (CurrentConveyerX) + (Speed *
                                     TotalTime.Milliseconds / 1000);
                 //  Pause video / device if Pause On No Match was
                 //  selected.
                if (optMatch.Checked == true)
                {
                        lblInfo.Text = "Paused for " + HighestMatch.ToString()
                                       + "% match found on " +
                                       LoadedModels[MainMatch].Name;;
                        if (cmdPlayPause.Visible == true)
                        {
                                PlayPause = 2;
                                CurrentVideo.Pause();
                                cmdPlayPause.ImageIndex = 0;
                        }
                        if (cmdCamPlayPause.Visible == true)
                        {
                                try
                                {
                                        wiaVideo.Pause();
                                        cmdCamPlayPause.ImageIndex = 0;
                                }
                                catch( Exception ) {}
                        }
                }
        }
else
{
                //  No object Found
                //  Add to the Log File Line.
                Log = Log + "N; ; ;Object couldn't be found; ; ; ; ; ; ; ; ;
                          ; ;";
```

```
                        //  Display message to the user and reset the
                        //  CapturePreview picturebox.
                            lblInfo.Text = "Object couldn't be found";
                            picCapturePreview.Image = null;
                            picCapturePreview.BackColor =
                                Color.FromName("Control");
                    }
                //  Write the Log File line.
                LogWriter.WriteLine(Log);
            }
        }
}
```

1.9.33 ContinuousMode timer

```
//  This subroutine is called whenever the ContinuousMode timer expires (500).
private void tmContinuousMode_Tick(object sender, System.EventArgs e)
{
        //  Capture a frame and perform recognition.
        StillRecognitionForCamVideo();
        //  Capture the desktop image.
        int hdcSrc = User32.GetWindowDC(User32.GetDesktopWindow()),
                hdcDest = GDI32.CreateCompatibleDC(hdcSrc),
                hBitmap = GDI32.CreateCompatibleBitmap(hdcSrc,
        GDI32.GetDeviceCaps(hdcSrc,8),GDI32.GetDeviceCaps(hdcSrc,10));
                GDI32.SelectObject(hdcDest,hBitmap);
                GDI32.BitBlt(hdcDest,0,0,GDI32.GetDeviceCaps(hdcSrc,8),
                GDI32.GetDeviceCaps(hdcSrc,10),
                hdcSrc,0,0,0x00CC0020);
        //  Create the bitmap image and determine which section of
        //  the image should be used.
        Bitmap image =
                new Bitmap(Image.FromHbitmap(new IntPtr(hBitmap)),
                Image.FromHbitmap(new IntPtr(hBitmap)).Width,
                Image.FromHbitmap(new IntPtr(hBitmap)).Height);
        RectangleF a = new RectangleF(7,158,640,480);
        bitmap = (new Bitmap((Bitmap)
                    image.Clone(a,PixelFormat.Format24bppRgb))
        whiteboard.Restart();
}
```

1.9.34 Context Menu Options

```
//  This subroutine is called when the Show in Own Window option is
//  selected in any of the picture context menu's.
private void mnuPictureShow_Click(object sender, System.EventArgs e)
{
        if (picPicturePreview.Image != null)
        {
                frmPicDisplay1 PicDisplay = new
                        frmPicDisplay1((Bitmap)picPicturePreview.Image);
                PicDisplay.Show();
        }
        else
                lblPictureInfo.Text = "No image to display";

}
```

1.9.35 External Classes

```
//  Image class used for capture.
class GDI32
{
        [DllImport("GDI32.dll")]
        public static extern bool BitBlt(int hdcDest,int nXDest,int nYDest,
                int nWidth,int nHeight,int hdcSrc,
                int nXSrc,int nYSrc,int dwRop);
        [DllImport("GDI32.dll")]
        public static extern int CreateCompatibleBitmap(int hdc,int nWidth,
                int nHeight);
        [DllImport("GDI32.dll")]
        public static extern int CreateCompatibleDC(int hdc);
        [DllImport("GDI32.dll")]
        public static extern bool DeleteDC(int hdc);
        [DllImport("GDI32.dll")]
        public static extern bool DeleteObject(int hObject);
        [DllImport("GDI32.dll")]
        public static extern int GetDeviceCaps(int hdc,int nIndex);
        [DllImport("GDI32.dll")]
        public static extern int SelectObject(int hdc,int hgdiobj);
}

//  Windows class used to get the current desktop window.
class    User32
{
        [DllImport("User32.dll")]
        public static extern int GetDesktopWindow();
```

```
        [DllImport("User32.dll")]
        public static extern int GetWindowDC(int hWnd);
        [DllImport("User32.dll")]
        public static extern int ReleaseDC(int hWnd,int hDC);
}
```

1.9.36 Structures

```
//  Structure used to store Model measurements.
struct Model
{
        public string Name;
        public int PerimeterLow;
        public int PerimeterHigh;
        public int AreaLow;
        public int AreaHigh;
        public int XDistanceLow;
        public int XDistanceHigh;
        public int YDistanceLow;
        public int YDistanceHigh ;
        public int EdgesLow ;
        public int EdgesHigh;
}

//  Structure used to store Model matches.
struct MatchStruct
{
        public bool Perimeter;
        public bool Area;
        public bool XDistance;
        public bool YDistance;
        public string Log;
}
}
```

1.10 Capture Still Images (frmCreateStills)

This form is called whenever the user wants to capture still images from either a video file or a camera stream. The captures can then be saved to permanent storage.

1.10.1 System References

```
//  Decleration of References
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.IO;
using System.Drawing.Imaging;
using System.Runtime.InteropServices;
using System.Threading;
using Microsoft.DirectX.AudioVideoPlayback;
using WIALib;
using WIAVIDEOLib;
```

1.10.2 Form Items and Global Variables

```
namespace AntsNest
{
/// <summary>
/// Summary description for frmCreateStills.
/// </summary>
public class frmCreateStills : System.Windows.Forms.Form
{
        private System.Windows.Forms.ToolBar tbType;
        private System.Windows.Forms.ToolBarButton tbbVideo;
        private System.Windows.Forms.ToolBarButton tbbCamera;
        private System.Windows.Forms.ImageList ilButtons;
        private System.ComponentModel.IContainer components;
        private System.Windows.Forms.TabControl tcStills;
        private System.Windows.Forms.TabPage tabCamVideo;
        private System.Windows.Forms.Button cmdCapture;
        private System.Windows.Forms.Label lblInfo;
        private System.Windows.Forms.GroupBox gbCamera;
        private System.Windows.Forms.Button cmdCamPlayPause;
        private System.Windows.Forms.Button cmdSaveAll;
        private System.Windows.Forms.PictureBox picCapture0;
        private System.Windows.Forms.SaveFileDialog sfdCaptures;
        private System.Windows.Forms.PictureBox picCapture14;
```

```
private System.Windows.Forms.PictureBox picCapture13;
private System.Windows.Forms.PictureBox picCapture12;
private System.Windows.Forms.PictureBox picCapture11;
private System.Windows.Forms.PictureBox picCapture10;
private System.Windows.Forms.PictureBox picCapture9;
private System.Windows.Forms.PictureBox picCapture8;
private System.Windows.Forms.PictureBox picCapture7;
private System.Windows.Forms.PictureBox picCapture6;
private System.Windows.Forms.PictureBox picCapture5;
private System.Windows.Forms.PictureBox picCapture4;
private System.Windows.Forms.PictureBox picCapture3;
private System.Windows.Forms.PictureBox picCapture2;
private System.Windows.Forms.PictureBox picCapture1;
private System.Windows.Forms.Button cmdBurst;
private System.Windows.Forms.ContextMenu cmCapture0;
private System.Windows.Forms.MenuItem miSaveAs0;
private System.Windows.Forms.MenuItem miRemove0;
private System.Windows.Forms.MenuItem miShow0;
private System.Windows.Forms.ContextMenu cmCapture1;
private System.Windows.Forms.MenuItem menuItem1;
private System.Windows.Forms.MenuItem menuItem2;
private System.Windows.Forms.MenuItem menuItem3;
private System.Windows.Forms.MenuItem menuItem4;
private System.Windows.Forms.MenuItem menuItem5;
private System.Windows.Forms.MenuItem menuItem6;
private System.Windows.Forms.MenuItem menuItem7;
private System.Windows.Forms.MenuItem menuItem8;
private System.Windows.Forms.MenuItem menuItem9;
private System.Windows.Forms.MenuItem menuItem10;
private System.Windows.Forms.MenuItem menuItem11;
private System.Windows.Forms.MenuItem menuItem12;
private System.Windows.Forms.MenuItem menuItem13;
private System.Windows.Forms.MenuItem menuItem14;
private System.Windows.Forms.MenuItem menuItem15;
private System.Windows.Forms.MenuItem menuItem16;
private System.Windows.Forms.MenuItem menuItem17;
private System.Windows.Forms.MenuItem menuItem18;
private System.Windows.Forms.MenuItem menuItem19;
private System.Windows.Forms.MenuItem menuItem20;
private System.Windows.Forms.MenuItem menuItem21;
private System.Windows.Forms.MenuItem menuItem22;
private System.Windows.Forms.MenuItem menuItem23;
private System.Windows.Forms.MenuItem menuItem24;
private System.Windows.Forms.MenuItem menuItem25;
private System.Windows.Forms.MenuItem menuItem26;
private System.Windows.Forms.MenuItem menuItem27;
```

```
private System.Windows.Forms.MenuItem menuItem28;
private System.Windows.Forms.MenuItem menuItem29;
private System.Windows.Forms.MenuItem menuItem30;
private System.Windows.Forms.MenuItem menuItem31;
private System.Windows.Forms.MenuItem menuItem32;
private System.Windows.Forms.MenuItem menuItem33;
private System.Windows.Forms.MenuItem menuItem34;
private System.Windows.Forms.MenuItem menuItem35;
private System.Windows.Forms.MenuItem menuItem36;
private System.Windows.Forms.MenuItem menuItem37;
private System.Windows.Forms.MenuItem menuItem38;
private System.Windows.Forms.MenuItem menuItem39;
private System.Windows.Forms.MenuItem menuItem40;
private System.Windows.Forms.MenuItem menuItem41;
private System.Windows.Forms.MenuItem menuItem42;
private System.Windows.Forms.ContextMenu cmCapture2;
private System.Windows.Forms.ContextMenu cmCapture3;
private System.Windows.Forms.ContextMenu cmCapture4;
private System.Windows.Forms.ContextMenu cmCapture5;
private System.Windows.Forms.ContextMenu cmCapture6;
private System.Windows.Forms.ContextMenu cmCapture7;
private System.Windows.Forms.ContextMenu cmCapture8;
private System.Windows.Forms.ContextMenu cmCapture9;
private System.Windows.Forms.ContextMenu cmCapture10;
private System.Windows.Forms.ContextMenu cmCapture11;
private System.Windows.Forms.ContextMenu cmCapture12;
private System.Windows.Forms.ContextMenu cmCapture13;
private System.Windows.Forms.ContextMenu cmCapture14;
private System.Windows.Forms.Button cmdRemoveAll;
private System.Windows.Forms.OpenFileDialog ofdVideo;
private System.Windows.Forms.GroupBox gbVideo;
private System.Windows.Forms.Button cmdStop;
private System.Windows.Forms.Button cmdPlayPause;
private System.Windows.Forms.CheckBox chkLoop;
private System.Windows.Forms.TrackBar tbVidVolume;
private System.Windows.Forms.CheckBox chkMute;
private System.Windows.Forms.GroupBox gbVidTimes;
private System.Windows.Forms.Label lblCurrentVidTimeLabel;
private System.Windows.Forms.Label lblVidCurrent;
private System.Windows.Forms.Label lblTotalVidTimeLabel;
private System.Windows.Forms.Label lblVidTotal;
private System.Windows.Forms.TrackBar tbVideo;
private System.Windows.Forms.Timer tmVideo;
private System.Windows.Forms.Panel pnlCamVideo;
private System.Windows.Forms.FolderBrowserDialog fbdSave;
//  Windows Image Aquisition Device variable used
```

```
        //  to store the currently loaded Capture Device.
        private WIA.Device CurrentCam;
        //  WIA manager COM object used to store the current system WIA
        //  information.
        private WiaClass            wiaManager;
        //  WIA ItemClass object used to store the currently selected capture
        //  device.
        private ItemClass           wiaCamera;
        //  WIAVideoClass object used to store the current video stream.
        private WiaVideoClass       wiaVideo;
        //  Video variable used to store the current video object.
        private Video CurrentVideo;
        //  Integer variable used to store the current state of the
        //  video stream.
        private int PlayPause;
        //  Integer variable used to store the current state of the
        //  WIA video stream.
        private int CamState;
        //  Integer variable used as an index to keep track of the
        //  current bitmap in the BitmapArray.
        int CurrentBitmap;
        private System.Windows.Forms.ImageList ilControls;
        Bitmap[] BitmapArray = new Bitmap[15];
```

1.10.3 Form Initialization

```
//  Startup subroutine for frmCreateStills.
public frmCreateStills()
{
                //  Initialize form items.
                InitializeComponent();
                //  Make sure no camera is currently loaded.
                CurrentCam = null;
                //  Disable the following buttons
                cmdCapture.Enabled = false;
                cmdSaveAll.Enabled = false;
                cmdBurst.Enabled = false;
                cmdRemoveAll.Enabled = false;
                //  Set the BitmapArray's index to the first item.
                CurrentBitmap = 0;
                //  Initialize the varaible
                PlayPause = 2;
                //  Make sure no tabpages are currently loaded
                tcStills.TabPages.Remove(tabCamVideo);
}
```

1.10.4 Form Designer Generated Code

```
/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
	if( disposing )
	{
		//	DisposePicure();
		if( wiaVideo != null )
		  // release any COM instances
		  Marshal.ReleaseComObject( wiaVideo ); wiaVideo = null;
		if( wiaCamera != null )
		  Marshal.ReleaseComObject( wiaCamera ); wiaCamera = null;
		if( wiaManager != null )
		  Marshal.ReleaseComObject( wiaManager ); wiaManager = null;
		if (components != null)
		{
			components.Dispose();
		}
	}
	base.Dispose( disposing );
}

#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
	//  Component code omitted.
}
#endregion
```

1.10.5 Type toolbar clicked

```
//  This subroutine acts as a switchboard to decide which button was
//  clicked on the tbType toolbar.
private void tbType_ButtonClick(object sender,
	 System.Windows.Forms.ToolBarButtonClickEventArgs e)
{
	switch(tbType.Buttons.IndexOf(e.Button))
	{
	    //  Called when the 'Video Mode' button is clicked.
```

```
                case 0: VideoMode();
                        break;
            //  Cakked when the 'Camera Mode' button is clicked.
                case 1: CameraMode();
                        break;
        }
}
```

1.10.6 Reset Page

```
//  This subroutine is used to set all of the form's variables and
//  components back to their original state.
private void ResetPage()
{
        //  If there is less than 15 bitmaps in the array,
        //  enable the buttons.
         if (CurrentBitmap < 14)
        {
                cmdCapture.Enabled = true;
                cmdBurst.Enabled = true;
        }
        //  Hide the video and camera controls
        gbCamera.Visible = false;
        gbVideo.Visible = false;
        //  If a video is currently playing; stop it.
        if (CurrentVideo != null)
        {
                CurrentVideo.Stop();
        }
        //  Removes the current video object and stops the
        //  video's timer.
        CurrentVideo = null;
        tmVideo.Enabled = false;
        //  Initialize the variable.
        PlayPause = 2;
        //  Make sure no tabpages are currently loaded.
        tcStills.TabPages.Remove(tabCamVideo);
        //  Set the Play/Pause button's image to the Play icon.
        cmdPlayPause.ImageIndex = 0;
        //  Sets the volume slider to maximum.
        tbVidVolume.Value = -1;
        //  Clears the info text.
        lblInfo.Text = "";
}
```

1.10.7 Video Selection

```
//  This subroutine is called when the Video Mode is selected.  It allows the user
//  to select a video to display.
private void VideoMode()
{
        //  Set the Open FIle Dialog options.
        ofdVideo.Filter = "AVI Files (*.avi)|*.avi|MPEG Files (*.mpg)|*.mpg|All Files
                          (*.*)|*.*";
        ofdVideo.RestoreDirectory = true ;
         //  Display the Open File Dialog.
        if (ofdVideo.ShowDialog() == DialogResult.OK)
        {
                //  Variables used to store the video runtime.
                int hours, minutes, seconds;
                //  Resets all the page variables.
                ResetPage();
                //  Add the tabpage.
                tcStills.TabPages.Add(tabCamVideo);
                //  Show the video controls.
                gbVideo.Visible = true;
                //  Informs the user that Video Mode was selected
                lblInfo.Text = "Video Mode";
                //  Sets the tabpage's text and icon.
                tabCamVideo.ImageIndex = 0;
                tabCamVideo.Text = "Video Mode:  " + ofdVideo.FileName;
                //  Sets the current video object to the selected video.
                CurrentVideo = new Video(ofdVideo.FileName);
                //  Assigns the video object to a component
                CurrentVideo.Owner = pnlCamVideo;
                //  Resizes the video.
                pnlCamVideo.Width = 640;
                pnlCamVideo.Height = 480;
                //  Calculates and displays the video runtime.
                seconds = (int)CurrentVideo.Duration % 60;
                minutes = (int)CurrentVideo.Duration / 60 % 60;
                hours = (int)CurrentVideo.Duration /60 / 60;
                if (hours < 10)
                   lblVidTotal.Text = "0" + hours.ToString() + ":";
                else
                   lblVidTotal.Text = hours.ToString() + ":";
                if (minutes < 10)
                   lblVidTotal.Text = lblVidTotal.Text + "0" + minutes.ToString() +
                                      ":";
                else
```

```
            lblVidTotal.Text = lblVidTotal.Text + minutes.ToString() + ":";
          if (seconds < 10)
            lblVidTotal.Text = lblVidTotal.Text + "0" + seconds.ToString();
          else
            lblVidTotal.Text = lblVidTotal.Text + seconds.ToString();
           //  Initializes the video slider.
          tbVideo.Maximum = (int) CurrentVideo.Duration;
          //  Play the video to get the first frame as a display
          //  and then pause.
          CurrentVideo.Play();
          CurrentVideo.Pause();
      }
}
```

1.10.8 Camera Selection

