ARTIFICIAL INTELLIGENCE MACHINE VISION GRADING SYSTEM

By

Nicolaas Johannes Luwes

Dissertation submitted in fulfilment of the requirements for the degree

Doctorate Technologiae: Engineering: Electrical

In the

School of Electrical and Computer Systems Engineering

Of the

Faculty of Engineering, Information and Communication Technology

Of the

Central University of Technology, Free State

Supervisor: Dr P.E. Hertzog D.Tech. (Eng)

VERKLARING TEN OPSIGTE VAN SELFSTANDIGE WERK

Ek, NICOLAAS JOHANNES LUWES, verklaar hiermee dat die navorsingsprojek wat vir die verwerwing van die graad DOKTORAAL TECHNOLOGIAE: INGENIEURSWESE: ELEKTRIES aan die Sentrale Universiteit van Tegnologie, Vrystaat deur my voorgelê, my selfstandige werk is en nie voorheen deur my of enige ander persoon ter verwerwing van enige kwalifikasie voorgelê is nie.

....

N.J. Luwes Datum

DECLARATION OF INDEPENDENT WORK

I, NICOLAAS JOHANNES LUWES, hereby declare that this research project, submitted for the degree DOCTORATE TECHNOLOGIAE: ENGINEERING: ELECTRICAL, is my own independent work that has not been submitted before to any institution by me or anyone else as part of any qualification.

....

N.J. Luwes Datum

ACKNOWLEDGEMENTS

I would like to thank the following persons and institutions for their help with, and contribution to, the completion of this project:

The Central University of Technology, Free State, for the opportunity to do this project.

Dr P.E. Hertzog for his friendship and assistance as study leader.

My parents for their love and encouragement.

Miss C. Foster for her support and motivation.

God Almighty, the Creator of all.

OPSOMMING

Masjienvisiestelsels word deesdae meer gereeld in die gradering van kos en ander biologiese en vervaardigde produkte gebruik. Hulle het egter beperkings. Vervaardigde produkte is minder problematies om te gradeer. Gradering is redelik eenvoudig deurdat 'n foto van die perfekte monster geneem word en dan met díe van die vervaardigde voorwerp vergelyk word. Indien die voorwerpe korreleer, kan die produk deurgelaat word. Indien nie, kan dit verwyder word. Redes vir verwydering kan byvoorbeeld vervorming of verkleuring wees. Na die aanvanklike tydrowende programmeringstydperk, werk die masjienvisiestelsel effektief en moeiteloos totdat die produk ten opsigte van, byvoorbeeld, sy vorm, tekstuur, kleur, ens. sou verander. In vinnig veranderende markte gebeur opgradering gereeld. Vars produkte is moeiliker om deur masjienvisiestelsels te gradeer deurdat monsters van dieselfde klas nie noodwendig dieselfde sal lyk nie. Hiervoor benodig 'n mens 'n stelsel wat, soos die menslike ekwivalent, 'n voorspelling van gradering kan maak. Die ontwerpte stelsel in hierdie studie kan voorspellings maak (soos die menslike ekwivalent), en het dus die vermoë om unieke landbou- en biologiese produkte of enige tipe vervaardigde produk te gradeer. Die stelsel is in staat om vervaardigde produkte, ongeag hul oriëntasie op die vervoerband, te gradeer. Dit kan eenvoudig en vinnig herleer word as die produkte en seisoen sou verander. Die werking word verkry deur 'n kamera aan intelligente sagteware, wat uit twee dele bestaan, te koppel. Die eerste deel het ten doel om die kenmerke uit al die data af te lei wat vir die tweede deel, d.w.s. die besluitnemende neurale netwerk, nodig is. Hierdie tweedeel-sagtewarestelsel verkry die beste werking uit 'n moderne rekenaar se argitektuur vir effektiewe gradering.