```
//  This subroutine is called whenever a capture device needs to be
//  selected.
private void CameraMode()
{
       //  Tries to load a capture device.  If an error is found, an appropriate
      //  message is shown to the user.
      try
      {
            //  Create a new instance of a WiaClass object.
            wiaManager = new WiaClass();
            //  Create a new Select Device Dialog Box.
            WIA.CommonDialogClass class1 = new
                  WIA.CommonDialogClass();
            //  If any other device is loaded, destroy its reference.
            if (CurrentCam != null)
                  wiaVideo.DestroyVideo();
            //  Show the Select Device Dialog Box and assigns the selected
            //  device to CurrentCam.
            CurrentCam = class1.ShowSelectDevice
                        (WIA.WiaDeviceType.UnspecifiedDeviceType,
                         true, false);
            //  If a device was selected.
            if (CurrentCam != null)
            {
                  //  Assigns the selected device's name to the DeviceID
                   //  string.
                  string DeviceID = CurrentCam.DeviceID.ToString();
                  //  Initialize variables
                  object foundID = null;
                  CollectionClass wiaDevs = null;
```

```
DeviceInfoClass devInfo = null;
int Found = 0;
//  Assigns all of the current WIA devices in the system to
//  wiaDevs.
wiaDevs = wiaManager.Devices as CollectionClass;
//  If there were devices
if( wiaDevs != null )
{
        //  Run thrugh all of the devices.
        foreach( object wiaObj in wiaDevs )
        {
                //  Stop the loop if the selected device was
                //  found.
                if (Found == 1)
                        break;
                //  Wrap the wiaObj in a DeviceInfoClass
                //  wrapper.
                devInfo = (DeviceInfoClass)
                        Marshal.CreateWrapperOfType
                        (wiaObj, typeof(DeviceInfoClass) );
                //  Test whether the current wiaObj matches
                //  the selected device.
                if (devInfo.Id.ToString() == DeviceID)
                {
                        //  Used to stop the loop.
                        Found = 1;
                        //  Assigns the object's ID to foundID
                        //  and releases any resources used by
                        //  the wiaObj and devInfo.
                        foundID = devInfo.Id;
                        Marshal.ReleaseComObject( wiaObj );
                        Marshal.ReleaseComObject( devInfo );
                }
        }
}

// Reinitialize variables
devInfo = null;
foundID = System.Reflection.Missing.Value;
//  Creates a video capture object using the selected device.
wiaCamera = (ItemClass) wiaManager.Create( ref foundID );
//  May sometimes take a while, so inform the user that the
//  system  is loading.
Cursor.Current = Cursors.WaitCursor;
//  Try to load the video stream, if it fails a relevant message
//  is shown to the user.
```

```
try
{
        //  Create a new instance of a WiaVideoClass object.
        wiaVideo = new WiaVideoClass();
        //  MSDN specifies that the ImagesDirectory should
        //  be set  to the WIA_DPV_IMAGES_DIRECTORY.
        //  This is specified as 3587.
        string videoDir = (string) wiaCamera.GetPropById(
                        (WiaItemPropertyId) 3587 );
        wiaVideo.ImagesDirectory = videoDir;
        //  Assigns the ID of the selected device to the
        //  cameraID string.
        string cameraID = wiaCamera.GetPropById(
                (WiaItemPropertyId) WiaDeviceInfoPropertyId.
                DeviceInfoDevId ) as string;
        //  Accesses system pointers (handles) so the code
        //  needs to be declared unsafe in order
        //  for the compiler to process it.
        unsafe
        {
                //  In order to create the video display,
                //  wiaVideo needs the handle of the
                //  componenent on which the video needs to
                //  be displayed.  This reference needs to be in
                //  the format of a
                //  WIAVideoLib._RemotableHandle,
                //  so the the Handle of pnlCam needs to be
                //  cast to this format.
                WIAVIDEOLib._RemotableHandle *handle =
                        (WIAVIDEOLib._RemotableHandle*)
                        pnlCam.Handle.ToPointer();
                wiaVideo.CreateVideoByWiaDevID(cameraID,
                        ref *handle, 1, 0);
        }
}
catch( Exception )
{
        //  If a wiaVideo object was loaded, release it.
        if( wiaVideo != null )
                Marshal.ReleaseComObject( wiaVideo );
        //  Reinitialize the variable.
        wiaVideo = null;
        //  Display an appropriate message to the user.
        MessageBox.Show( this, "Create video stream failed",
                        "WIA", MessageBoxButtons.OK,
```

```
                                                    MessageBoxIcon.Stop );

                        }
                        finally
                        {
                                //  If everything was successful, do the following.
                                try
                                {
                                        //  Make the video display visible.
                                        pnlCamVideo.Visible = true;
                                        //  Add the tabpage.
                                        tcStills.TabPages.Add(tabCamVideo);
                                        //  Play the capture device stream.
                                        wiaVideo.Play();
                                        //  Sets the variable to indicate play.
                                        CamState = 1;
                                        //  Display a message to the user.
                                        lblInfo.Text = "Camera Loaded";
                                        //  Set the tabpage's icon and text
                                        tabCamVideo.ImageIndex = 1;
                                        tabCamVideo.Text = "Camera Mode";
                                        //  Make the capture stream's controls
                                        //  visible.
                                        gbCamera.Visible = true;
                                        //  Enable the Capture and Burst buttons.
                                        cmdCapture.Enabled = true;
                                        cmdBurst.Enabled = true;
                                }
                                catch( Exception ) {}
                        }
                }
        }
        catch( Exception )
        {
                //  Display an appropriate message to the user.
                MessageBox.Show( this, "Camera connection failed.", "Message",
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Stop );
        }
}
```

1.10.9 Camera Play/Pause Button

```
//  This subroutine is called when the capture stream's
//  Play/Pause button is clicked.
private void cmdCamPlayPause_Click(object sender, System.EventArgs e)
{
        //  If the stream is played, pause it.  If it's paused
        //  play it.
        if (CamState == 0)
        {
                try
                {
                        wiaVideo.Play();
                        CamState = 1;
                        cmdCamPlayPause.ImageIndex = 1;
                }
                catch( Exception ) {}
        }
        else
        {
                try
                {
                        wiaVideo.Pause();
                        CamState = 0;
                        cmdCamPlayPause.ImageIndex = 0;
                }
                catch( Exception ) {}
        }
}
```

1.10.10 Video Play/Pause Button

```
//  This subroutine is called when the video's Play/Pause
//  button is clicked.
private void cmdPlayPause_Click(object sender, System.EventArgs e)
{
        //  Make sure the Video timer is enabled.
        tmVideo.Enabled = true;
        //  If the video is paused, play it.  If its playing,
        //  pause it.
        if (PlayPause == 1)
        {
                PlayPause = 2;
                CurrentVideo.Pause();
```

```
            cmdPlayPause.ImageIndex = 0;
        }
        else
            if (PlayPause == 2)
            {
                PlayPause = 1;
                CurrentVideo.Play();
                cmdPlayPause.ImageIndex = 1;
                cmdStop.Enabled = true;
            }
}
```

1.10.11 Video Stop Button

```
//  This subroutine is called whenever the video's Stop
//  button is clicked.
private void cmdStop_Click(object sender, System.EventArgs e)
{
        //  Stop the video.
        CurrentVideo.Stop();
        //  Disable the video timer.
        tmVideo.Enabled = false;
        //  Reset the video controls.
        cmdPlayPause.ImageIndex = 0;
        PlayPause = 2;
        lblVidCurrent.Text = "00:00:00";
        tbVideo.Value = 0;
}
```

1.10.12 Video Mute checkbox

```
//  This subroutine is called when the Mute checkbox is clicked.
//  It mutes or re-enables the sound of the video stream.

private void chkMute_CheckedChanged(object sender, System.EventArgs e)
{
        if (chkMute.Checked == true)
        {
            CurrentVideo.Audio.Volume = -10000;
            tbVidVolume.Enabled = false;
        }
        if (chkMute.Checked == false)
        {
            CurrentVideo.Audio.Volume = tbVidVolume.Value;
            tbVidVolume.Enabled = true;
        }}
```

1.10.13 Video Position scrollbar

```
//  This subroutine is called whenever the Video position
//  slider's value changes.
private void tbVideo_Scroll(object sender, System.EventArgs e)
{
        //  Changes the video's position to that of the slider.
        CurrentVideo.CurrentPosition = tbVideo.Value;
        //  Integer variables used to store the video's
        //  current time value.
        int hours, minutes, seconds;
        //  Calculates and displays the video's current time.
        seconds = (int)CurrentVideo.CurrentPosition % 60;
        minutes = (int)CurrentVideo.CurrentPosition / 60 % 60;
        hours = (int)CurrentVideo.CurrentPosition /60 / 60;
        if (hours < 10)
                lblVidCurrent.Text = "0" + hours.ToString() + ":";
        else
                lblVidCurrent.Text = hours.ToString() + ":";
        if (minutes < 10)
                lblVidCurrent.Text = lblVidCurrent.Text + "0" + minutes.ToString() +
                                     ":";
        else
                lblVidCurrent.Text = lblVidCurrent.Text + minutes.ToString() + ":";
        if (seconds < 10)
                lblVidCurrent.Text = lblVidCurrent.Text + "0" + seconds.ToString();
        else
                lblVidCurrent.Text = lblVidCurrent.Text + seconds.ToString();
}
```

1.10.14 Video Volume scrollbar

```
//  This slider is called whenever the Video volume slider's
//  position changes.
private void tbVidVolume_Scroll(object sender, System.EventArgs e)
{
        //  Sets the video's volume to the value of the slider.
        CurrentVideo.Audio.Volume = tbVidVolume.Value;
}
```

1.10.15 Video Timer

```
//  This subroutine is called every time the Video timer's
//  reaches its threshold (100).
private void tmVideo_Tick(object sender, System.EventArgs e)
{
```

```
//  Integer variables used to store the video's
//  current time value.
int hours, minutes, seconds;
//  Calculates and displays the video's current time.
seconds = (int)CurrentVideo.CurrentPosition % 60;
minutes = (int)CurrentVideo.CurrentPosition / 60 % 60;
hours = (int)CurrentVideo.CurrentPosition /60 / 60;
if (hours < 10)
        lblVidCurrent.Text = "0" + hours.ToString() + ":";
else
        lblVidCurrent.Text = hours.ToString() + ":";
if (minutes < 10)
        lblVidCurrent.Text = lblVidCurrent.Text + "0" + minutes.ToString() +
                                ":";
else
        lblVidCurrent.Text = lblVidCurrent.Text + minutes.ToString() + ":";
if (seconds < 10)
        lblVidCurrent.Text = lblVidCurrent.Text + "0" + seconds.ToString();
else
        lblVidCurrent.Text = lblVidCurrent.Text + seconds.ToString();
//  Sets the Video position slider's value to the current
//  position of the video.
tbVideo.Value = (int) CurrentVideo.CurrentPosition;
//  Check to see whether the video needs to be looped.
if (chkLoop.Checked == true)
{
        //  Loop it.
        if (lblVidCurrent.Text == lblVidTotal.Text)
        {
                CurrentVideo.CurrentPosition = 0;
                tbVideo.Value = 0;
        }
}
else
        //  Don't loop it.
        if (lblVidCurrent.Text == lblVidTotal.Text)
        {
                //  Reset the video control variables.
                PlayPause = 2;
                cmdPlayPause.ImageIndex = 0;
                tmVideo.Enabled = false;
                CurrentVideo.CurrentPosition = 0;
                tbVideo.Value = 0;
                CurrentVideo.Stop();
                lblVidCurrent.Text = "00:00:00";
        }}
```

1.10.16 Capture Button

```
//  This subroutine is called whenever the Capture button
//  is clicked.  It captures an image and stores it in the
//  next available picture box.
private void cmdCapture_Click(object sender, System.EventArgs e)
{
        //  Make sure that 15 images haven't been captured.
        if (CurrentBitmap < 15)
        {
                //  Gets a handle to the current desktop image and
                //  assigns it to a bitmap variable.
                int hdcSrc = User32.GetWindowDC(User32.GetDesktopWindow()),
                        hdcDest = GDI32.CreateCompatibleDC(hdcSrc),
                        hBitmap = GDI32.CreateCompatibleBitmap(hdcSrc,
                                        GDI32.GetDeviceCaps(hdcSrc,8),
                                        GDI32.GetDeviceCaps(hdcSrc,10));
                GDI32.SelectObject(hdcDest,hBitmap);
                GDI32.BitBlt(hdcDest,0,0,GDI32.
                                GetDeviceCaps(hdcSrc,8),
                                GDI32.GetDeviceCaps(hdcSrc,10),
                hdcSrc,0,0,0x00CC0020);
                Bitmap image =
                                new Bitmap(Image.FromHbitmap(new
                                IntPtr(hBitmap)),
                                Image.FromHbitmap(new IntPtr(hBitmap)).Width,
                                Image.FromHbitmap(new IntPtr(hBitmap)).Height);
                //  Set the size of and position of the capture area.
                RectangleF a = new RectangleF(6,130,640,480);
                //  Assign the captured image to the last position in
                //  the bitmap array.
                BitmapArray[CurrentBitmap] = (Bitmap)
                                image.Clone(a,PixelFormat.Format24bppRgb);
                //  Release the resources used to capture the image.
                User32.ReleaseDC(User32.GetDesktopWindow(),hdcSrc);
                GDI32.DeleteDC(hdcDest);
                GDI32.DeleteObject(hBitmap);
                //  Determine which picturebox the image should be
                //  displayed in.  Enable the SaveAll and RemoveAll
                //  buttons if at least 2 images have been captured.
                //  Disable the Capture and Burst buttons when 15 images
                //  have been captured.
                switch(CurrentBitmap)
                {
                        case 0:   picCapture0.Image = BitmapArray[CurrentBitmap];
                                break;
```

```
                    case 1:   picCapture1.Image = BitmapArray[CurrentBitmap];
                              cmdSaveAll.Enabled = true;
                              cmdRemoveAll.Enabled = true;
                              break;
                    case 2:   picCapture2.Image = BitmapArray[CurrentBitmap];
                              break;
                    case 3:   picCapture3.Image = BitmapArray[CurrentBitmap];
                              break;
                    case 4:   picCapture4.Image = BitmapArray[CurrentBitmap];
                              break;
                    case 5:   picCapture5.Image = BitmapArray[CurrentBitmap];
                              break;
                    case 6:   picCapture6.Image = BitmapArray[CurrentBitmap];
                              break;
                    case 7:   picCapture7.Image = BitmapArray[CurrentBitmap];
                              break;
                    case 8:   picCapture8.Image = BitmapArray[CurrentBitmap];
                              break;
                    case 9:   picCapture9.Image = BitmapArray[CurrentBitmap];
                              break;
                    case 10:   picCapture10.Image =
                                   BitmapArray[CurrentBitmap];
                              break;
                    case 11:   picCapture11.Image =
                                   BitmapArray[CurrentBitmap];
                              break;
                    case 12:   picCapture12.Image =
                                   BitmapArray[CurrentBitmap];
                              break;
                    case 13:   picCapture13.Image =
                                   BitmapArray[CurrentBitmap];
                              break;
                    case 14:   picCapture14.Image =
                                   BitmapArray[CurrentBitmap];
                              cmdCapture.Enabled = false;
                              cmdBurst.Enabled = false;
                              lblInfo.Text = "Image Buffer is full. Please save or
                              remove images.";
                              break;
                }
                //  Increment the bitmap array's index.
                CurrentBitmap++;
        }
}
```

1.10.17 Burst Button

```
//  This subroutine is called when the Burst button is clicked.
//  It captures images until all of the pictureboxes are filled.
private void cmdBurst_Click(object sender, System.EventArgs e)
{
        //  Disable the Capture and Burst buttons and
        //  enable the SaveAll and RemoveAll buttons.
        cmdCapture.Enabled = false;
        cmdBurst.Enabled = false;
        cmdSaveAll.Enabled = true;
        cmdRemoveAll.Enabled = true;
         //  Set the capture interval.
        int temp = (15 - CurrentBitmap) * 10000000;
        //  Run until the total time is reached.
        for(int i = 0; i < temp; i++)
        {
                //  Capture an image if the time interval is reached
                //  and store it in the first available picturebox.
                if (i % 10000000 == 0)
                {
                        //  Gets a handle to the current desktop image and
                        //  assigns it to a bitmap variable.
                        int hdcSrc =
                            User32.GetWindowDC(User32.GetDesktopWindow()),
                        hdcDest = GDI32.CreateCompatibleDC(hdcSrc),
                                hBitmap = GDI32.CreateCompatibleBitmap(hdcSrc,
                                GDI32.GetDeviceCaps(hdcSrc,8),
                                GDI32.GetDeviceCaps(hdcSrc,10));
                        GDI32.SelectObject(hdcDest,hBitmap);
                        GDI32.BitBlt(hdcDest,0,0,GDI32.
                                 GetDeviceCaps(hdcSrc,8),
                                GDI32.GetDeviceCaps(hdcSrc,10),
                                hdcSrc,0,0,0x00CC0020);
                        Bitmap image =
                                new Bitmap(Image.FromHbitmap(new
                                IntPtr(hBitmap)),
                                Image.FromHbitmap(new IntPtr(hBitmap)).Width,
                                Image.FromHbitmap(new IntPtr(hBitmap)).Height);
                        //  Set the size of and position of the capture area.
                        RectangleF a = new RectangleF(6,130,640,480);
                        //  Assign the captured image to the last position in
                        //  the bitmap array.
                        BitmapArray[CurrentBitmap] = (Bitmap)
                                 image.Clone(a,PixelFormat.Format24bppRgb);
                        //  Release the resources used to capture the image.
```

```
User32.ReleaseDC(User32.GetDesktopWindow(),hdcSrc);
GDI32.DeleteDC(hdcDest);
GDI32.DeleteObject(hBitmap);
//  Determine which picturebox the image should be
//  displayed in.  Enable the SaveAll and RemoveAll
//  buttons if at least 2 images have been captured.
//  Disable the Capture and Burst buttons when 15 images
//  have been captured.
switch(CurrentBitmap)
{
        case 0:   picCapture0.Image =
                  BitmapArray[CurrentBitmap];
                  break;
        case 1:   picCapture1.Image =
                  BitmapArray[CurrentBitmap];
                  break;
        case 2:   picCapture2.Image =
                  BitmapArray[CurrentBitmap];
                  break;
        case 3:   picCapture3.Image =
                  BitmapArray[CurrentBitmap];
                  break;
        case 4:   picCapture4.Image =
                  BitmapArray[CurrentBitmap];
                  break;
        case 5:   picCapture5.Image =
                  BitmapArray[CurrentBitmap];
                  break;
        case 6:   picCapture6.Image =
                  BitmapArray[CurrentBitmap];
                  break;
        case 7:   picCapture7.Image =
                  BitmapArray[CurrentBitmap];
                  break;
        case 8:   picCapture8.Image =
                  BitmapArray[CurrentBitmap];
                  break;
        case 9:   picCapture9.Image =
                  BitmapArray[CurrentBitmap];
                  break;
        case 10:   picCapture10.Image =
                  BitmapArray[CurrentBitmap];
                  break;
        case 11:   picCapture11.Image =
                  BitmapArray[CurrentBitmap];
                  break;
```

```
                        case 12:   picCapture12.Image =
                                BitmapArray[CurrentBitmap];
                                break;
                        case 13:   picCapture13.Image =
                                BitmapArray[CurrentBitmap];
                                break;
                        case 14:   picCapture14.Image =
                                BitmapArray[CurrentBitmap];
                                lblInfo.Text = "Image Buffer is full. Please save
                                        or remove images.";
                                break;
                    }
                    //  Refresh the form to show the latest image.
                    this.Refresh();
                    //  Increment the bitmap array's index.
                    CurrentBitmap++;
                }
        }
}
```

1.10.18 Save All Button

```
//  This subroutine is called when the Save All button is clicked.
//  It saves the images in the pictureboxes to permanent storage.
private void cmdSaveAll_Click(object sender, System.EventArgs e)
{
        //  Show a Select Direct Dialog Box and save all
        //  of the images to the selected directory.
        if (fbdSave.ShowDialog() == DialogResult.OK)
        {
                for (int i = 0; i < CurrentBitmap; i++)
                        BitmapArray[i].Save(fbdSave.SelectedPath  + "\\Capture" +
                                i + ".jpg",ImageFormat.Jpeg);
                //  Clear all of the pictureboxes.
                picCapture0.Image = null;
                picCapture1.Image = null;
                picCapture2.Image = null;
                picCapture3.Image = null;
                picCapture4.Image = null;
                picCapture5.Image = null;
                picCapture6.Image = null;
                picCapture7.Image = null;
                picCapture8.Image = null;
                picCapture9.Image = null;
                picCapture10.Image = null;
                picCapture11.Image = null;
```

```
                picCapture12.Image = null;
                picCapture13.Image = null;
                picCapture14.Image = null;
                //  Set the bitmap array's index to 0.
                CurrentBitmap = 0;
                //  Display a message to the user.
                lblInfo.Text = "Images Saved";
                //  Enable and Disable the appropriate buttons.
                cmdSaveAll.Enabled = false;
                cmdCapture.Enabled = true;
                cmdBurst.Enabled = true;
        }
}
```

1.10.19 Remove All Button

```
//  This subroutine is called when the Remove All button is clicked.
//  It clears all of the pictureboxes.
private void cmdRemoveAll_Click(object sender, System.EventArgs e)
{
        //  Clear the bitmap array.
        for (int i = 0; i < CurrentBitmap; i++)
                BitmapArray[i] = null;
         //  Initialize the bitmap array's index.
        CurrentBitmap = 0;
        //  Clear all of the pictureboxes.
        picCapture0.Image = null;
        picCapture1.Image = null;
        picCapture2.Image = null;
        picCapture3.Image = null;
        picCapture4.Image = null;
        picCapture5.Image = null;
        picCapture6.Image = null;
        picCapture7.Image = null;
        picCapture8.Image = null;
        picCapture9.Image = null;
        picCapture10.Image = null;
        picCapture11.Image = null;
        picCapture12.Image = null;
        picCapture13.Image = null;
        picCapture14.Image = null;
        //  Enable and disable the appropriate buttons.
        cmdSaveAll.Enabled = false;
        cmdCapture.Enabled = true;
        cmdBurst.Enabled = true;
        cmdRemoveAll.Enabled = false;}
```

1.10.20 Save As Context Menu option

```
//  This subroutine is called when the Save As option is
//  selected in a picturebox's context menu.
private void SaveAs(int CalledBy)
{
        //  Sets the Save File Dialog Box's properties and
        //  displays it.
        Bitmap temp = null;
        sfdCaptures.Filter = "JPEG Files (*.jpg)|*.jpg";
        sfdCaptures.AddExtension = true;
        sfdCaptures.DefaultExt = "*.jpg";
        sfdCaptures.FileName = "";

        if (sfdCaptures.ShowDialog() == DialogResult.OK)
        {
                //  Assigns the image of the selected picturebox
                //  to the temp variable.
                switch(CalledBy)
                {
                        case 0:   temp = (Bitmap) picCapture0.Image;
                                  break;
                        case 1:   temp = (Bitmap) picCapture1.Image;
                                  break;
                        case 2:   temp = (Bitmap) picCapture2.Image;
                                  break;
                        case 3:   temp = (Bitmap) picCapture3.Image;
                                  break;
                        case 4:  temp = (Bitmap) picCapture4.Image;
                                  break;
                        case 5:  temp = (Bitmap) picCapture5.Image;
                                  break;
                        case 6:  temp = (Bitmap) picCapture6.Image;
                                  break;
                        case 7:  temp = (Bitmap) picCapture7.Image;
                                  break;
                        case 8:  temp = (Bitmap) picCapture8.Image;
                                  break;
                        case 9:  temp = (Bitmap) picCapture9.Image;
                                  break;
                        case 10: temp = (Bitmap) picCapture10.Image;
                                  break;
                        case 11: temp = (Bitmap) picCapture11.Image;
                                  break;
                        case 12: temp = (Bitmap) picCapture12.Image;
                                  break;
```

```
                    case 13: temp = (Bitmap) picCapture13.Image;
                            break;
                    case 14: temp = (Bitmap) picCapture14.Image;
                            break;
                }
                // Save the selected image.
                temp.Save(sfdCaptures.FileName,
                        ImageFormat.Jpeg);
                // Calls the subroutine to remove the selected image.
                Remove(CalledBy);
        }
}
```

1.10.21 Remove Context Menu option

```
// This subroutine is called when the Remove option is selected in
// a picturebox's context menu or when a image has been saved..
private void Remove(int CalledBy)
{
        // Enable the Capture and Burst buttons.
        cmdCapture.Enabled = true;
        cmdBurst.Enabled = true;
        // Decrement the bitmap array's index.
        CurrentBitmap--;
        // Sets the selected picturebox's image to null and moves up
        // all the others.
        switch(CalledBy)
        {
                case 0:   picCapture0.Image =
                        BitmapArray[0] = (Bitmap)picCapture1.Image;
                        picCapture1.Image =
                        BitmapArray[1] = (Bitmap)picCapture2.Image;
                        picCapture2.Image =
                        BitmapArray[2] = (Bitmap)picCapture3.Image;
                        picCapture3.Image =
                        BitmapArray[3] = (Bitmap)picCapture4.Image;
                        picCapture4.Image =
                        BitmapArray[4] = (Bitmap)picCapture5.Image;
                        picCapture5.Image =
                        BitmapArray[5] = (Bitmap)picCapture6.Image;
                        picCapture6.Image =
                        BitmapArray[6] = (Bitmap)picCapture7.Image;
                        picCapture7.Image =
                        BitmapArray[7] = (Bitmap)picCapture8.Image;
                        picCapture8.Image =
                        BitmapArray[8] = (Bitmap)picCapture9.Image;
```

```
            picCapture9.Image =
            BitmapArray[9] = (Bitmap)picCapture10.Image;
            picCapture10.Image =
            BitmapArray[10] = (Bitmap)picCapture11.Image;
            picCapture11.Image =
            BitmapArray[11] = (Bitmap)picCapture12.Image;
            picCapture12.Image =
            BitmapArray[12] = (Bitmap)picCapture13.Image;
            picCapture13.Image =
            BitmapArray[13] = (Bitmap)picCapture14.Image;
            picCapture14.Image =
            BitmapArray[14] = null;
            break;
        case 1:
            picCapture1.Image =
            BitmapArray[1] = (Bitmap)picCapture2.Image;
            picCapture2.Image =
            BitmapArray[2] = (Bitmap)picCapture3.Image;
            picCapture3.Image =
            BitmapArray[3] = (Bitmap)picCapture4.Image;
            picCapture4.Image =
            BitmapArray[4] = (Bitmap)picCapture5.Image;
            picCapture5.Image =
            BitmapArray[5] = (Bitmap)picCapture6.Image;
            picCapture6.Image =
            BitmapArray[6] = (Bitmap)picCapture7.Image;
            picCapture7.Image =
            BitmapArray[7] = (Bitmap)picCapture8.Image;
            picCapture8.Image =
            BitmapArray[8] = (Bitmap)picCapture9.Image;
            picCapture9.Image =
            BitmapArray[9] = (Bitmap)picCapture10.Image;
            picCapture10.Image =
            BitmapArray[10] = (Bitmap)picCapture11.Image;
            picCapture11.Image =
            BitmapArray[11] = (Bitmap)picCapture12.Image;
            picCapture12.Image =
            BitmapArray[12] = (Bitmap)picCapture13.Image;
            picCapture13.Image =
            BitmapArray[13] = (Bitmap)picCapture14.Image;
            picCapture14.Image =
            BitmapArray[14] = null;
            break;
        case 2:
            picCapture2.Image =
            BitmapArray[2] = (Bitmap)picCapture3.Image;
```

```
            picCapture3.Image =
            BitmapArray[3] = (Bitmap)picCapture4.Image;
            picCapture4.Image =
            BitmapArray[4] = (Bitmap)picCapture5.Image;
            picCapture5.Image =
            BitmapArray[5] = (Bitmap)picCapture6.Image;
            picCapture6.Image =
            BitmapArray[6] = (Bitmap)picCapture7.Image;
            picCapture7.Image =
            BitmapArray[7] = (Bitmap)picCapture8.Image;
            picCapture8.Image =
            BitmapArray[8] = (Bitmap)picCapture9.Image;
            picCapture9.Image =
            BitmapArray[9] = (Bitmap)picCapture10.Image;
            picCapture10.Image =
            BitmapArray[10] = (Bitmap)picCapture11.Image;
            picCapture11.Image =
            BitmapArray[11] = (Bitmap)picCapture12.Image;
            picCapture12.Image =
            BitmapArray[12] = (Bitmap)picCapture13.Image;
            picCapture13.Image =
            BitmapArray[13] = (Bitmap)picCapture14.Image;
            picCapture14.Image =
            BitmapArray[14] = null;
            break;
        case 3:
            picCapture3.Image =
            BitmapArray[3] = (Bitmap)picCapture4.Image;
            picCapture4.Image =
            BitmapArray[4] = (Bitmap)picCapture5.Image;
            picCapture5.Image =
            BitmapArray[5] = (Bitmap)picCapture6.Image;
            picCapture6.Image =
            BitmapArray[6] = (Bitmap)picCapture7.Image;
            picCapture7.Image =
            BitmapArray[7] = (Bitmap)picCapture8.Image;
            picCapture8.Image =
            BitmapArray[8] = (Bitmap)picCapture9.Image;
            picCapture9.Image =
            BitmapArray[9] = (Bitmap)picCapture10.Image;
            picCapture10.Image =
            BitmapArray[10] = (Bitmap)picCapture11.Image;
            picCapture11.Image =
            BitmapArray[11] = (Bitmap)picCapture12.Image;
            picCapture12.Image =
            BitmapArray[12] = (Bitmap)picCapture13.Image;
```

```
            picCapture13.Image =
            BitmapArray[13] = (Bitmap)picCapture14.Image;
            picCapture14.Image =
            BitmapArray[14] = null;
            break;
        case 4:
            picCapture4.Image =
            BitmapArray[4] = (Bitmap)picCapture5.Image;
            picCapture5.Image =
            BitmapArray[5] = (Bitmap)picCapture6.Image;
            picCapture6.Image =
            BitmapArray[6] = (Bitmap)picCapture7.Image;
            picCapture7.Image =
            BitmapArray[7] = (Bitmap)picCapture8.Image;
            picCapture8.Image =
            BitmapArray[8] = (Bitmap)picCapture9.Image;
            picCapture9.Image =
            BitmapArray[9] = (Bitmap)picCapture10.Image;
            picCapture10.Image =
            BitmapArray[10] = (Bitmap)picCapture11.Image;
            picCapture11.Image =
            BitmapArray[11] = (Bitmap)picCapture12.Image;
            picCapture12.Image =
            BitmapArray[12] = (Bitmap)picCapture13.Image;
            picCapture13.Image =
            BitmapArray[13] = (Bitmap)picCapture14.Image;
            picCapture14.Image =
            BitmapArray[14] = null;
            break;
        case 5:
            picCapture5.Image =
            BitmapArray[5] = (Bitmap)picCapture6.Image;
            picCapture6.Image =
            BitmapArray[6] = (Bitmap)picCapture7.Image;
            picCapture7.Image =
            BitmapArray[7] = (Bitmap)picCapture8.Image;
            picCapture8.Image =
            BitmapArray[8] = (Bitmap)picCapture9.Image;
            picCapture9.Image =
            BitmapArray[9] = (Bitmap)picCapture10.Image;
            picCapture10.Image =
            BitmapArray[10] = (Bitmap)picCapture11.Image;
            picCapture11.Image =
            BitmapArray[11] = (Bitmap)picCapture12.Image;
            picCapture12.Image =
            BitmapArray[12] = (Bitmap)picCapture13.Image;
```

```
            picCapture13.Image =
            BitmapArray[13] = (Bitmap)picCapture14.Image;
            picCapture14.Image =
            BitmapArray[14] = null;
            break;
        case 6:
            picCapture6.Image =
            BitmapArray[6] = (Bitmap)picCapture7.Image;
            picCapture7.Image =
            BitmapArray[7] = (Bitmap)picCapture8.Image;
            picCapture8.Image =
            BitmapArray[8] = (Bitmap)picCapture9.Image;
            picCapture9.Image =
            BitmapArray[9] = (Bitmap)picCapture10.Image;
            picCapture10.Image =
            BitmapArray[10] = (Bitmap)picCapture11.Image;
            picCapture11.Image =
            BitmapArray[11] = (Bitmap)picCapture12.Image;
            picCapture12.Image =
            BitmapArray[12] = (Bitmap)picCapture13.Image;
            picCapture13.Image =
            BitmapArray[13] = (Bitmap)picCapture14.Image;
            picCapture14.Image =
            BitmapArray[14] = null;
            break;
        case 7:
            picCapture7.Image =
            BitmapArray[7] = (Bitmap)picCapture8.Image;
            picCapture8.Image =
            BitmapArray[8] = (Bitmap)picCapture9.Image;
            picCapture9.Image =
            BitmapArray[9] = (Bitmap)picCapture10.Image;
            picCapture10.Image =
            BitmapArray[10] = (Bitmap)picCapture11.Image;
            picCapture11.Image =
            BitmapArray[11] = (Bitmap)picCapture12.Image;
            picCapture12.Image =
            BitmapArray[12] = (Bitmap)picCapture13.Image;
            picCapture13.Image =
            BitmapArray[13] = (Bitmap)picCapture14.Image;
            picCapture14.Image =
            BitmapArray[14] = null;
            break;
        case 8:
            picCapture8.Image =
            BitmapArray[8] = (Bitmap)picCapture9.Image;
```

```
            picCapture9.Image =
            BitmapArray[9] = (Bitmap)picCapture10.Image;
            picCapture10.Image =
            BitmapArray[10] = (Bitmap)picCapture11.Image;
            picCapture11.Image =
            BitmapArray[11] = (Bitmap)picCapture12.Image;
            picCapture12.Image =
            BitmapArray[12] = (Bitmap)picCapture13.Image;
            picCapture13.Image =
            BitmapArray[13] = (Bitmap)picCapture14.Image;
            picCapture14.Image =
            BitmapArray[14] = null;
            break;
        case 9:
            picCapture9.Image =
            BitmapArray[9] = (Bitmap)picCapture10.Image;
            picCapture10.Image =
            BitmapArray[10] = (Bitmap)picCapture11.Image;
            picCapture11.Image =
            BitmapArray[11] = (Bitmap)picCapture12.Image;
            picCapture12.Image =
            BitmapArray[12] = (Bitmap)picCapture13.Image;
            picCapture13.Image =
            BitmapArray[13] = (Bitmap)picCapture14.Image;
            picCapture14.Image =
            BitmapArray[14] = null;
            break;
        case 10:
            picCapture10.Image =
            BitmapArray[10] = (Bitmap)picCapture11.Image;
            picCapture11.Image =
            BitmapArray[11] = (Bitmap)picCapture12.Image;
            picCapture12.Image =
            BitmapArray[12] = (Bitmap)picCapture13.Image;
            picCapture13.Image =
            BitmapArray[13] = (Bitmap)picCapture14.Image;
            picCapture14.Image =
            BitmapArray[14] = null;
            break;
        case 11:
            picCapture11.Image =
            BitmapArray[11] = (Bitmap)picCapture12.Image;
            picCapture12.Image =
            BitmapArray[12] = (Bitmap)picCapture13.Image;
            picCapture13.Image =
            BitmapArray[13] = (Bitmap)picCapture14.Image;
```

```
                picCapture14.Image =
                BitmapArray[14] = null;
                picCapture14.Image = null;
                break;
        case 12:
                picCapture12.Image =
                BitmapArray[12] = (Bitmap)picCapture13.Image;
                picCapture13.Image =
                BitmapArray[13] = (Bitmap)picCapture14.Image;
                picCapture14.Image =
                BitmapArray[14] = null;
                break;
        case 13:
                picCapture13.Image =
                BitmapArray[13] = (Bitmap)picCapture14.Image;
                picCapture14.Image =
                BitmapArray[14] = null;
                break;
        case 14:
                picCapture14.Image =
                BitmapArray[14] = null;
                break;
        }
}
```

1.10.22 Show in Own Window Context Menu option