SUMMARY

Machine vision systems are used most commonly for grading food and other biological produce and manufactured products. One problem with machine vision systems is that they have limitations. The grading of manufactured products is less problematic. Grading is fairly straightforward: a photograph of a perfect sample is taken and compared with an image of the manufactured objects. If the objects correlate, they can be allowed to pass on a conveyer belt, and if not, they can be removed. The reasons for non-correlation might be malformation or discolouration, etc. After an initial time-consuming programming period, the machine vision system will work effectively and effortlessly until the product changes or its form, texture, colour, etc. are upgraded. In fast-changing markets, upgrading is frequently done. Fresh produce is far more difficult to grade for machine vision systems as samples of the same class may not look the same. Here a system is needed that can, like a human, do predictive grading. Such an artificial intelligence machine vision system can be taught as humans can be taught. The system developed in this study is able to make predictions (like a human) and has the ability to grade unique agricultural and biological products or any type of manufactured product. This system is able to grade products regardless of their orientation on the conveyer belt. It can be retrained easily and quickly if the product or season changes. This capability is accomplished by connecting a camera to intelligent software consisting of two parts. The first part is the feature extraction part that extracts all the data necessary for the decision-making neural network part. This two-part software fully utilises the capability of modern computer architecture for effective grading.

LIST OF ACRONYMS

ADC	Analog-to-Digital Converter
AI	Artificial Intelligence
CCD	Charge-coupled device
CCIR	Comité Consultatif International des Radiocommunications
CIE	Commission Internationale de l'éclairage (International Commission on Illumination)
CMOS	Complementary metal–oxide semiconductor
d.c.	Direct current
dpt	Dioptre
DSP	Digital Signal Processing
EIA	Electronics Industries Association
FPGA	Field-Programmable Gate Array
HIS	Hue, Intensity and Saturation
HSV	Hue, Saturation and Value
IEEE	Institute of Electrical and Electronics Engineers
LED	Light Emitting Diode
LTI	Linear Time Invariant
m	Metre
mm	Millimetre
NI DSP	National Instruments Digital Signal Processor
NI	National Instruments
RGB	Red, Green and Blue
ROI	Region of Interest

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Companies invest heavily and consistently in solutions to optimise their manufacturing operations.

Many producers are lowering their waste thresholds in production and improving product quality using today's more focused, more affordable machine vision solutions. These can deliver a reduction in the waste threshold, boost return on investment and help fund more capital-intensive production upgrades. A by-product of improved quality is improved retailer and customer confidence.

Food and fresh produce have slim profit margins compared to products of other manufacturing industries, and increasing quality and variety demands from customers, retailers and regulatory agencies are forcing the fresh-produce industry to take an incremental improvement approach. This is evident from the fact that machine vision food sorters are widely used for the fresh-pack market in Europe and North America.

In North America, companies like Autoline, Aweta, Barco, Best Compaq Delta Computer Systems, Delta Technology and Dipix Technology, to name just a few, have become key players in the fresh pack market [40, p. 1].

These days, manufactured products are produced at great speeds on fast-moving conveyers. To ensure quality product distribution, products are inspected using machine vision solutions. Machine vision systems are used to remove sub-quality products from the assembly and distribution line.

1.1 Problem statement

Industries use visual inspection to assure that products or produce meet the required quality standards. The two basic industries that grade according to appearance are the agricultural and manufacturing industries. Cultivated food and other biological produce includes fruit, vegetables, grains, etc. Manufactured products, on the other hand, are not biological and may include objects such as cellphones, automotive parts, laptop computers, etc.

Each of these different market segments may have different inspection and grading requirements, but all share some general requirements [39]:

- Sort by size
- Sort by shape
- Sort by colour
- Sort by ripeness
- Sort by grade or a by a combination of size, shape, ripeness, colour.

1.1.1 Food and other biological produce

As stated, food and other biological produce are valued for their appearance. Appearance is mainly evaluated according to the size, shape and colour of an object.

The importance of size can be observed from the fact that it is so clearly stipulated in grading standards. Note the stipulation of size in, for instance, the standards for strawberries as published in the United States Standards for Grades of Strawberries:

- “Size. Unless otherwise specified, the minimum diameter of each strawberry is not less than three-fourths inch.” [8, p. 1].Or the stipulation of size for peaches as published in the United States Standards for Grades of Peaches:

- “Size requirements.

  (a) The numerical count or a count-size based on equivalent tray pack size designations or the minimum diameter of the peaches packed in a closed container shall be indicated on the container.

  (b) When the numerical count is not shown the minimum diameter shall be plainly stamped, stencilled, or otherwise marked on the container in terms of whole inches, whole and half inches, whole and quarter inches, or whole and eighth inches, as 2 inches minimum, 2 ¼ inches minimum, 1 7/8 inches minimum, in accordance with the facts. The minimum and maximum diameters may both be stated, as 1 7/8 to 2 inches, or 2 to 2 ¼ inches, in accordance with the facts.