```
//  This subroutine is called when the Show in Own Window option
//  is selected in a picturebox's context menu.
private void ShowInOwn(int CalledBy)
{
        //  Create a new instance of frmPicDisplay1.
        frmPicDisplay1 PicDisplay = null;
        //  Determine which image needs to be displayed and then
        //  initializes PicDisplay with it.
        switch(CalledBy)
        {
                case 0:   PicDisplay = new
                        frmPicDisplay1((Bitmap)picCapture0.Image);
                        break;
                case 1:   PicDisplay = new
                        frmPicDisplay1((Bitmap)picCapture1.Image);
                        break;
                case 2:  PicDisplay = new
                        frmPicDisplay1((Bitmap)picCapture2.Image);
                        break;
```

```
                case 3:  PicDisplay = new
                        frmPicDisplay1((Bitmap)picCapture3.Image);
                        break;
                case 4:  PicDisplay = new
                        frmPicDisplay1((Bitmap)picCapture4.Image);
                        break;
                case 5: PicDisplay = new
                        frmPicDisplay1((Bitmap)picCapture5.Image);
                        break;
                case 6: PicDisplay = new
                        frmPicDisplay1((Bitmap)picCapture6.Image);
                        break;
                case 7: PicDisplay = new
                        frmPicDisplay1((Bitmap)picCapture7.Image);
                        break;
                case 8:  PicDisplay = new
                        frmPicDisplay1((Bitmap)picCapture8.Image);
                        break;
                case 9: PicDisplay = new
                        frmPicDisplay1((Bitmap)picCapture9.Image);
                        break;
                case 10: PicDisplay = new
                        frmPicDisplay1((Bitmap)picCapture10.Image);
                        break;
                case 11: PicDisplay = new
                        frmPicDisplay1((Bitmap)picCapture11.Image);
                        break;
                case 12: PicDisplay = new
                        frmPicDisplay1((Bitmap)picCapture12.Image);
                        break;
                case 13: PicDisplay = new
                        frmPicDisplay1((Bitmap)picCapture13.Image);
                        break;
                case 14: PicDisplay = new
                        frmPicDisplay1((Bitmap)picCapture14.Image);
                        break;
                }
        //  Show PicDisplay.
        PicDisplay.ShowDialog();
}
```

1.10.23 Context Menu Items

```
//  The following subroutines are called whenever a specific picturebox's
//  Save As, Remove or Show in Own Window option is selected in its context
//  menu.

private void miSaveAs0_Click(object sender, System.EventArgs e)
{
        if (picCapture0.Image != null)
          SaveAs(0);
        else
          lblInfo.Text = "No image loaded";
}

private void miRemove0_Click(object sender, System.EventArgs e)
{
        if (picCapture0.Image != null)
          Remove(0);
        else
          lblInfo.Text = "No image loaded";
}

private void miShow0_Click(object sender, System.EventArgs e)
{
        if (picCapture0.Image != null)
                ShowInOwn(0);
        else
                lblInfo.Text = "No image loaded";
}

private void miSaveAs1_Click(object sender, System.EventArgs e)
{
        if (picCapture1.Image != null)
                SaveAs(1);
        else
                lblInfo.Text = "No image loaded";
}

private void miRemove1_Click(object sender, System.EventArgs e)
{
        if (picCapture1.Image != null)
                Remove(1);
        else
                lblInfo.Text = "No image loaded";
}
```

```
private void miShow1_Click(object sender, System.EventArgs e)
{
	if (picCapture1.Image != null)
		ShowInOwn(1);
	else
		lblInfo.Text = "No image loaded";
}

private void miSaveAs2_Click(object sender, System.EventArgs e)
{
	if (picCapture2.Image != null)
		SaveAs(2);
	else
		lblInfo.Text = "No image loaded";
}

private void miRemove2_Click(object sender, System.EventArgs e)
{
	if (picCapture2.Image != null)
		Remove(2);
	else
		lblInfo.Text = "No image loaded";
}

private void miShow2_Click(object sender, System.EventArgs e)
{
	if (picCapture2.Image != null)
		ShowInOwn(2);
	else
		lblInfo.Text = "No image loaded";
}

private void miSaveAs3_Click(object sender, System.EventArgs e)
{
	if (picCapture1.Image != null)
		SaveAs(3);
	else
		lblInfo.Text = "No image loaded";
}

private void miRemove3_Click(object sender, System.EventArgs e)
{
	if (picCapture1.Image != null)
		Remove(3);
	else
		lblInfo.Text = "No image loaded";
```

```
}

private void miShow3_Click(object sender, System.EventArgs e)
{
	if (picCapture3.Image != null)
		ShowInOwn(3);
	else
		lblInfo.Text = "No image loaded";
}

private void miSaveAs4_Click(object sender, System.EventArgs e)
{
	if (picCapture4.Image != null)
		SaveAs(4);
	else
		lblInfo.Text = "No image loaded";
}

private void miRemove4_Click(object sender, System.EventArgs e)
{
	if (picCapture4.Image != null)
		Remove(4);
	else
		lblInfo.Text = "No image loaded";
}

private void miShow4_Click(object sender, System.EventArgs e)
{
	if (picCapture4.Image != null)
		ShowInOwn(4);
	else
		lblInfo.Text = "No image loaded";
}


private void miSaveAs5_Click(object sender, System.EventArgs e)
{
	if (picCapture5.Image != null)
		SaveAs(5);
	else
		lblInfo.Text = "No image loaded";
}

private void miRemove5_Click(object sender, System.EventArgs e)
{
	if (picCapture5.Image != null)
```

```
                Remove(5);
        else
                lblInfo.Text = "No image loaded";
}

private void miShow5_Click(object sender, System.EventArgs e)
{
        if (picCapture5.Image != null)
                ShowInOwn(5);
        else
                lblInfo.Text = "No image loaded";
}

private void miSaveAs6_Click(object sender, System.EventArgs e)
{
        if (picCapture6.Image != null)
                SaveAs(6);
        else
                lblInfo.Text = "No image loaded";
}

private void miRemove6_Click(object sender, System.EventArgs e)
{
        if (picCapture6.Image != null)
                Remove(6);
        else
                lblInfo.Text = "No image loaded";
}

private void miShow6_Click(object sender, System.EventArgs e)
{
        if (picCapture6.Image != null)
                ShowInOwn(6);
        else
                lblInfo.Text = "No image loaded";
}

private void miSaveAs7_Click(object sender, System.EventArgs e)
{
        if (picCapture7.Image != null)
                SaveAs(7);
        else
                lblInfo.Text = "No image loaded";
}

private void miRemove7_Click(object sender, System.EventArgs e)
```

```
{
        if (picCapture7.Image != null)
                Remove(7);
        else
                lblInfo.Text = "No image loaded";
}

private void miShow7_Click(object sender, System.EventArgs e)
{
        if (picCapture7.Image != null)
                ShowInOwn(7);
        else
                lblInfo.Text = "No image loaded";
}

private void miSaveAs8_Click(object sender, System.EventArgs e)
{
        if (picCapture8.Image != null)
                SaveAs(8);
        else
                lblInfo.Text = "No image loaded";
}

private void miRemove8_Click(object sender, System.EventArgs e)
{
        if (picCapture8.Image != null)
                Remove(8);
        else
                lblInfo.Text = "No image loaded";
}

private void miShow8_Click(object sender, System.EventArgs e)
{
        if (picCapture8.Image != null)
                ShowInOwn(8);
        else
                lblInfo.Text = "No image loaded";
}

private void miSaveAs9_Click(object sender, System.EventArgs e)
{
        if (picCapture9.Image != null)
                SaveAs(9);
        else
                lblInfo.Text = "No image loaded";
}
```

```
private void miRemove9_Click(object sender, System.EventArgs e)
{
	if (picCapture9.Image != null)
		Remove(9);
	else
		lblInfo.Text = "No image loaded";
}

private void miShow9_Click(object sender, System.EventArgs e)
{
	if (picCapture9.Image != null)
		ShowInOwn(9);
	else
		lblInfo.Text = "No image loaded";
}

private void miSaveAs10_Click(object sender, System.EventArgs e)
{
	if (picCapture10.Image != null)
		SaveAs(10);
	else
		lblInfo.Text = "No image loaded";
}

private void miRemove10_Click(object sender, System.EventArgs e)
{
	if (picCapture10.Image != null)
		Remove(10);
	else
		lblInfo.Text = "No image loaded";
}

private void miShow10_Click(object sender, System.EventArgs e)
{
	if (picCapture10.Image != null)
		ShowInOwn(10);
	else
		lblInfo.Text = "No image loaded";
}

private void miSaveAs11_Click(object sender, System.EventArgs e)
{
	if (picCapture1.Image != null)
		SaveAs(11);
	else
```

```
                lblInfo.Text = "No image loaded";
}

private void miRemove11_Click(object sender, System.EventArgs e)
{
        if (picCapture11.Image != null)
                Remove(11);
        else
                lblInfo.Text = "No image loaded";
}

private void miShow11_Click(object sender, System.EventArgs e)
{
        if (picCapture11.Image != null)
                ShowInOwn(11);
        else
                lblInfo.Text = "No image loaded";
}

private void miSaveAs12_Click(object sender, System.EventArgs e)
{
        if (picCapture12.Image != null)
                SaveAs(12);
        else
                lblInfo.Text = "No image loaded";
}

private void miRemove12_Click(object sender, System.EventArgs e)
{
        if (picCapture12.Image != null)
                Remove(12);
        else
                lblInfo.Text = "No image loaded";
}

private void miShow12_Click(object sender, System.EventArgs e)
{
        if (picCapture12.Image != null)
                ShowInOwn(12);
        else
                lblInfo.Text = "No image loaded";
}

private void miSaveAs13_Click(object sender, System.EventArgs e)
{
        if (picCapture13.Image != null)
```

```
                SaveAs(13);
        else
                lblInfo.Text = "No image loaded";
}

private void miRemove13_Click(object sender, System.EventArgs e)
{
        if (picCapture13.Image != null)
                Remove(13);
        else
                lblInfo.Text = "No image loaded";
}

private void miShow13_Click(object sender, System.EventArgs e)
{
        if (picCapture13.Image != null)
                ShowInOwn(13);
        else
                lblInfo.Text = "No image loaded";
}

private void miSaveAs14_Click(object sender, System.EventArgs e)
{
        if (picCapture14.Image != null)
                SaveAs(14);
        else
                lblInfo.Text = "No image loaded";
}

private void miRemove14_Click(object sender, System.EventArgs e)
{
        if (picCapture14.Image != null)
                Remove(14);
        else
                lblInfo.Text = "No image loaded";
}

private void miShow14_Click(object sender, System.EventArgs e)
{
        if (picCapture14.Image != null)
                ShowInOwn(14);
        else
                lblInfo.Text = "No image loaded";
}
}
}
```

1.10.24 External Classes

```
//  Image class used for capture.
class GDI32
{
        [DllImport("GDI32.dll")]
        public static extern bool BitBlt(int hdcDest,int nXDest,int nYDest,
                int nWidth,int nHeight,int hdcSrc,
                int nXSrc,int nYSrc,int dwRop);
        [DllImport("GDI32.dll")]
        public static extern int CreateCompatibleBitmap(int hdc,int nWidth,
                int nHeight);
        [DllImport("GDI32.dll")]
        public static extern int CreateCompatibleDC(int hdc);
        [DllImport("GDI32.dll")]
        public static extern bool DeleteDC(int hdc);
        [DllImport("GDI32.dll")]
        public static extern bool DeleteObject(int hObject);
        [DllImport("GDI32.dll")]
        public static extern int GetDeviceCaps(int hdc,int nIndex);
        [DllImport("GDI32.dll")]
        public static extern int SelectObject(int hdc,int hgdiobj);
}

//  Windows class used to get the current desktop window.
class    User32
{
        [DllImport("User32.dll")]
        public static extern int GetDesktopWindow();
        [DllImport("User32.dll")]
        public static extern int GetWindowDC(int hWnd);
        [DllImport("User32.dll")]
        public static extern int ReleaseDC(int hWnd,int hDC);
}
```

1.11 Create Video from Camera Feed

This form is called whenever the user wants to record a video from a device feed. Recorded videos can be used in the update or recognition processes. The Create Video form is actually an external .dll file. The code for this .dll file is public domain [], so rather than pull the entire project's code files and forms into my own project, I altered the interface and added some of my own functions; such as video preview and selective saving. However, the changes I made do not warrant displaying the total code listing for the project in these pages.

1.12 Change Brightness Levels (frmBrightness)

This form can be called from either the Update Model or Perform Recognition forms. It is used in Still Image mode to change the brightness levels of the image to be processed.

1.12.1 System References

```
//  Declaration of References
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Drawing.Imaging;
```

1.12.2 Form Items and Global Variables

```
namespace AntsNest
{
/// <summary>
/// Summary description for frmBrightness.
/// </summary>
public class frmBrightness : System.Windows.Forms.Form
{
        private System.Windows.Forms.Button cmdApply;
        private System.Windows.Forms.Button cmdCancel;
        private System.Windows.Forms.GroupBox groupBox1;
        private System.Windows.Forms.GroupBox groupBox2;
        private System.Windows.Forms.PictureBox picPreview;
        private System.Windows.Forms.PictureBox picOriginal;
        private System.Windows.Forms.GroupBox groupBox3;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.Label label1;
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.Container components = null;
        private System.Windows.Forms.TrackBar scrollBrightness;
        private System.Windows.Forms.GroupBox groupBox4;
        private System.Windows.Forms.Label lblValue;
        //  Picturebox variable used as an alias for picStudy on
        //  frmRecognition or frmUpdateModel.
```

```
        private System.Windows.Forms.PictureBox PicBox;
        //  Bitmap variable used as an alias for the picture displayed
        //  in picStudy on frmRecognition or frmUpdateModel.
        private Bitmap picInUse;
```

1.12.3 Form Initialization

```
//  Startup subroutine for frmBrightness.
//  Takes 2 arguments: -  A reference to the image displayed in picStudy on
//                        frmRecognition or frmUpdateModel.
//                     -  A reference to picStudy itself.
public frmBrightness(Bitmap pic,System.Windows.Forms.PictureBox picbox)
{
        InitializeComponent();
        //  Assign passed variables to their local counterparts.
        picInUse = pic;
        PicBox = picbox;
         //  Sets both picPreview and picOriginal to the image displayed
        //  in the picturebox picStudy on frmRecognition or frmUpdateModel
        //  and sizes it appropriately.
        picPreview.Image = picInUse;
        picPreview.SizeMode = PictureBoxSizeMode.StretchImage;
        picOriginal.Image = picInUse;
        picOriginal.SizeMode = PictureBoxSizeMode.StretchImage;
        //  Assigns the label lblValue the default value of scrollbar
        //  scrollBrightness, which is zero.
        lblValue.Text = scrollBrightness.Value.ToString();
}
```

1.12.4 Form Designer Generated Code

```
/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
        if( disposing )
        {
                if(components != null)
                {
                        components.Dispose();
                }
        }
        base.Dispose( disposing );
}
```

```
#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
        //  Component code omitted.
}
#endregion
```

1.12.5 Apply Button

```
//  This subroutine is called when the 'Apply' button is clicked.
private void cmdApply_Click(object sender, System.EventArgs e)
{
        //  Sets the picturebox picStudy on frmRecognition or frmUpdateModel
        //  to the newly brightness-adjusted value and sizes it appropriately.
        //  This change only applies to the current session and is not actually
        //  saved to the image.
        picInUse = Brightness(picInUse, (int)scrollBrightness.Value);
        PicBox.Image = picInUse;
        PicBox.SizeMode = PictureBoxSizeMode.StretchImage;
        //  Closes the form.
        this.Close();
}
```

1.12.6 Cancel Button

```
//  This subroutine is called when the 'Cancel' button is clicked.
private void cmdCancel_Click(object sender, System.EventArgs e)
{
        //  Closes the form, without making any changes.
        this.Close();
}
```

1.12.7 Change the Brightness Level

```
//  This function is called when changes need to be made to an image's
//  brightness levels.  It returns the altered bitmap.
//  Takes 2 arguments: -  A reference to the image to be altered.
//                     -  A reference to the chosen brightness level.
private static Bitmap Brightness(Bitmap pic, int nBrightness)
{
        //  Locks the region of the bitmap on which processing is to be performed.
```

```
//  In this case, the whole bitmap is processed.
BitmapData bmData = pic.LockBits(new Rectangle(0, 0, pic.Width,
        pic.Height), ImageLockMode.ReadWrite,
        PixelFormat.Format24bppRgb);
//  Sets how wide a bitmap scanline is in bytes and
//  the location in memory where to start reading.
//  Because this is a 24 bits per pixel bitmap, each
//  pixel uses 3 bytes to store its data.  So the stride
//  will basically be (bitmap width) * 3.
int stride = bmData.Stride;
System.IntPtr Scan0 = bmData.Scan0;
//  Used to store a temporary value, which will be
//  assigned to the bitmap as it's new value at the
//  current point.
int nVal = 0;
//  Allows the use of pointers.
unsafe
{
        //  Creates a pointer to the spot where to start
        //  reading the bitmap.
        byte * p = (byte *)(void *)Scan0;
        //  Sets it up so we start working at the start of
        //  the first pixel of each row.
        int nOffset = stride - pic.Width*3;
        //  Makes sure that every byte in a row is processed.
        //  That means the same results will be applied to the
        //  Red, Green and Blue values of each pixel.
        int nWidth = pic.Width * 3;
        //  Run through all the bytes in the bitmap.
        for(int y=0;y<pic.Height;++y)
        {
                for(int x=0; x < nWidth; ++x )
                {
                        //  Creates a new brightness-shifted
                        //  value for the specific byte
                        nVal = (int) (p[0] + nBrightness);
                        //  Normalizes the values.  A single byte
                        //  does not allow values less than 0 or more
                        //  than 255.
                        if (nVal < 0)
                            nVal = 0;
                        if (nVal > 255)
                            nVal = 255;
                        //  Assigns the brightness-shifted value to
                        //  the current byte.
                        p[0] = (byte)nVal;
```