  (c) “Diameter” means the greatest dimension measured at right angles to a line from stem to blossom end of the fruit.

  (d) In order to allow for variations incident to proper sizing, not more than 10 percent, by count, of peaches in any lot may be below the specified minimum size and not more than 15 percent may be above any specified maximum size” [7, p. 4].

Shape is also an important visual quality parameter of food and fresh produce, especially of fruit and vegetables. Human sorters are currently employed to sort fruit based on shape. Shape is a feature that is easily comprehended by a

human, but is difficult to quantify or define by computer, since agricultural and biological products are unique in nature and the growing environment cause various boundary irregularities. Machine vision shape detection is more successful when used on industrial objects, which have a more definite structure [30, p. 227]. Take for example the cellphone industry: In a cellphone face manufacturing plant, after the cellphone face has been removed from the mould, it is placed on a conveyer belt to be conveyed to the shipping bay. It is inspected on this belt by a camera, which correlates it with a previously acquired picture of the perfect cover as saved in the system's memory. Correlation is done on face colour and correct size, among other things. If the correlation does not meet a certain percentage, then that face is rejected.Colour is another major characteristic which is traditionally inspected by humans. The colour of fresh produce and products indicates parameters such as ripeness, defects, etc. Quality decisions vary among the graders and are often inconsistent. The adaptation of the human eye to small changes in colour and the effect of the background on the perceived colour and colour intensity are the main sources of error [31, p. 284].

There is therefore a need for an artificial intelligence machine vision system that can be taught like a human, but which has better throughput and consistency.

This can be accomplished by using the same algorithms used by a human or biological brain. Such a system should also be able to make predictions to give it the ability to grade unique agricultural and biological products without fatigue or wrongly perceived information.

1.1.2 Industrial products

Industrial machine vision systems are custom designed for a specific application. These systems are only effective if extensive mathematical analysis has been done on a specific product to identify the key points to be inspected. This analysis is time consuming and expensive. The reason for this is that a skilled engineer is needed for the analysis and programming.

Machine vision systems for industrial objects will operate effectively and effortlessly once the initial programming has been done, but as soon as the product changes, the specialist engineer must return and re-analyse and re-program the system.

In the cellphone industry, for instance, phone models are changed or upgraded frequently. This requires frequent reprogramming of the image processor system.

To overcome these drawbacks, the artificial intelligence machine vision system is taught by example. This entails that it must be able to analyse and program itself, resulting in a system that can grade a new product in a few minutes.

1.2 Hypothesis

A specialised, high-quality, artificial intelligence machine vision system that has the ability to learn how to recognise produce, such as potatoes or oranges, and products, such as plastic blocks, for the purpose of quality grading, can be designed. It should be able to do quality grading by recognising peripherals such as shape, size and colour by simply being shown examples of each grading class.

2 LITERATURE REVIEW

2.1 Digital image processing

The goal of a machine vision system is to create a model of the real world from acquired images. A machine vision system recovers useful information about a scene from its two-dimensional projections [12] [14, p. 1]. Applications of the technology range from vision-guided robot assembly to inspection tasks involving measuring, verification and detecting defects [14, p. 1].