```
                        // Increment p after the changes have been made.
                        ++p;
                    }
                    //  Moves the pointer to the start of the next row.
                    p += nOffset;
                }
            }
            //  Releases the bitmap.
            pic.UnlockBits(bmData);
            //  Returns the altered bitmap.
            return pic;
}
```

1.12.8 Brightness Scrollbar

```
//  This subroutine is called when the scrollbar scrollBrightness is
//  scrolled.
private void scrollBrightness_Scroll(object sender, System.EventArgs e)
{
        //  Bitmap variable used to store the altered bitmap returned from
        //  the Brightness function.
        Bitmap temp;
        temp = Brightness((Bitmap) picInUse.Clone(), (int)scrollBrightness.Value);
        //  Sets picturebox picPreview to the altered bitmap and sizes
        //  it appropriately.
        picPreview.Image = temp;
        picPreview.SizeMode = PictureBoxSizeMode.StretchImage;
        //  Assigns the label lblValue the current value of scrollbar
        //  scrollBrightness.
        lblValue.Text = scrollBrightness.Value.ToString();
}

}
}
```

1.13 Changing Contrast Levels (frmContrast)

This form can be called from either the Update Model or Perform Recognition forms. It is used in Still Image mode to change the contrast levels of the image to be processed.

1.13.1 System References

```
//  Declaration of References
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Drawing.Imaging;
```

1.13.2 Form Items and Global Variables

```
namespace AntsNest
{
/// <summary>
/// Summary description for frmContrast.
/// </summary>
public class frmContrast : System.Windows.Forms.Form
{
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.Container components = null;
        private System.Windows.Forms.GroupBox groupBox3;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.TrackBar scrollContrast;
        private System.Windows.Forms.GroupBox groupBox2;
        private System.Windows.Forms.PictureBox picOriginal;
        private System.Windows.Forms.GroupBox groupBox1;
        private System.Windows.Forms.PictureBox picPreview;
        private System.Windows.Forms.Button cmdCancel;
        private System.Windows.Forms.Button cmdApply;
        private System.Windows.Forms.GroupBox groupBox4;
        private System.Windows.Forms.Label lblValue;
        //  Picturebox variable used as an alias for picStudy on
        //  frmRecognition or frmUpdateModel.
```

```
        private System.Windows.Forms.PictureBox PicBox;
        //  Bitmap variable used as an alias for the picture displayed
        //  in picStudy on frmRecognition or frmUpdateModel.
        private Bitmap picInUse;
```

1.13.3 Form Initialization

```
//  Startup subroutine for frmContrast
//  Takes 2 arguments: -  A reference to the image displayed in picStudy on
//                              frmRecognition or frmUpdateModel.
//                        -  A reference to picStudy itself.
public frmContrast(Bitmap pic,System.Windows.Forms.PictureBox picbox)
{
        InitializeComponent();
        //  Assign passed variables to their local counterparts.
        picInUse = pic;
        PicBox = picbox;
        //  Sets both picPreview and picOriginal to the image displayed
        //  in the picturebox picStudy on frmRecognition or frmUpdateModel
        //  and sizes it appropriately.
        picPreview.Image = picInUse;
        picPreview.SizeMode = PictureBoxSizeMode.StretchImage;
        picOriginal.Image = picInUse;
        picOriginal.SizeMode = PictureBoxSizeMode.StretchImage;
        //  Assigns the label lblValue the default value of scrollbar
        //  scrollBrightness, which is zero.
        lblValue.Text = scrollContrast.Value.ToString();
}
```

1.13.4 Form Designer Generated Code

```
/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
        if( disposing )
        {
                if(components != null)
                {
                        components.Dispose();
                }
        }
        base.Dispose( disposing );
}
```

```
#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
        //  Component code omitted.
}
#endregion
```

1.13.5 Apply Button

```
//  This subroutine is called when the 'Apply' button is clicked.
private void cmdApply_Click(object sender, System.EventArgs e)
{
        //  Sets the picturebox picStudy on frmRecognition or frmUpdateModel
        //  to the newly contrast-adjusted value and sizes it appropriately.
        //  This change only applies to the current session and is not actually
        //  saved to the image.
        picInUse = Contrast(picInUse, (sbyte)scrollContrast.Value);
        PicBox.Image = picInUse;
        PicBox.SizeMode = PictureBoxSizeMode.StretchImage;
        //  Closes the form.
        this.Close();
}
```

1.13.6 Cancel Button

```
//  This subroutine is called when the 'Cancel' button is clicked.
private void cmdCancel_Click(object sender, System.EventArgs e)
{
        //  Closes the form, without making any changes.
        this.Close();
}
```

1.13.7 Change the Contrast Level

```
//  This function is called when changes need to be made to an image's
//  contrast levels.  It returns the altered bitmap.
//  Takes 2 arguments: -  A reference to the image to be altered.
//                     -  A reference to the chosen contrast level.
public Bitmap Contrast(Bitmap pic, sbyte nContrast)
{
        //  Double variable used to temporarily store the
```

```
//  Contrast calculation.
double pixel = 0;
//  Double varialbe used to store the Contrast value
//  to be assigned to the image.  Values range from
//  0 to 2.
double contrast;
//  Checks to make sure contrast wasn't adjusted to -100,
//  because that would cause a division by zero.
if (nContrast == -100)
        contrast = 0;
else
        contrast = (100.0 + nContrast) / 100.0;
//  Contrast will always be a positive value from 0 to 4.
contrast *= contrast;
//  Integer variables used to store the Red, Green or Blue
//  value of the pixel currently being processed.
int red, green, blue;
//  Locks the region of the bitmap on which processing is to be performed.
//  In this case, the whole bitmap is processed.
BitmapData bmData = pic.LockBits(new Rectangle
        (0, 0, pic.Width, pic.Height), ImageLockMode.ReadWrite,
        PixelFormat.Format24bppRgb);
//  Sets how wide a bitmap scanline is in bytes and
//  the location in memory where to start reading.
//  Because this is a 24 bits per pixel bitmap, each
//  pixel uses 3 bytes to store its data.  So the stride
//  will basically be (bitmap width) * 3.
int stride = bmData.Stride;
System.IntPtr Scan0 = bmData.Scan0;
 //  Allows the use of pointers.
unsafe
{
        //  Creates a pointer to the spot where to start
        //  reading the bitmap.
        byte * p = (byte *)(void *)Scan0;
        //  Sets it up so we start working at the start of
        //  the first pixel of each row.
        int nOffset = stride - pic.Width*3;
        //  Run through all the pixels in the bitmap.
        for(int y=0;y<pic.Height;++y)
        {
                for(int x=0; x < pic.Width; ++x )
                {
                        //  Assigns the Blue, Green and Red Values of the
                        //  current pixel to integer variables.  Note the
                        //  order, the values are returned as BGR and not
```

```
//  RGB.
blue = p[0];
green = p[1];
red = p[2];
//  Determines exactly how red the pixel is on a
//  scale of 0 to 1.
pixel = red/255.0;
//  Subtract the median value of the scale.  If this
//  value is not adjusted, the final red value will not
//  be scaled correctly when multiplied with the
//  Contrast value (0 - 4).
pixel -= 0.5;
//  Adjust the red value to reflect the Contrast change.
pixel *= contrast;
//  Add the median back to the red value.
pixel += 0.5;
//  Return the red value to a value from 0 to 255
pixel *= 255;
//  Makes sure the value didn't jump out of the 0 to
//  255 byte scale.
if (pixel < 0) pixel = 0;
if (pixel > 255) pixel = 255;
//  Assign the contrast-adjusted red value back to the
//  pixel.
p[2] = (byte) pixel;
//  Determines exactly how green the pixel is on a
//  scale of 0 to 1.
pixel = green/255.0;
//  Subtract the median value of the scale.  If this
//  value is not adjusted, the final green value will not
//  be scaled correctly when multiplied with the
//  Contrast value (0 - 4).
pixel -= 0.5;
//  Adjust the green value to reflect the Contrast
//  change.
pixel *= contrast;
//  Add the median back to the green value.
pixel += 0.5;
//  Return the green value to a value from 0 to 255
pixel *= 255;
//  Makes sure the value didn't jump out of the 0 to
//  255 byte scale.
if (pixel < 0) pixel = 0;
if (pixel > 255) pixel = 255;
//  Assign the contrast-adjusted green value back to
//  the pixel.
```

```
                        p[1] = (byte) pixel;
                        //  Determines exactly how blue the pixel is on a
                        //  scale of 0 to 1.
                        pixel = blue/255.0;
                        //  Subtract the median value of the scale.  If this
                        //  value is not adjusted, the final blue value will not
                        //  be scaled correctly when multiplied with the
                        //  Contrast value (0 - 4).
                        pixel -= 0.5;
                        //  Adjust the blue value to reflect the Contrast
                        //  change.
                        pixel *= contrast;
                        //  Add the median back to the blue value.
                        pixel += 0.5;
                        //  Return the blue value to a value from 0 to 255
                        pixel *= 255;
                        //  Makes sure the value didn't jump out of the 0 to
                        //  255 byte scale.
                        if (pixel < 0) pixel = 0;
                        if (pixel > 255) pixel = 255;
                        //  Assign the contrast-adjusted blue value back to the
                        // pixel.
                        p[0] = (byte) pixel;
                        //  Sets the pointer to the start of the next pixel.
                        p += 3;
                    }
                    //  Moves the pointer to the start of the next row.
                    p += nOffset;
                }
            }
            //  Releases the bitmap.
            pic.UnlockBits(bmData);
             //  Returns the altered bitmap.
            return pic;
}
```

1.13.8 Contrast Scrollbar

```
//  This subroutine is called when the scrollbar scrollContrast is
//  scrolled.
private void scrollContrast_Scroll(object sender, System.EventArgs e)
{
        //  Bitmap variable used to store the altered bitmap returned from
        //  the Contrast function.
        Bitmap temp;
        temp = Contrast((Bitmap) picInUse.Clone(), (sbyte)scrollContrast.Value);
```

```
            //  Sets picturebox picPreview to the altered bitmap and sizes
            //  it appropriately.
            picPreview.Image = temp;
            picPreview.SizeMode = PictureBoxSizeMode.StretchImage;
            //  Assigns the label lblValue the current value of scrollbar
            //  scrollContrast.
            lblValue.Text = scrollContrast.Value.ToString();
        }

    }
    }
```

1.14 Picture Display (frmPicDisplay)

This form is called whenever the Show In Own Window option is selected on any of the picture context menus in the program. It displays the image in a 640 x 480 window.

1.14.1 System References

```
//  Declaration of References
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
```

1.14.2 Form Items and Global Variables

```
namespace AntsNest
{
/// <summary>
/// Summary description for frmPicDisplay1.
/// </summary>
public class frmPicDisplay1 : System.Windows.Forms.Form
{
        private System.Windows.Forms.PictureBox picImage;
        private System.Windows.Forms.ContextMenu mnuPicDisplay;
        private System.Windows.Forms.MenuItem mnuPicDisplay_SaveAs;
        private System.Windows.Forms.SaveFileDialog sfdPicDisplay;
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.Container components = null;
        //  Bitmap variable used to store the bitmap image received from the
        //  caller.
        private Bitmap bmpInUse;
```

1.14.3 Form Initialization

```
//  Startup subroutine for frmPicDisplay1.
//  Takes 1 argument:  A reference to the image to be displayed.
public frmPicDisplay1(Bitmap a)
{
        InitializeComponent();
        //  Creates a clone of the image to be displayed.
        bmpInUse = (Bitmap) a.Clone();
```

```
        //  Displays the image.
        picImage.Image = bmpInUse;
        picImage.SizeMode = PictureBoxSizeMode.StretchImage;

}
```

1.14.4 Form Designer Generated Code

```
/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
        if( disposing )
        {
                if(components != null)
                {
                        components.Dispose();
                }
        }
        base.Dispose( disposing );
}

#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
        //  Component code omitted.
}
#endregion
```

1.14.5 Click on Image

```
//  This subroutine is called when the user clicks on the image.
private void picImage_MouseDown(object sender,
                System.Windows.Forms.MouseEventArgs e)
{
        //  Determines whether the right mouse button was clicked.  If it was;
        //  a context menu is displayed.
        if (e.Button == System.Windows.Forms.MouseButtons.Right)
        {
                System.Drawing.Point CurrentMouse = new
                        System.Drawing.Point();
```

```
                CurrentMouse.X = e.X;
                CurrentMouse.Y = e.Y;
                mnuPicDisplay.Show(picImage, CurrentMouse);
        }
}
```

1.14.6 Save Image

```
//  This subroutine is called when the user selects the 'Save As" option
//  in the context menu.  It calls a Save File Dialog Box.
private void mnuPicDisplay_SaveAs_Click_1(object sender,
                                                System.EventArgs e)
{
        sfdPicDisplay.Filter = "JPEG Files (*.jpg)|*.jpg";
        sfdPicDisplay.FileName = "";
        if(sfdPicDisplay.ShowDialog() == DialogResult.OK)
        {
              bmpInUse.Save(sfdPicDisplay.FileName,
                        System.Drwing.Imaging.ImageFormat.Jpeg);
        }
}

}
}
```

2. Code Pages

2.1 Main Edge Detector (MainEdgeDetectionAgent)

This agent is used to perform the first level edge detection on the image.

2.1.1 System References

```
//  Declaration of References
using System;
using System.Data;
using System.Drawing;
using System.Collections;
using System.Drawing.Imaging;
using System.ComponentModel;
```

2.1.2 Global Variables

```
namespace AntsNest
{
/// <summary>
/// Summary description for MainEdgeDetectionAgent.
/// </summary>
public class MainEdgeDetectionAgent
{
      //  Integer variables used to store the dimensions of the
      //  image to be processed.
      private int picWidth, picHeight;
      //  Instance of the Whiteboard class, used as a refererence to the
      //  shared Whiteboard object.
      private Whiteboard WB;
```

2.1.3 Constructor

```
//  Startup subroutine (Constructor) for the MainEdgeDetectionAgent.
//  Takes 3 arguments: - 2 references to the dimensions of the image to be
//                       processed.
//                     - A reference to the shared Whiteboard object.
public MainEdgeDetectionAgent(int picwidth, int picheight,
                              Whiteboard whiteboard)
{
        picWidth = picwidth;
        picHeight = picheight;
        WB = whiteboard;
}
```

2.1.4 Wait for Resources

```
//  This subroutine keeps the agent running until the necessary
//  resource is freed.
public void StartWait()
{
        while (WB.RunStartWait == 0)
        {
                if (WB.counter == 1)
                {
                        MainEdgeDetection();
                }
        }
}
```

2.1.5 Edge Detection Subroutine

```
//  It works by finding the greatest difference between pixel colour
//  values in a 3x3 matrix.
public void MainEdgeDetection()
{

        //  This value will serve as a reference.  Any edge value lower than
        //  this, will be forced to Zero.
        int Threshold = 30;
        //  Sets how wide a bitmap scanline is in bytes and
        //  the locations in memory where to start reading
        //  the original and its copy.
        //  Because this is a 24 bits per pixel bitmap, each
        //  pixel uses 3 bytes to store its data.  So the stride
        //  will basically be (bitmap width) * 3.
        int stride = WB.bmData.Stride;
        System.IntPtr Scan0 = WB.bmData.Scan0;
        System.IntPtr Scan02 = WB.bmData2.Scan0;
        //  Allows the use of pointers.
        unsafe
        {
                //  Creates a pointer to the spots where to start
                //  reading the bitmap and its copy.
                byte * p = (byte *)(void *)Scan0;
                byte * p2 = (byte *)(void *)Scan02;
                //  Sets it up so that we start working at the start of
                //  the first pixel of each row.
                int Offset = stride - picInUse.Width*3;
                //  Makes sure that every byte in a row is processed.
                //  That means the same results will be applied to the
```

```
//  Red, Green and Blue values of each pixel.
int Width = picInUse.Width * 3;
//  Integer variables used to temporarily store the
//  the median between 2 pixels' colour values.
int Pixel = 0;
int PixelMax = 0;
//  Sets it up so the image is processed from its
//  second row.  This is crucial, because the process
//  makes use of a 3x3 pixel colour matrix.
p += stride;
p2 += stride;
//  Runs through all of the bytes of the image.
for(int y=1;y<picInUse.Height-1;++y)
{
        //  Sets it up so the image is processed from the
        //  second pixel in each row.  This is crucial, because
        //  the process makes use of a 3x3 pixel colour matrix.
        //  In other words, this sets
        //  up the pointer to point to the following area in the
        //  matrix.
        //  _____
        //  |  |  |  |
        //  _____
        //  |  |x |  |
        //  _____
        //  |  |  |  |
        //  _____

        //  The copy will be run through for its colour values,
        //  with the changes being applied to the original
        //  bitmap.
        p += 3;
        p2 += 3;
        for(int x=3; x < Width-3; ++x )
        {
                //  Checks the following areas in the matrix and
                //  subtracts the bottom one from the top one.
                //  Results are converted to positive.
                //  _____
                //  |  |  |x |
                //  _____
                //  |  |  |  |
                //  _____
                //  |x |  |  |
                //  _____
                PixelMax = Math.Abs((p2 - stride + 3)[0] - (p2+stride-
```

```
                    3)[0]);
//  Checks the following areas in the matrix and
//  stubtracts the top one from the bottom one.
//  Results are converted to positive.
//  _____
//  |x |  |  |
//  _____
//  |  |  |  |
//  _____
//  |  |  |x |
//  _____
Pixel = Math.Abs((p2 + stride + 3)[0] - (p2 - stride –
          3)[0]);
//  Makes sure that PixelMax is the highest of the two
//  values.
if (Pixel>PixelMax)
          PixelMax = Pixel;
//  Checks the following areas in the matrix and
//  subtracts the bottom one from the top one.  Results
//  are converted to positive.
//  _____
//  |  |x |  |
//  _____
//  |  |  |  |
//  _____
//  |  |x |  |
//  _____
Pixel = Math.Abs((p2 - stride)[0] - (p2 + stride)[0]);
   //  Makes sure that PixelMax is still the highest
//  value.
if (Pixel>PixelMax)
          PixelMax = Pixel;
//  Checks the following areas in the matrix and
//  subtracts the left one from the right one.  Results
//  are converted to positive.
//  _____
//  |  |  |  |
//  _____
//  |x |  |x |
//  _____
//  |  |  |  |
//  _____
Pixel = Math.Abs((p2+3)[0] - (p2 - 3)[0]);
 //  Makes sure that PixelMax is still the highest value.
if (Pixel>PixelMax)
          PixelMax = Pixel;
```

```
                                        //  Makes sure that PixelMax is higher than the
                                        //  specified threshold,
                                        //  if it's not, set it to zero.
                                        If (PixelMax < Threshold)
                                                PixelMax = 0;
                                         //  Sets the middlepoint of the matrix to the value of
                                        //  PixelMax, on the original bitmap.
                                        p[0] = (byte) PixelMax;
                                       // Sets p and p2 to the byte.
                                       ++ p;
                                       ++ p2;
                                }
                       //  Moves both pointers to the second pixel of the next row.
                        p += 3 + Offset;
                        p2 += 3 + Offset;
                        }
                }
                //  Increments the variable to let the next agent know that it may process
                //  the image.
                lock(this)
                {
                        WB.Increment();
                }
}

}
}
```

2.2 Invert Colours (InvertColoursAgent)

This agent is used to convert light to dark colours and dark to light colours.

2.2.1 System References

```
//  Declaration of References
using System;
using System.Data;
using System.Drawing;
using System.Collections;
using System.Drawing.Imaging;
using System.ComponentModel;
```

2.2.2 Global Variables

```
namespace AntsNest
{
/// <summary>
/// Summary description for InvertColorsAgent.
/// </summary>
public class InvertColoursAgent
{
      //  Integer variables used to store the dimensions of the
      //  image to be processed.
      private int picWidth, picHeight;
      //  Instance of the Whiteboard class, used as a refererence to the
      //  shared Whiteboard object.
      private Whiteboard WB;
```

2.2.3 Constructor

```
//  Startup subroutine (Constructor) for the InvertColoursAgent.
//  Takes 3 arguments: - 2 references to the dimensions of the image to be
//                       processed.
//                     - A reference to the shared Whiteboard object.
public InvertColoursAgent(int picwidth, int picheight,
                          Whiteboard whiteboard)
{
        picWidth = picwidth;
        picHeight = picheight;
        WB = whiteboard;
}
```

2.2.4 Wait for Resources

```
//  This subroutine keeps the agent running until the necessary
//  resource is freed.
public void StartWait()
{
        while (WB.RunStartWait == 0)
        {
                if (WB.counter == 1)
                {
                        InvertColours();
                }
        }
}
```

2.2.5 Invert Colours Subroutine

```
//  This subroutine does the actual colour inversion.
private void InvertColours()
{

        //  Sets how wide a bitmap scanline is in bytes and
        //  the location in memory where to start reading
        //  the bitmap.
        //  Because this is a 24 bits per pixel bitmap, each
        //  pixel uses 3 bytes to store its data.  So the stride
        //  will basically be (bitmap width) * 3.
        int stride = WB.bmData.Stride;
        System.IntPtr Scan0 = WB.bmData.Scan0;
        //  Allows the use of pointers.
        unsafe
        {
                //  Creates a pointer to the spot where to start
                //  reading the bitmap.
                byte * p = (byte *)(void *)Scan0;
                //  Sets it up so that we start working at the start of
                //  the first pixel of each row.
                int Offset = stride - picInUse.Width * 3;
                //  Makes sure that every byte in a row is processed.
                //  That means the same results will be applied to the
                //  Red, Green and Blue values of each pixel.
                int Width = picInUse.Width * 3;
                //  Runs through all of the bytes of the image.
                for(int y = 0;y < picInUse.Height ;++y)
                {
                        for(int x = 0; x < Width; ++x )
```

```
                        {
                                //  Sets the current byte value to it's mirror value
                                //  on the 0 to 255 scale, e.g. 255 - 0 = 255, thus light
                                //  becomes dark and 255 - 255 = 0, thus dark
                                //  becomes light.
                                p[0] = (byte)(255-p[0]);
                                //  Moves p to the next byte.
                                ++p;
                        }
                        //  Moves the pointer to the start of the next row.
                        p += Offset;
                }
        }
        //  Increments the variable to let the next agent know that it may process
        //  the image.
        lock(this)
        {
                WB.Increment();
        }
}

}
}
```

2.3 Convert Image to Binary (ToBinaryAgent)

This agent converts any light colour to white and any dark colour to black.

2.3.1 System References

```
//  Declaration of References
using System;
using System.Data;
using System.Drawing;
using System.Collections;
using System.Drawing.Imaging;
using System.ComponentModel;
```

2.3.2 Global Variables

```
namespace AntsNest
{
/// <summary>
/// Summary description for ToBlackAndWhiteAgent.
/// </summary>
public class ToBinaryAgent
{
      //  Integer variables used to store the dimensions of the
      //  image to be processed.
      private int picWidth, picHeight;
      //  Instance of the Whiteboard class, used as a refererence to the
      //  shared Whiteboard object.
      private Whiteboard WB;
```

2.3.3 Constructor

```
//  Startup subroutine (Constructor) for the ToBinaryAgent.
//  Takes 3 arguments: - 2 references to the dimensions of the image to be
//                       processed.
//                     - A reference to the shared Whiteboard object.
public ToBinaryAgent(int picwidth, int picheight,
                     Whiteboard whiteboard)
{
        picWidth = picwidth;
        picHeight = picheight;
        WB = whiteboard;
}
```

2.3.4 Wait for Resources

```
//  This subroutine keeps the agent running until the necessary
//  resource is freed.
public void StartWait()
{
        while (WB.RunStartWait == 0)
        {
                if (WB.counter == 1)
                {
                        ToBinary();
                }
        }
}
```

2.3.3 To Binary Subroutine

```
//  This subroutine converts the image to binary.
private void ToBinary()
{
        //  This value will serve as a reference.  Any value lower than this will
        //  be forced to zero. Values higher or equal will be set to 255,
        //  resulting in a binary image.
        int Threshold = 220;
        //  Sets how wide a bitmap scanline is in bytes and
        //  the location in memory where to start reading
        //  the bitmap.
        //  Because this is a 24 bits per pixel bitmap, each
        //  pixel uses 3 bytes to store its data.  So the stride
        //  will basically be (bitmap width) * 3.
        int stride = WB.bmData.Stride;
        System.IntPtr Scan0 = WB.bmData.Scan0;
         //  Allows the use of pointers.
        unsafe
        {
                //  Creates a pointer to the spot where to start
                //  reading the bitmap.
                byte * p = (byte *)(void *)Scan0;
                //  Sets it up so that we start working at the start of
                //  the first pixel of each row.
                int Offset = stride - picInUse.Width*3;
                //  Byte variables used to store the Red, Green or Blue
                //  value of the pixel currently being processed.
                byte red, green, blue;
               //  Byte variable used to temporarily store the average of
                //  a pixel's Blue, Green and Red values.
```

```
                byte binary;
                //  Runs through all of the pixels in an image.
                for(int y=0;y < picInUse.Height;++y)
                {
                        for(int x=0; x < picInUse.Width; ++x )
                        {
                                //  Assigns the individual color values of each pixel
                                //  to byte variables.
                                blue = p[0];
                                green = p[1];
                                red = p[2];
                                //  Finds the average of the three color values.
                                binary = (byte) ((blue + green + red) * 0.333);
                                //  If the average is less than the threshold, all
                                //  the color values get set to zero, else they all
                                //  get set to 255.
                                if (binary < Threshold)
                                        p[0] = p[1] = p[2] = 0;
                                else
                                        p[0] = p[1] = p[2] = 255;

                                //  Sets the pointer the next pixel.
                                p += 3;
                        }
                        //  Moves the pointer to the start of the next row.
                        p += Offset;
                }
        }
        //  Increments the variable to let the next agent know that it may process
        //  the image. Also makes a clone of the image at this stage of processing.
        lock(this)
        {
                WB.picInUseTB = (Bitmap)WB.picInUse.Clone();
                WB.Increment();
        }
}

}
}
```

2.4 Find Perimeter (PerimeterAgent)

This agent calculates and draws the object's perimeter as well as calculating the area, greatest x-axis and y-axis difference.

2.4.1 System References

```
//  Declaration of References.
using System;
using System.Data;
using System.Drawing;
using System.Collections;
using System.Drawing.Imaging;
using System.ComponentModel;
using System.Threading;
```

2.4.2 Global Variables

```
namespace AntsNest
{
/// <summary>
/// Summary description for PerimeterAgent.
/// </summary>
public class PerimeterAgent
{
        //  Bitmap variables used as aliases for the image to be processed
        //  and the Edges bitmap.
        private Bitmap picInUse, Edges;
        //  Integer variables used to store the measurements.
        private int Perimeter, Area;
        private int XDistance, YDistance;
        private int HighestX, HighestY, LowestX, LowestY;
        private int Shortest,Longest;
        //  Integer variables used to find the perimeter.
        private int StartY, EndY;
        private int StartX, EndX;
        private int CurrentX, CurrentY;
        //  Bitmap variable used to find the first perimeter pixel
        //  in each thread.
        private Bitmap Thread1Pic, Thread2Pic, Thread3Pic;
        private Bitmap Thread4Pic, Thread5Pic, Thread6Pic;
        //  Variable used to keep track of whether a perimeter pixel
        //  was found.
        private int found;
        //  Integer variables used to keep track of the bounding box.
        private int topy, bottomy,topx, bottomx;
```

```
	//  Instance of the Whiteboard class, used as a refererence to the
	//  shared Whiteboard object.
	private Whiteboard WB;
	//  Integer variable used to determine where the agent was created.
	private int Type;
```

2.4.3 Constructor

```
//  Startup subroutine (Constructor) for the PerimeterAgent.
//  Takes 6 arguments: - A reference to the shared Whiteboard object.
//                     - 4 References to the area to be processed.
//                     - A reference to where the object is created.
public PerimeterAgent(Whiteboard whiteboard,int TopX, int TopY, int BottomX,
                      int BottomY, int type)
{
	WB = whiteboard;
	//  Set the bounding box values.
	topy = TopY;
	bottomy = BottomY;
	topx = TopX;
	bottomx = BottomX;
	Type = type;
}
```

2.4.4 Wait for Resources

```
//  This subroutine keeps the agent running until the necessary
//  resource is freed.
public void StartWait()
{
	while (WB.RunStartWait == 0)
	{
		if (WB.counter == 4)
		{
			FindPerimeter(WB.picInUse,topx,topy,bottomx,bottomy);
		}
	}
}
```

2.4.5 Find Perimeter Subroutine

```
//  This subroutine is used to find the object's perimeter,area and bounding box.
//   It takes 5 variables: - A reference to the image to be processed.
//                         - References to the area to be processed.
private void FindPerimeter(Bitmap EdgeBitmap, int TopX, int TopY, int BottomX,
                           int BottomY)
{
        WB.ReleaseBitmapData();
        //  Clone the image to be processed.
        Edges = (Bitmap) EdgeBitmap.Clone();
        //  Create a new image and  clone it for
        //  the threads.
        picInUse = new Bitmap(Edges.Width, Edges.Height);
        Thread1Pic = (Bitmap) EdgeBitmap.Clone();
        Thread2Pic = (Bitmap) EdgeBitmap.Clone();
        Thread3Pic = (Bitmap) EdgeBitmap.Clone();
        Thread4Pic = (Bitmap) EdgeBitmap.Clone();
        Thread5Pic = (Bitmap) EdgeBitmap.Clone();
        Thread6Pic = (Bitmap) EdgeBitmap.Clone();
         //  Set the bounding box values.
         topy = TopY;
        bottomy = BottomY;
        topx = TopX;
        bottomx = BottomX;
        //  Integer variable used to keep track of whether
        //  the perimeter is getting too big.
        int flag = 0;
        //  Initialize the variable.
        found = 0;
        //  Variable used to keep track of the middle of the image.
        int MiddleX;
        //  Initialize the variables.
        LowestX = Edges.Width;
        LowestY = Edges.Height;
        LowestX = Edges.Width;
        LowestY = Edges.Height;
        HighestX = 0;
        HighestY = 0;
        Perimeter = 0;
        Area = 0;
        MiddleX = TopX + ((BottomX - TopX) / 2);
        StartX = EndX = 0;
        XDistance = 0;
        YDistance = 0;
        StartX = StartY = 0;
```

```
//  Integer variables used to keep track of pixels.
int Prev = 0;
int first = 0;
int last = 0;
//  Create threads to get the start pixel of the perimeter
Thread Pic1 = new Thread(new ThreadStart(FirstThreadSearch));
Thread Pic2 = new Thread(new ThreadStart(SecondThreadSearch));
Thread Pic3 = new Thread(new ThreadStart(ThirdThreadSearch));
Thread Pic4 = new Thread(new ThreadStart(FourthThreadSearch));
Thread Pic5 = new Thread(new ThreadStart(FifthThreadSearch));
Thread Pic6 = new Thread(new ThreadStart(SixthThreadSearch));
//  Name the threads.
Pic1.Name = "Pic1";
Pic2.Name = "Pic2";
Pic3.Name = "Pic3";
Pic4.Name = "Pic1";
Pic5.Name = "Pic2";
Pic6.Name = "Pic3";
//  Start the threads.
Pic1.Start();
Pic2.Start();
Pic3.Start();
Pic4.Start();
Pic5.Start();
Pic6.Start();
//  Coordinate the threads.
Pic1.Join();
Pic2.Join();
Pic3.Join();
Pic4.Join();
Pic5.Join();
Pic6.Join();
//  See if a pixel was found.
if (found == 1)
{
        //  One was found.
        //  The following piece of code will run through all of
        //  the perimeter pixels and create an image that consists of
        //  a white background with the perimeter indicated in black.
        //  Draw the first perimeter pixel.
        picInUse.SetPixel(StartX,StartY,Color.Black);
        Edges.SetPixel(StartX,StartY,Color.Black);
        //  Find and draw the next perimeter pixel.
        Perimeter++;
        if (StartX < LowestX)
                LowestX = StartX;
```

```
if (StartX > HighestX)
        HighestX = StartX;
if (StartY < LowestY)
        LowestY = StartY;
if (StartY > HighestY)
        HighestY = StartY;
if (Edges.GetPixel(StartX - 1, StartY - 1).ToArgb() ==
   Color.Black.ToArgb())
{
        Prev = 2;
        CurrentX = StartX - 1;
        CurrentY = StartY - 1;
}
else
if (Edges.GetPixel(StartX - 1, StartY).ToArgb() ==
   Color.Black.ToArgb())
{
        Prev = 3;
        CurrentX = StartX - 1;
        CurrentY = StartY;
}
else
if (Edges.GetPixel(StartX - 1, StartY + 1).ToArgb() ==
   Color.Black.ToArgb())
{
        Prev = 4;
        CurrentX = StartX - 1;
        CurrentY = StartY + 1;
}
else
if (Edges.GetPixel(StartX, StartY + 1).ToArgb() ==
   Color.Black.ToArgb())
{
        Prev = 5;
        CurrentX = StartX;
        CurrentY = StartY + 1;
}
else
if (Edges.GetPixel(StartX + 1, StartY + 1).ToArgb() ==
   Color.Black.ToArgb())
{
        Prev = 6;
        CurrentX = StartX + 1;
        CurrentY = StartY + 1;
}
else
```

```
if (Edges.GetPixel(StartX + 1, StartY).ToArgb() ==
  Color.Black.ToArgb())
{
        Prev = 7;
        CurrentX = StartX + 1;
        CurrentY = StartY;
}
else
if (Edges.GetPixel(StartX + 1, StartY - 1).ToArgb() ==
   Color.Black.ToArgb())
{
        Prev = 8;
        CurrentX = StartX + 1;
        CurrentY = StartY - 1;
}
else
if (Edges.GetPixel(StartX, StartY - 1).ToArgb() ==
   Color.Black.ToArgb())
{
        Prev = 1;
        CurrentX = StartX;
        CurrentY = StartY - 1;
}

picInUse.SetPixel(CurrentX,CurrentY,Color.Black);
Edges.SetPixel(CurrentX,CurrentY,Color.Black);
Perimeter++;
if (CurrentX < LowestX)
        LowestX = CurrentX;
if (CurrentX > HighestX)
        HighestX = CurrentX;
if (CurrentY < LowestY)
        LowestY = CurrentY;
if (CurrentY > HighestY)
        HighestY = CurrentY;
flag = 0;
//  Find and draw the rest of the perimeter pixels.
while ((flag == 0) && ((CurrentX != 0) && (CurrentX != Edges.Width
       - 1) && (CurrentY != 0) && (CurrentY != Edges.Height - 1)))
{
        if (Prev == 1)
        {
                if (Edges.GetPixel(CurrentX + 1, CurrentY –
                   1).ToArgb() == Color.Black.ToArgb())
                {
                        Prev = 8;
```

```
            CurrentX = CurrentX + 1;
            CurrentY = CurrentY - 1;
}
else
if (Edges.GetPixel(CurrentX, CurrentY - 1).ToArgb()
  == Color.Black.ToArgb())
{
            Prev = 1;
            CurrentX = CurrentX;
            CurrentY = CurrentY - 1;
}
else
if (Edges.GetPixel(CurrentX - 1, CurrentY –
  1).ToArgb() == Color.Black.ToArgb())
{
            Prev = 2;
            CurrentX = CurrentX - 1;
            CurrentY = CurrentY - 1;
}
else
if (Edges.GetPixel(CurrentX - 1, CurrentY).ToArgb()
  == Color.Black.ToArgb())
{
            Prev = 3;
            CurrentX = CurrentX - 1;
            CurrentY = CurrentY;
}
else
if (Edges.GetPixel(CurrentX - 1, CurrentY +
  1).ToArgb() == Color.Black.ToArgb())
{
            Prev = 4;
            CurrentX = CurrentX - 1;
            CurrentY = CurrentY + 1;
}
else
if (Edges.GetPixel(CurrentX, CurrentY + 1).ToArgb()
  == Color.Black.ToArgb())
{
            Prev = 5;
            CurrentX = CurrentX;
            CurrentY = CurrentY + 1;
}
else
if (Edges.GetPixel(CurrentX + 1, CurrentY +
  1).ToArgb() == Color.Black.ToArgb())
```