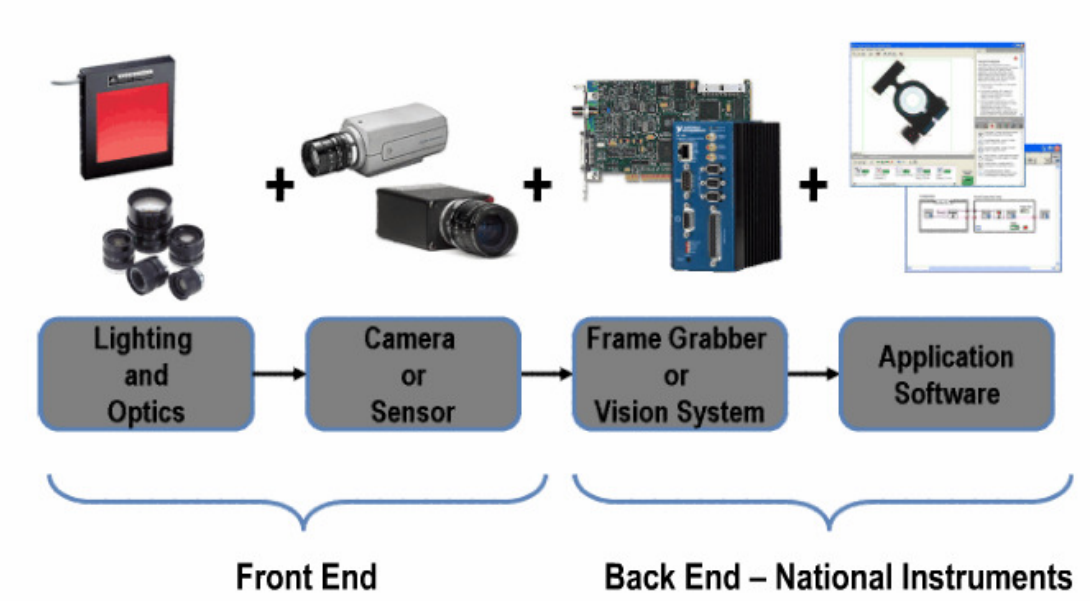


Figure 2-1: Machine vision system [20]

Figure 2-1 shows the four key components that are required for a machine vision system, namely the lighting and optics to enhance the object under test, a camera or sensor to capture the image, an image acquisition device, and the software to complete the vision system.

Images are one or more dimensional signals and can be anything from light intensity, reflectance, colour, temperature, radioactivity, X-rays, etc.

If such an image (defined as *a)* depends on spatial coordinates (*x, y),* these can be defined as *a(x,y)* for an image or *a(x,y,t)* for film, and *a(x,y,λ)* for a multi-spectral image, where *λ* represents the wavelength of light. Colour images can then be defined as *a(x,y,k),* where *k* represents the colour channel [15, p. 298].

A digital image is acquired by sampling an image at a finite number of points (depending on the sensor) and each sample has to be represented within the finite word size of the computer. This word size is known as the quantisation. Each sample is called a pixel or picture element.

These samples are sampled on a grid of squares or rectangles. The number of sampling points is called the sampling rate or image resolution.

Image resolution on standard video sensors is 800x600 pixels, but digital cameras are not bound by this, and can give resolutions of millions of pixels.

The quantisation rate determines how many intensity levels will be used to represent the intensity of each pixel. The intensity of a pixel is represented as an 8-bit integer (unsigned character in the range of 0 to 255 for monochrome images where 0 is black and 255 is white and all the values in between are the corresponding shades of grey.

Colour images on the other hand are represented in three channels, namely Red, Green and Blue (RGB in the CIE system), or Hue, Saturation and Intensity (in the HSI system). These three channels can be regarded as three separate monochrome images of 8-bit integers.

2.1.1 Image acquisition

2.1.1.1 Light and illumination

Lighting or illumination is the addition of a light source that will provide the necessary contrast. This can also be added to reduce shadows and glare from the image.

Light is the part of the electromagnetic spectrum that can be perceived by the visual system.

An electromagnetic wave is described by its frequency f in Hertz (Hz) or its wavelength V(λ) in metres (m). These two are co-dependent on the speed of light [15, p. 83] [36, p. 5).

$$\lambda = \frac{c}{f} \qquad 2\text{-}1$$

where:

c = Speed of light

f = Frequency

λ = Wavelength.