```
            {
                Prev = 6;
                CurrentX = CurrentX + 1;
                CurrentY = CurrentY + 1;
            }
            else
            if (Edges.GetPixel(CurrentX + 1, CurrentY).ToArgb()
              == Color.Black.ToArgb())
            {
                Prev = 7;
                CurrentX = CurrentX + 1;
                CurrentY = CurrentY;
            }
        }
        else
        if (Prev == 2)
        {
            if (Edges.GetPixel(CurrentX, CurrentY - 1).ToArgb()
               == Color.Black.ToArgb())
            {
                Prev = 1;
                CurrentX = CurrentX;
                CurrentY = CurrentY - 1;
            }
            else
            if (Edges.GetPixel(CurrentX - 1, CurrentY –
               1).ToArgb() == Color.Black.ToArgb())
            {
                Prev = 2;
                CurrentX = CurrentX - 1;
                CurrentY = CurrentY - 1;
            }
            else
            if (Edges.GetPixel(CurrentX - 1, CurrentY).ToArgb()
              == Color.Black.ToArgb())
            {
                Prev = 3;
                CurrentX = CurrentX - 1;
                CurrentY = CurrentY;
            }
            else
            if (Edges.GetPixel(CurrentX - 1, CurrentY +
              1).ToArgb() == Color.Black.ToArgb())
            {
                Prev = 4;
                CurrentX = CurrentX - 1;
```

```
                CurrentY = CurrentY + 1;
            }
            else
            if (Edges.GetPixel(CurrentX, CurrentY + 1).ToArgb()
              == Color.Black.ToArgb())
            {
                Prev = 5;
                CurrentX = CurrentX;
                CurrentY = CurrentY + 1;
            }
            else
            if (Edges.GetPixel(CurrentX + 1, CurrentY +
              1).ToArgb() == Color.Black.ToArgb())
            {
                Prev = 6;
                CurrentX = CurrentX + 1;
                CurrentY = CurrentY + 1;
            }
            else
            if (Edges.GetPixel(CurrentX + 1, CurrentY).ToArgb()
              == Color.Black.ToArgb())
            {
                Prev = 7;
                CurrentX = CurrentX + 1;
                CurrentY = CurrentY;
            }
            else
            if (Edges.GetPixel(CurrentX + 1, CurrentY –
               1).ToArgb() == Color.Black.ToArgb())
            {
                Prev = 8;
                CurrentX = CurrentX + 1;
                CurrentY = CurrentY - 1;
            }
        }
        else
        if (Prev == 3)
        {
            if (Edges.GetPixel(CurrentX - 1, CurrentY –
              1).ToArgb() == Color.Black.ToArgb())
            {
                Prev = 2;
                CurrentX = CurrentX - 1;
                CurrentY = CurrentY - 1;
            }
            else
```

```
if (Edges.GetPixel(CurrentX - 1, CurrentY).ToArgb()
  == Color.Black.ToArgb())
{
        Prev = 3;
        CurrentX = CurrentX - 1;
        CurrentY = CurrentY;
}
else
if (Edges.GetPixel(CurrentX - 1, CurrentY +
  1).ToArgb() == Color.Black.ToArgb())
{
        Prev = 4;
        CurrentX = CurrentX - 1;
        CurrentY = CurrentY + 1;
}
else
if (Edges.GetPixel(CurrentX, CurrentY + 1).ToArgb()
  == Color.Black.ToArgb())
{
        Prev = 5;
        CurrentX = CurrentX;
        CurrentY = CurrentY + 1;
}
else
if (Edges.GetPixel(CurrentX + 1, CurrentY +
  1).ToArgb() == Color.Black.ToArgb())
{
        Prev = 6;
        CurrentX = CurrentX + 1;
        CurrentY = CurrentY + 1;
}
else
if (Edges.GetPixel(CurrentX + 1, CurrentY).ToArgb()
   == Color.Black.ToArgb())
{
        Prev = 7;
        CurrentX = CurrentX + 1;
        CurrentY = CurrentY;
}
else
if (Edges.GetPixel(CurrentX + 1, CurrentY –
  1).ToArgb() == Color.Black.ToArgb())
{
        Prev = 8;
        CurrentX = CurrentX + 1;
        CurrentY = CurrentY - 1;
```

```
            }
            else
            if (Edges.GetPixel(CurrentX, CurrentY - 1).ToArgb()
                == Color.Black.ToArgb())
            {
                    Prev = 1;
                    CurrentX = CurrentX;
                    CurrentY = CurrentY - 1;
            }
    }
    else
    if (Prev == 4)
    {
            if (Edges.GetPixel(CurrentX - 1, CurrentY).ToArgb()
              == Color.Black.ToArgb())
            {
                    Prev = 3;
                    CurrentX = CurrentX - 1;
                    CurrentY = CurrentY;
            }
            else
            if (Edges.GetPixel(CurrentX - 1, CurrentY +
              1).ToArgb() == Color.Black.ToArgb())
            {
                    Prev = 4;
                    CurrentX = CurrentX - 1;
                    CurrentY = CurrentY + 1;
            }
            else
            if (Edges.GetPixel(CurrentX, CurrentY + 1).ToArgb()
              == Color.Black.ToArgb())
            {
                    Prev = 5;
                    CurrentX = CurrentX;
                    CurrentY = CurrentY + 1;
            }
            else
            if (Edges.GetPixel(CurrentX + 1, CurrentY +
              1).ToArgb() == Color.Black.ToArgb())
            {
                    Prev = 6;
                    CurrentX = CurrentX + 1;
                    CurrentY = CurrentY + 1;
            }
            else
            if (Edges.GetPixel(CurrentX + 1, CurrentY).ToArgb()
```

```
            == Color.Black.ToArgb())
        {
                Prev = 7;
                CurrentX = CurrentX + 1;
                CurrentY = CurrentY;
        }
        else
        if (Edges.GetPixel(CurrentX + 1, CurrentY –
          1).ToArgb() == Color.Black.ToArgb())
        {
                Prev = 8;
                CurrentX = CurrentX + 1;
                CurrentY = CurrentY - 1;
        }
        else
        if (Edges.GetPixel(CurrentX, CurrentY - 1).ToArgb()
          == Color.Black.ToArgb())
        {
                Prev = 1;
                CurrentX = CurrentX;
                CurrentY = CurrentY - 1;
        }
        else
        if (Edges.GetPixel(CurrentX - 1, CurrentY –
          1).ToArgb() == Color.Black.ToArgb())
        {
                Prev = 2;
                CurrentX = CurrentX - 1;
                CurrentY = CurrentY - 1;
        }
}
else
if (Prev == 5)
{
        if (Edges.GetPixel(CurrentX - 1, CurrentY +
          1).ToArgb() == Color.Black.ToArgb())
        {
                Prev = 4;
                CurrentX = CurrentX - 1;
                CurrentY = CurrentY + 1;
        }
        else
        if (Edges.GetPixel(CurrentX, CurrentY + 1).ToArgb()
          == Color.Black.ToArgb())
        {
                Prev = 5;
```

```
            CurrentX = CurrentX;
            CurrentY = CurrentY + 1;
}
else
if (Edges.GetPixel(CurrentX + 1, CurrentY +
    1).ToArgb() == Color.Black.ToArgb())
{
            Prev = 6;
            CurrentX = CurrentX + 1;
            CurrentY = CurrentY + 1;
}
else
if (Edges.GetPixel(CurrentX + 1, CurrentY).ToArgb()
   == Color.Black.ToArgb())
{
            Prev = 7;
            CurrentX = CurrentX + 1;
            CurrentY = CurrentY;
}
else
if (Edges.GetPixel(CurrentX + 1, CurrentY –
   1).ToArgb() == Color.Black.ToArgb())
{
            Prev = 8;
            CurrentX = CurrentX + 1;
            CurrentY = CurrentY - 1;
}
else
if (Edges.GetPixel(CurrentX, CurrentY - 1).ToArgb()
   == Color.Black.ToArgb())
{
            Prev = 1;
            CurrentX = CurrentX;
            CurrentY = CurrentY - 1;
}
else
if (Edges.GetPixel(CurrentX - 1, CurrentY –
    1).ToArgb() == Color.Black.ToArgb())
{
            Prev = 2;
            CurrentX = CurrentX - 1;
            CurrentY = CurrentY - 1;
}
else
if (Edges.GetPixel(CurrentX - 1, CurrentY).ToArgb()
   == Color.Black.ToArgb())
```

```
        {
                Prev = 3;
                CurrentX = CurrentX - 1;
                CurrentY = CurrentY;
        }
}
else
if (Prev == 6)
{
        if (Edges.GetPixel(CurrentX, CurrentY + 1).ToArgb()
          == Color.Black.ToArgb())
        {
                Prev = 5;
                CurrentX = CurrentX;
                CurrentY = CurrentY + 1;
        }
        else
        if (Edges.GetPixel(CurrentX + 1, CurrentY +
          1).ToArgb() == Color.Black.ToArgb())
        {
                Prev = 6;
                CurrentX = CurrentX + 1;
                CurrentY = CurrentY + 1;
        }
        else
        if (Edges.GetPixel(CurrentX + 1, CurrentY).ToArgb()
          == Color.Black.ToArgb())
        {
                Prev = 7;
                CurrentX = CurrentX + 1;
                CurrentY = CurrentY;
        }
        else
        if (Edges.GetPixel(CurrentX + 1, CurrentY –
           1).ToArgb() == Color.Black.ToArgb())
        {
                Prev = 8;
                CurrentX = CurrentX + 1;
                CurrentY = CurrentY - 1;
        }
        else
        if (Edges.GetPixel(CurrentX, CurrentY - 1).ToArgb()
          == Color.Black.ToArgb())
        {
                Prev = 1;
                CurrentX = CurrentX;
```

```
                CurrentY = CurrentY - 1;
            }
            else
            if (Edges.GetPixel(CurrentX - 1, CurrentY –
              1).ToArgb() == Color.Black.ToArgb())
            {
                Prev = 2;
                CurrentX = CurrentX - 1;
                CurrentY = CurrentY - 1;
            }
            else
            if (Edges.GetPixel(CurrentX - 1, CurrentY).ToArgb()
              == Color.Black.ToArgb())
            {
                Prev = 3;
                CurrentX = CurrentX - 1;
                CurrentY = CurrentY;
            }
            else
            if (Edges.GetPixel(CurrentX - 1, CurrentY +
              1).ToArgb() == Color.Black.ToArgb())
            {
                Prev = 4;
                CurrentX = CurrentX - 1;
                CurrentY = CurrentY + 1;
            }
        }
        else
        if (Prev == 7)
        {
            if (Edges.GetPixel(CurrentX + 1, CurrentY +
               1).ToArgb() == Color.Black.ToArgb())
            {
                Prev = 6;
                CurrentX = CurrentX + 1;
                CurrentY = CurrentY + 1;
            }
            else
            if (Edges.GetPixel(CurrentX + 1, CurrentY).ToArgb()
               == Color.Black.ToArgb())
            {
                Prev = 7;
                CurrentX = CurrentX + 1;
                CurrentY = CurrentY;
            }
            else
```

```
if (Edges.GetPixel(CurrentX + 1, CurrentY –
   1).ToArgb() == Color.Black.ToArgb())
{
        Prev = 8;
        CurrentX = CurrentX + 1;
        CurrentY = CurrentY - 1;
}
else
if (Edges.GetPixel(CurrentX, CurrentY - 1).ToArgb()
  == Color.Black.ToArgb())
{
        Prev = 1;
        CurrentX = CurrentX;
        CurrentY = CurrentY - 1;
}
else
if (Edges.GetPixel(CurrentX - 1, CurrentY –
   1).ToArgb() == Color.Black.ToArgb())
{
        Prev = 2;
        CurrentX = CurrentX - 1;
        CurrentY = CurrentY - 1;
}
else
if (Edges.GetPixel(CurrentX - 1, CurrentY).ToArgb()
  == Color.Black.ToArgb())
{
        Prev = 3;
        CurrentX = CurrentX - 1;
        CurrentY = CurrentY;
}
else
if (Edges.GetPixel(CurrentX - 1, CurrentY +
   1).ToArgb() == Color.Black.ToArgb())
{
        Prev = 4;
        CurrentX = CurrentX - 1;
        CurrentY = CurrentY + 1;
}
else
if (Edges.GetPixel(CurrentX, CurrentY + 1).ToArgb()
  == Color.Black.ToArgb())
{
        Prev = 5;
        CurrentX = CurrentX;
        CurrentY = CurrentY + 1;
```

```
        }
}
else
if (Prev == 8)
{
        if (Edges.GetPixel(CurrentX + 1, CurrentY).ToArgb()
          == Color.Black.ToArgb())
        {
                Prev = 7;
                CurrentX = CurrentX + 1;
                CurrentY = CurrentY;
        }
        else
        if (Edges.GetPixel(CurrentX + 1, CurrentY –
           1).ToArgb() == Color.Black.ToArgb())
        {
                Prev = 8;
                CurrentX = CurrentX + 1;
                CurrentY = CurrentY - 1;
        }
        else
        if (Edges.GetPixel(CurrentX, CurrentY –
           1).ToArgb() == Color.Black.ToArgb())
        {
                Prev = 1;
                CurrentX = CurrentX;
                CurrentY = CurrentY - 1;
        }
        else
        if (Edges.GetPixel(CurrentX - 1, CurrentY –
          1).ToArgb() == Color.Black.ToArgb())
        {
                Prev = 2;
                CurrentX = CurrentX - 1;
                CurrentY = CurrentY - 1;
        }
        else
        if (Edges.GetPixel(CurrentX - 1, CurrentY).ToArgb()
          == Color.Black.ToArgb())
        {
                Prev = 3;
                CurrentX = CurrentX - 1;
                CurrentY = CurrentY;
        }
        else
        if (Edges.GetPixel(CurrentX - 1, CurrentY +
```

```
                    1).ToArgb() == Color.Black.ToArgb())
                {
                        Prev = 4;
                        CurrentX = CurrentX - 1;
                        CurrentY = CurrentY + 1;
                }
                else
                if (Edges.GetPixel(CurrentX, CurrentY + 1).ToArgb()
                    == Color.Black.ToArgb())
                {
                        Prev = 5;
                        CurrentX = CurrentX;
                        CurrentY = CurrentY + 1;
                }
                else
                if (Edges.GetPixel(CurrentX + 1, CurrentY +
                    1).ToArgb() == Color.Black.ToArgb())
                {
                        Prev = 6;
                        CurrentX = CurrentX + 1;
                        CurrentY = CurrentY + 1;
                }
        }
        picInUse.SetPixel(CurrentX,CurrentY,Color.Black);
        Edges.SetPixel(CurrentX,CurrentY,Color.Black);
        Perimeter++;
        if (CurrentX < LowestX)
                LowestX = CurrentX;
        if (CurrentX > HighestX)
                HighestX = CurrentX;
        if (CurrentY < LowestY)
                LowestY = CurrentY;
        if (CurrentY > HighestY)
                HighestY = CurrentY;
        if ((CurrentX < StartX + 1) && (CurrentX > StartX - 1))
                if ((CurrentY < StartY + 5) && (CurrentY > StartY - 5))
                        flag = 1;
        if (Perimeter > ((Edges.Width * Edges.Height) / 4))
        {
                flag = 1;
                Perimeter = 0;
        }
}
```

```
        //  See if a perimeter was found.  If there was, calculate
        //  the highest distance between the X and Y pixels and
        //  also calculate the area of the object.
        if ((CurrentX == 0) || (CurrentX == Edges.Width - 1) ||
                (CurrentY == 0) || (CurrentY == Edges.Height - 1))
                Perimeter = 0;

        if (Perimeter != 0)
        {
                for (int y = LowestY; y <= HighestY; y++)
                {
                        flag = 0;
                        first = 0;
                        last = 0;
                        for (int x = LowestX; x <= HighestX; x++)
                        {
                                if (Edges.GetPixel(x, y).ToArgb() ==
                                   Color.Black.ToArgb())
                                {
                                        if (flag == 0)
                                        {
                                                first = x;
                                                flag = 1;
                                        }
                                                last = x;
                                }
                        }

                        Area = Area + ((last - first) + 1);
                }
                lock(this)
                {
                        YDistance = HighestY - LowestY;
                        XDistance = HighestX - LowestX;
                }
        }
}
//  Increments the variable to let the next agent know that it may process
//  the image. Also makes a copy of the image at this stage of processing.
 lock(this)
{
        WB.picInUseDone = picInUse;
        WB.Increment();
         if (Type == 1)
           WB.Increment();
}}
```

2.4.5 Thread Functions

```
//  Find the first perimeter pixel on the first thread.
//  Halt all other threads if found.
private void FirstThreadSearch()
{
        int x = Thread1Pic.Width / 4 * 2;
        for (int y = topy; (y < bottomy) && (found == 0); y++)
                if (Thread1Pic.GetPixel(x,y).ToArgb() ==
                   Color.Black.ToArgb())
                {
                        lock(this)
                        {
                                StartX = x;
                                StartY = y;
                                found = 1;
                        }
                }
}

//  Find the first perimeter pixel on the second thread.
//  Halt all other threads if found.
private void SecondThreadSearch()
{
        int x = Thread2Pic.Width / 4;
        for (int y = topy; (y < bottomy) && (found == 0); y++)
                if (Thread2Pic.GetPixel(x,y).ToArgb() == Color.Black.ToArgb())
                {
                        lock(this)
                        {
                                StartX = x;
                                StartY = y;
                                found = 1;
                        }
                }
}

//  Find the first perimeter pixel on the third thread.
//  Halt all other threads if found.
private void ThirdThreadSearch()
{
        int x = Thread3Pic.Width / 4 * 3;
        for (int y = topy; (y < bottomy) && (found == 0); y++)
                if (Thread3Pic.GetPixel(x,y).ToArgb() == Color.Black.ToArgb())
                {
```

```
                        lock(this)
                        {
                                StartX = x;
                                StartY = y;
                                found = 1;
                        }
                }
}