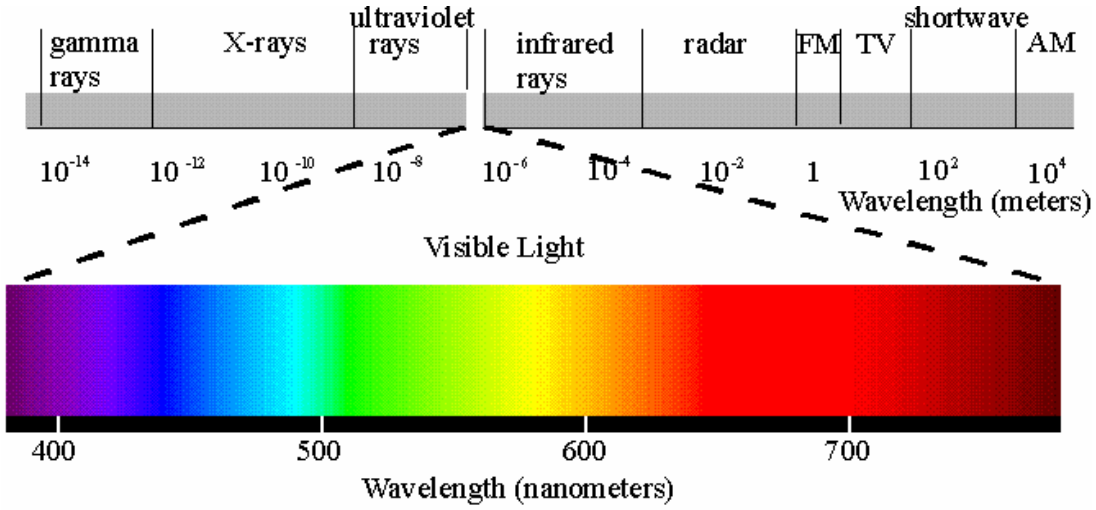


Figure 2-2: Visible light spectrum [31, p. 6] [36, p. 6] [15, p. 82] [18]

The visual part of the electromagnetic spectrum is indicated in Figure 2-2 as between 350 and 1 000 nanometres.

Another model for light is that light consists of a flow of small particles called photons. A photon is the smallest possible quantum of light and carries an amount of energy depending on its wavelength.

$$E = \frac{h.c}{\lambda} \qquad 2\text{-}2$$

where:

h = Planck's constant

c = Speed of light

λ = Wavelength

E = Energy.

2.1.1.1.1 Illumination sources

There are many ways to illuminate an object. Some illumination sources for machine vision applications are conventional electric light bulbs, fluorescent tubes, halogen or tungsten lamps with fibre optics or light-emitting diodes (LEDs). LEDs are the most frequently used light source due to the following advantages [15, p. 121] [10, p. 1229]:

- LEDs are powered by a d.c. power supply which means that they do not flicker at mains frequency
- More accurate measurements can be achieved as a result of the smaller chromatic aberration in the lens due to the monochromatic illumination
- The price of LEDs and their running costs are less than those of other light sources
- The life of LEDs is approximately 100 000 hours
- LED illumination does not produce heat, noise and vibration, and does not require high voltages.

2.1.1.1.2 Illumination methods

There are many illumination set-ups, each with their own advantages. The most common set-up is front lighting where the camera and light source are arranged on the same side of the object. Some sub-assemblies of this set-up can be for instance:

- Bright-field illumination, where there is a direct path from the light source to the camera [15, p. 158].
- Dark-field illumination, where there is no direct light path from the light

source to the camera. This application is used to inspect for scratches on surfaces [15, p. 174] [10, p. 1233] [41, p. 108].

- Diffused lighting which consists of multiple-angled light sources, used to eliminate reflections [41, p. 91].
- Shadow projection, which is used to measure an object's border.

2.1.1.2 Colour

White light is a mixture of all the visible spectral components. For example, a red object would only reflect the red component of the white part.

The law of colour metrics states that any colour can be additively composed of only three primary colours, namely red, green and blue. This can be realised by colour addition as follows:

- Three colour light sources that illuminate at the same time, for example side projectors with colour filters.
- Illumination by the basic colours at irresolvable frequencies, for example the three-colour spinning wheel.
- A spatial matrix that cannot be resolved spatially, for example television screens and monitors.

Another method of producing composite colour is to put colour filters on top of each other. This is known as subtractive colour composition, and is used in conventional colour photography and printing. The basic colours of subtractive colour composition are yellow, purple and cyan.

2.1.1.2.1 CIE colour systems

Additive colour composition can be calculated as a vector addition. This means that the three primary colours can be used to build a three-dimensional colour vector space. Thus the value of each primary colour can be used as a colour measure.

2.1.1.2.2 HIS colour space

The CIE diagram does not correspond to human vision and is only a tool for colour definition and measurement. In the HIS colour space colour is defined according to three variables, namely hue, saturation and intensity. Hue is the angle describing the wavelength. Saturation represents the amount of the colour present, for instance the difference between red and pink. Intensity is the amount of light, for instance between dark red and light red [32, p. 45].

In other words, the HIS colour space uses cylindrical coordinates, where saturation is proportional to the radial distance, hue is a function of the angle in the polar coordinate system, and intensity is the distance along the axis perpendicular to the polar coordinate system.