//  Find the first perimeter pixel on the fourth thread.
//  Halt all other threads if found.
private void FourthThreadSearch()
{
        int y = Thread4Pic.Height / 4 * 2;
        for (int x = topx; (x < bottomx) && (found == 0); x++)
                if (Thread4Pic.GetPixel(x,y).ToArgb() == Color.Black.ToArgb())
                {
                        lock(this)
                        {
                                StartX = x;
                                StartY = y;
                                found = 1;
                        }
                }
}

//  Find the first perimeter pixel on the fifth thread.
//  Halt all other threads if found.
private void FifthThreadSearch()
{
        int y = Thread5Pic.Height / 4;
        for (int x = topx; (x < bottomx) && (found == 0); x++)
                if (Thread5Pic.GetPixel(x,y).ToArgb() == Color.Black.ToArgb())
                {
                        lock(this)
                        {
                                StartX = x;
                                StartY = y;
                                found = 1;
                        }
                }
}
```

```
//  Find the first perimeter pixel on the sixth thread.
//  Halt all other threads if found.
private void SixthThreadSearch()
{
        int y = Thread6Pic.Height / 4 * 3;
        for (int x = topx; (x < bottomx) && (found == 0); x++)
                if (Thread6Pic.GetPixel(x,y).ToArgb() == Color.Black.ToArgb())
                {
                        lock(this)
                        {
                                StartX = x;
                                StartY = y;
                                found = 1;
                        }
                }
}
```

2.4.6 Return picInUse

```
//  This funtion returns the generate Perimeter bitmap.
public Bitmap getpicInUse()
{
        return picInUse;
}
```

2.4.7 Return Perimeter

```
//  This function returns the perimeter.
public int getPerimeter()
{
        return Perimeter;
}
```

2.4.8 Return Area

```
//  This function returns the area.
public int getArea()
{
        return Area;
}
```

2.4.9 Return XDistance

```
//  This function returns the greatest X Distance
//  between pixels.
public int getXDistance()
{
        return XDistance;
}
```

2.4.10 Return YDistance

```
//  This function returns the greatest Y Distance
//  between pixels.
public int getYDistance()
{
        return YDistance;
}
```

2.4.11 Return LowestX

```
//  This function returns the lowest x value of
//  the bounding box.
public int getLowestX()
{
        return LowestX;
}
```

2.4.12 Return LowestY

```
//  This function returns the lowest y value of
//  the bounding box.
public int getLowestY()
{
        return LowestY;
}
```

2.4.13 Return HighestX

```
//  This function returns the highest x value of
//  the bounding box.
public int getHighestX()
{
        return HighestX;
}
```

2.4.14 Return HighestY

```
//  This function returns the highest y value of
//  the bounding box.
public int getHighestY()
{
        return HighestY;
}
}
}
```

2.5 Create EdgeGraph (EdgeGraphAgent)

This agent draws the Edge Graph for a specific object.

2.5.1 System References

```
//  Declaration of References
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Collections;
```

2.5.2 Global Variables

```
namespace AntsNest
{
/// <summary>
/// Summary description for EdgeGraphAgent.
/// </summary>
public class EdgeGraphAgent
{
        //  String variable used to store the generated EdgeGraph string.
        string GraphString;
        //  Bitmap variable to store the newly created EdgeGraph.
        Bitmap Graph;
        //  ArrayList used to keep track of all the generated EdgeGraphs.
        ArrayList Graphs = new ArrayList();
        //  Integer variable used to keep track of the number of edges
        //  in the EdgeGraph.
        int EdgeCount;
        //  Integer variables used to keep track of the bounding box.
        private int topy, bottomy,topx, bottomx;
        //  Instance of the Whiteboard class, used as a refererence to the
        //  shared Whiteboard object.
        private Whiteboard WB;
```

2.5.3 Constructor

```
//  Startup subroutine (Constructor) for the EdgeGraphAgent.
//  Takes 5 arguments: - A reference to the shared Whiteboard object.
//                     - References to the object's bounding box.
public EdgeGraphAgent(Whiteboard whiteboard, int TopY, int BottomY, int TopX,
                      int BottomX)
{
        WB = whiteboard;
        topy = TopY;
```

```
        bottomy = BottomY;
        topx = TopX;
        bottomx = BottomX;
}
```

2.5.4 Wait for Resources

```
//  This subroutine keeps the agent running until the necessary
//  resource is freed.
public void StartWait()
{
        while (WB.RunStartWait == 0)
        {
                if (WB.counter == 6)
                  CreateEdgeGraph(WB.picInUseTB,topy,bottomy,topx,bottomx);
        }
}
```

2.5.5 Create Edge Graph Subroutine

```
public void CreateEdgeGraph(Bitmap a, int TopY, int BottomY, int TopX, int
                                    BottomX)
{
        //  Integer variable used to keep track of the middle of the
        //  bounding box.
        int MiddleX;
        //  String variables used to process the EdgeGraph.
        string holdchar,tempGraphString;
        //  Integer variables used to process the EdgeGraph.
        int Count,CountZeroes;
        //  Integer variable used to store the last X value.
        int BottomLine = a.Width - 1;
        //  Initialize the variables.
        tempGraphString = "";
        EdgeCount = 0;
         //  Create an empty bitmap on which to draw.
        Graph = new Bitmap(a.Width,a.Height);
        Graph.RotateFlip(RotateFlipType.Rotate90FlipNone);
        //  Run through the middle of the bounding box.  Determinine
        //  which pixels are background and which are edges.  Write the
        //  values to a string, with 1's signifying edges and o's background.
        MiddleX = TopX + ((BottomX - TopX) / 2);
        tempGraphString = "";
        if (a.GetPixel(MiddleX,0).ToArgb() == Color.Black.ToArgb())
                GraphString = "1";
        else
```

```
        GraphString = "0";
for (int y = 1; y < TopY; y++)
        GraphString = GraphString + "0";
        for (int y = TopY; y <= BottomY; y++)
        {
                if (a.GetPixel(MiddleX,y).ToArgb() == Color.Black.ToArgb())
                        GraphString = GraphString + "1";
                else
                        GraphString = GraphString + "0";
        }
for (int y = (BottomY + 1); y <= a.Height; y++)
        GraphString = GraphString + "0";

//  Run through the GraphString.  Draw a line when the string
//  consists of 1's and move further along the image when the
//  string consists of 0's.  It also shortens the GraphString
//  to an E for an Edge and the number of pixels followed by a
//  Z for background.
Count = 0;
CountZeroes = 0;
for (int i = 0; i < a.Height; i++)
{
        holdchar = GraphString.Substring(i,1);
        if (holdchar == "0")
        {
                if (Count != 0)
                {
                        tempGraphString = tempGraphString + "E";
                        for (int j = BottomLine - 1; j > (BottomLine - (Count *
                             6)); j--)
                                Graph.SetPixel((i - (Count / 2)),j, Color.Red);
                }
                Count = 0;
                CountZeroes++;
        }
        if (holdchar == "1")
        {
                Count++;
                if (CountZeroes > 0)
                {
                        tempGraphString = tempGraphString +
                            CountZeroes.ToString() + "Z";
                        EdgeCount++;
                }
        CountZeroes = 0;
        }
```

```
                Graph.SetPixel(i,BottomLine,Color.Red);
        }
        WB.Increment();
}
```

2.5.6 Return Graphs

```
//  This function returns the Graphs ArrayList.
public ArrayList getGraphs()
{
        return Graphs;
}
```

2.5.7 Return Graph

```
//  This function returns the EdgeGraph.
public Bitmap getGraph()
{
        return Graph;
}
```

2.5.8 Return EdgeCount

```
//  This function returns the number of edges on the EdgeGraph.
public int getEdgeCount()
{
        return EdgeCount;
}

}
}
```

2.6 Remove Background (RemoveOutsideBackground)

This class sets the area of an image, surrounding the bounding box, to white.

2.6.1 System References

```
 //  Declaration of References
using System;
using System.Drawing;
```

2.6.2 Global Variables

```
namespace AntsNest
{
// <summary>
// Summary description for RemoveOutsideBackgroundAgent.
// </summary>
public class RemoveOutsideBackgroundAgent
{
        //  Bitmap variable used as an alias for the image to be
        //  processed.
        private Bitmap picInUse;
```

2.6.3 Constructor

```
//  Startup subroutine (Constructor) for the RemoveOutsideBackgroundAgent.
//  Takes 5 arguments: - A reference to the image to be processed.
//                     - A reference to the left-most x-coordinate of the object's
//                       bounding box.
//                     - A reference to the right-most x-coordinate of the object's
//                       bounding box.
//                     - A reference to the top-most y-coordinate of the object's
//                       bounding box.
//                      - A reference to the bottom-most y-coordinate of the
//                        object's bounding box.
public RemoveOutsideBackgroundAgent(Bitmap a,int topX, int bottomX, int topY,
                                                    int bottomY)
{
        //  Assigns the passed variable to its local counterpart
        picInUse = a;
        //  Runs through every pixel in the image.
        //  Sets pixels to the left of topX, to the right
        //  of bottomX, to the top of topY and to the bottom of
        //  bottomY to white.
        for (int i = 0; i < picInUse.Width; i++)
                for(int j = 0; j < picInUse.Height; j++)
```

```
                {
                        if (j <= topY)
                                picInUse.SetPixel(i,j,System.Drawing.Color.White);
                        if (i >= bottomX)
                                picInUse.SetPixel(i,j,System.Drawing.Color.White);
                        if (i <= topX)
                                picInUse.SetPixel(i,j,System.Drawing.Color.White);
                        if (j >= bottomY)
                                picInUse.SetPixel(i,j,System.Drawing.Color.White);
                }
}
```

2.6.4 Return picInUse

```
//  This function returns the altered bitmap.
public Bitmap getpicInUse()
{
        return picInUse;
}

}
}
```

2.7 Trace Perimeter of Image (Overlay)

This class overlays an object's perimeter or edges onto the original image. The image is converted to greyscale and the overlay is done in red.

2.7.1 System References

```
//  Declaration of References
using System;
using System.Drawing;
```

2.7.2 Global Variables

```
namespace AntsNest
{
/// <summary>
/// Summary description for OverlayAgent.
/// </summary>
public class OverlayAgent
{
        //  Bitmap variables used as aliases for the images to be
        //  processed.
        private Bitmap bmpEdges, bmpOutline, bmpBoth;
        //  Bitmap variable used to store the edges and the outline
        //  on a white background.
        private Bitmap bmpBothWhite;
```

2.7.3 Constructor

```
//  Startup subroutine (Constructor) for the OverlayAgent.
//  Takes 3 arguments: - A reference to the image to be processed.
//                     - A reference to the edges to be superimposed.
//                     - A reference to the outline to be superimposed.
public OverlayAgent(Bitmap Original, Bitmap Edges, Bitmap Outline)
{
        //  Set the image to greyscale.
        //  Assigns the passed variable to its local counterpart.
        picInUse = (Bitmap) Original.Clone();
        //  Locks the region of the bitmap on which processing is to be performed.
        //  In this case, the whole bitmap is processed.
        BitmapData bmData = picInUse.LockBits(new Rectangle(0, 0,
            picInUse.Width, picInUse.Height), ImageLockMode.ReadWrite,
            PixelFormat.Format24bppRgb);
        //  Sets how wide a bitmap scanline is in bytes and
        //  the location in memory where to start reading
        //  the bitmap.
```

```
//  Because this is a 24 bits per pixel bitmap, each
//  pixel uses 3 bytes to store its data.  So the stride
//  will basically be (bitmap width) * 3.
int stride = bmData.Stride;
System.IntPtr Scan0 = bmData.Scan0;
 //  Allows the use of pointers.
unsafe
{
        //  Creates a pointer to the spot where to start
        //  reading the bitmap.
        byte * p = (byte *)(void *)Scan0;
        //  Sets it up so that we start working at the start of
        //  the first pixel of each row.
        int Offset = stride - picInUse.Width*3;
        //  Byte variables used to store the Red, Green or Blue
        //  value of the pixel currently being processed.
        byte red, green, blue;
         //  Runs through all of the pixels in an image.
        for(int y=0;y<picInUse.Height;++y)
        {
                for(int x=0; x < picInUse.Width; ++x )
                {
                        //  Assigns the individual color values of each pixel
                        //  to byte variables.
                        blue = p[0];
                        green = p[1];
                        red = p[2];
                        //  Sets each pixel's colour values to the average of its
                        //  3 colour values.
                        p[0] = p[1] = p[2] = (byte) ((blue + green + red) *
                                0.333);
                        //  Sets the pointer the next pixel.
                        p += 3;
                }
                //  Moves the pointer to the start of the next row.
                p += Offset;
        }
}
//  Releases the bitmap.
picInUse.UnlockBits(bmData);
//  Sets all three the images to the original greyscale image.
bmpEdges = (Bitmap) picInUse.Clone();
bmpOutline = (Bitmap) picInUse.Clone();
bmpBoth = (Bitmap) picInUse.Clone();
```

```
            //  Creates an exact copy of the original image, but set all its
            //  pixels to white.
            bmpBothWhite = (Bitmap) Original.Clone();
            for(int x = 0; x < bmpBothWhite.Width; x++)
                    for (int y = 0; y < bmpBothWhite.Height; y++)
                            bmpBothWhite.SetPixel(x,y,Color.White);
            //  All superimposing done in the following 2 loops will
            //  be done in red on the greyscale images, but in black
            //  on the white bitmap.
            //  Runs through all of the pixels in the Edges bitmap.
            //  Any pixel set to black specifies an edge pixel.
            //  Any edge pixels are then superimposed alone on a greyscale
            //  image of the original, superimposed on a greyscale image
            //  of the original along with the outline, superimposed on
            //  a white background along with the outline.
            for (int x = 0; x < Edges.Width - 1; x++)
                    for(int y = 0; y < Edges.Height - 1; y++)
                            if (Edges.GetPixel(x,y).ToArgb() == Color.Black.ToArgb())
                            {
                                    bmpEdges.SetPixel(x,y,Color.Red);
                                    bmpBoth.SetPixel(x,y,Color.Red);
                                    bmpBothWhite.SetPixel(x,y,Color.Black);
                            }
            //  Runs through all of the pixels in the outline bitmap.
            //  Any pixel set to black specifies an outline pixel.
            //  Any outline pixels are then superimposed alone on a greyscale
            //  image of the original, superimposed on a greyscale image
            //  of the original along with the edges, superimposed on
            //  a white background along with the edges.
            for (int x = 0; x < Outline.Width - 1; x++)
                    for(int y = 0; y < Outline.Height - 1; y++)
                            if (Outline.GetPixel(x,y).ToArgb() == Color.Black.ToArgb())
                            {
                                    bmpOutline.SetPixel(x,y,Color.Red);
                                    bmpBoth.SetPixel(x,y,Color.Red);
                                    bmpBothWhite.SetPixel(x,y,Color.Black);
                            }
    }
```

2.7.4 Return bmpEdges

```
//  This function returns the newly created Edge bitmap.
public Bitmap getbmpEdges()
{
        return bmpEdges;
}
```

2.7.5 Return bmpOutline

```
//  This function returns the newly created Outline bitmap.
public Bitmap getbmpOutline()
{
        return bmpOutline;
}
```

2.7.6 Return bmpBoth

```
//  This function returns the newly created Outline and Edge
//  greyscale background bitmap.
public Bitmap getbmpBoth()
{
        return bmpBoth;
}
```

2.7.7 Return bmpBothWhite

```
//  This function returns the newly created Outline and Edge
//  white background bitmap.
public Bitmap getbmpBothWhite()
{
        return bmpBothWhite;
}

}
}
```

2.8 Agent Communication Blackboard (Whiteboard)

This class allows the different agents to communicate with each other in order to determine who needs to process the image next.

2.8.1 System References

```
//  Declaration of References
using System;
using System.Data;
using System.Drawing;
using System.Collections;
using System.Drawing.Imaging;
using System.ComponentModel;
using System.IO;
```

2.8.2 Global Variables

```
namespace AntsNest
{
/// <summary>
/// Summary description for ToBlackAndWhiteAgent.
/// </summary>
public class Whiteboard
{
        //  Integer variable used to keep track of the agent currently processing.
        public int counter;
        //  BitmapData variable used to lock away part of the image to be
        //  processed.
        public BitmapData bmData = null;
        public BitmapData bmData2 = null;
        //  Bitmap variables used as aliases for the current image.
        public Bitmap picInUse = null;
        public Bitmap picInUseClone = null;
        public Bitmap picInUseTB = null;
        public Bitmap picInUseDone = null;
        //  Integer variable used to keep the agents in a waiting state.
        public int RunStartWait = 0;
```

2.8.3 Constructor

```
//  Startup subroutine (Constructor) for the GreyScaleAgent.
//  Takes 1 argument: A reference to the image to be processed.
public Whiteboard(Bitmap a)
{
        counter = 0;
```

```
        picInUse = a;
        picInUseClone = (Bitmap) a.Clone();
        bmData = picInUse.LockBits(new Rectangle(0, 0, picInUse.Width,
                    picInUse.Height), ImageLockMode.ReadWrite,
                    PixelFormat.Format24bppRgb);
        bmData2 = picInUseClone.LockBits(new Rectangle(0, 0, picInUse.Width,
                     picInUse.Height), ImageLockMode.ReadWrite,
                     PixelFormat.Format24bppRgb);
}
```

2.8.4 Increment Subroutine

```
//  This subroutine increments the global counter to let individual agents know
//  when to process.
public void Increment()
{
        counter++;
}
```

2.8.5 Restart Subroutine

```
//  This subroutine clears the blackboard so that the process can restart for a new
//  image.
public void Restart()
{
        bmData = picInUse.LockBits(new Rectangle(0, 0, picInUse.Width,
                    picInUse.Height), ImageLockMode.ReadWrite,
                    PixelFormat.Format24bppRgb);
        bmData2 = picInUseClone.LockBits(new Rectangle(0, 0, picInUse.Width,
                     picInUse.Height), ImageLockMode.ReadWrite,
                     PixelFormat.Format24bppRgb);
        counter = 1;
}
```

2.8.6 ReleaseBitmapData Subroutine

```
//  This subroutine releases the area of the image currently being processed.
public void ReleaseBitmapData()
{
        picInUse.UnlockBits(bmData);
        picInUseClone.UnlockBits(bmData2);
        bmData = null;
        bmData2 = null;
}
}
}
```