This means that a point in the RGB colour space can be transformed to the HIS colour space by [29, p. 122]:

$$I = \frac{R + G + B}{3} \qquad \text{2-3}$$

where:

I = Intensity

R = Red

G = Green

B = Blue.

$$S = 1 - 3 \times \frac{\min(R,G,B)}{R+G+B} \qquad 2\text{-}4$$

where:

S = Saturation

R = Red

G = Green

B = Blue.

$$H = \arccos\left[\frac{\frac{1}{2} \times ((R-G)+(R-B)}{\sqrt{(R-G)(R-G)+(R-B)(G-B)}}\right] \qquad 2\text{-}5$$

If $B > G$ then $H = 2\pi - H$

where:

H = Hue

R = Red

G = Green

B = Blue.

Note that this transformation has a few singularities, namely saturation, where R=G=B=0 resulting in black, and hue, where R=G=B, a condition which applies to all the grey, black and white values.

2.1.1.3 Optics

Optics is the science that describes the behaviour and properties of light and the interaction of light with matter.

In optics Fermat's law states that light rays do not take the shortest path between two points, but take the shortest time. This may lead to directional changes in the light rays if they move from a medium in which the speed of light is lower to a medium with a higher optical density. This is called refraction. Refraction is the basis of the operation of fibre optics.

The optical density is described by the refraction index n, where $n1$ and $n2$ are the refraction index of the two mediums, respectively. The law of refraction (Snell's law) is as follows [25, p. 168] [33, p. 11]:

$$\frac{\sin \alpha 1}{\sin \alpha 2} = \frac{n2}{n1} = \frac{c1}{c2} \qquad 2\text{-}6$$

where:

α = Angle

n = Refraction index

c = Speed of light.

Table 1 shows some optical indices [37, p. 21]:

Table 1: Optical indices

Vacuum and air	n =1
Water	n =1.333
Glass	n =1.51
Diamond	n =2.42

If the angle $\alpha 2$ exceeds a refraction angle, then there will be no refraction and total internal reflection will take place.

2.1.1.3.1 Depiction systems

The next step in an image processing system is to project the real world onto a two-dimensional sensor system through a depiction system.

The simplest depiction system is a pinhole camera.

Figure 2-3: Pinhole or black box camera [21]

This camera operates without a lens. It operates on the theory that light travels in a straight line. Light reflecting off a surface will converge into a pinhole camera's aperture and emerge within the camera as a cone of light onto the surface of the film.

A drawback of this camera is its low light intensity which results in a long exposure time.

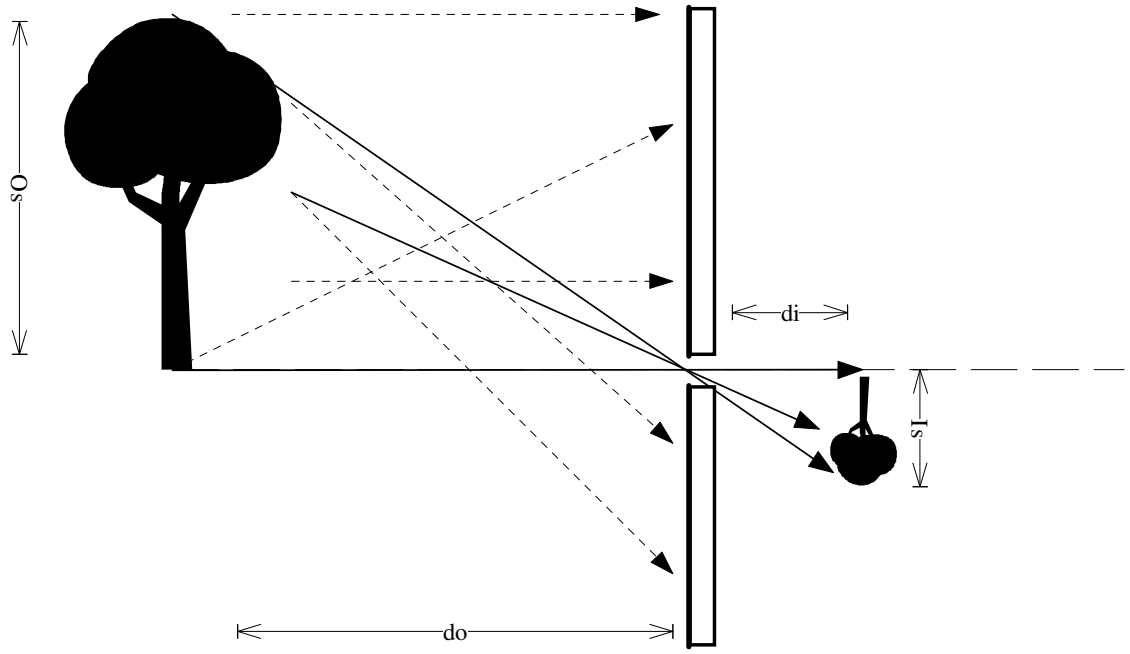


Figure 2-4: Illustration of a pinhole depiction system

Figure 2-4 shows that only part of the light rays enter through the hole, hence the longer exposure time required. The depiction law of this set-up is as follows [36, p. 18]:

$$\frac{Is}{Os} = \frac{di}{do} \qquad 2\text{-}7$$

where:

Is = Image size

Os = Object size

di = Distance from hole to image

do = Distance from hole to object.

The following type of depiction system is a lens system. A lens system operates on the theory that the rays from an infinite light source enter the system parallel to each other.

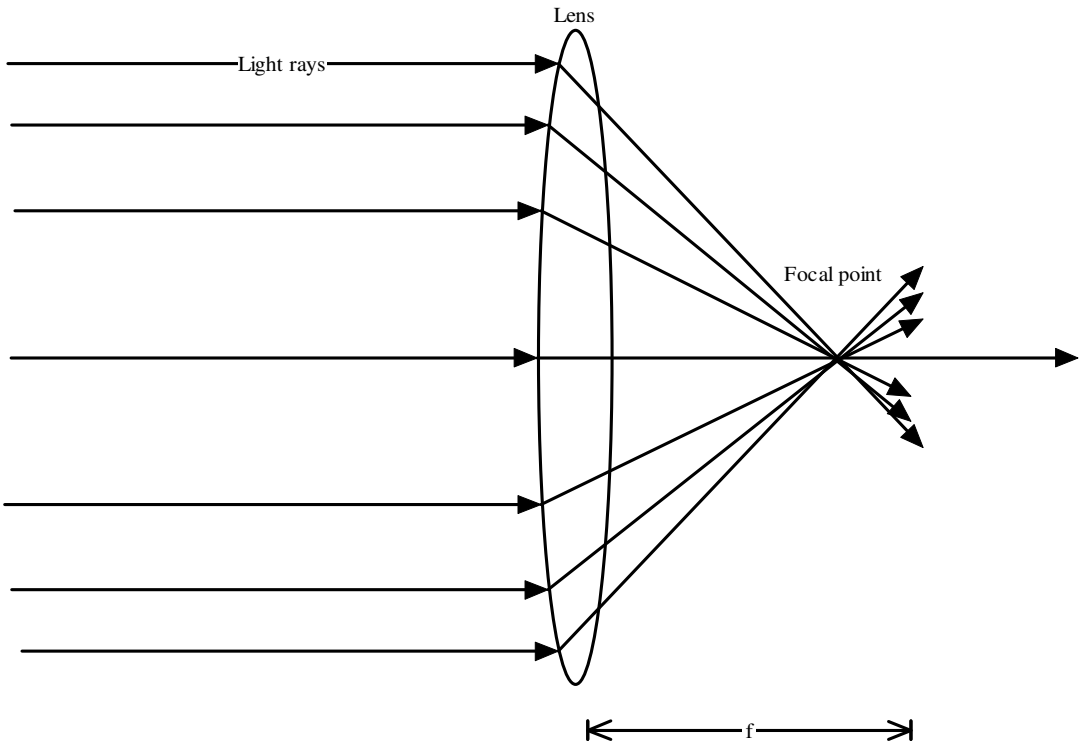


Figure 2-5: Lens depiction system

The lens is positioned perpendicular to the parallel rays. Due to its form, the lens bends the rays to a focal point. The focal point is defined as a mapping of an infinitely remote light source. The distance between the focal point and the lens is defined as the focal length.

The disadvantage of a lens system is that it must be focused for a sharp image. In a pinhole system only the size of the image changes if the distance of an object changes with respect to the hole. In a lens system the rays are bent, so if an object is moved closer to the lens, the rays will come together behind the focal point. To acquire a sharp image the sensor must be moved in accordance with the object's distance from the lens.

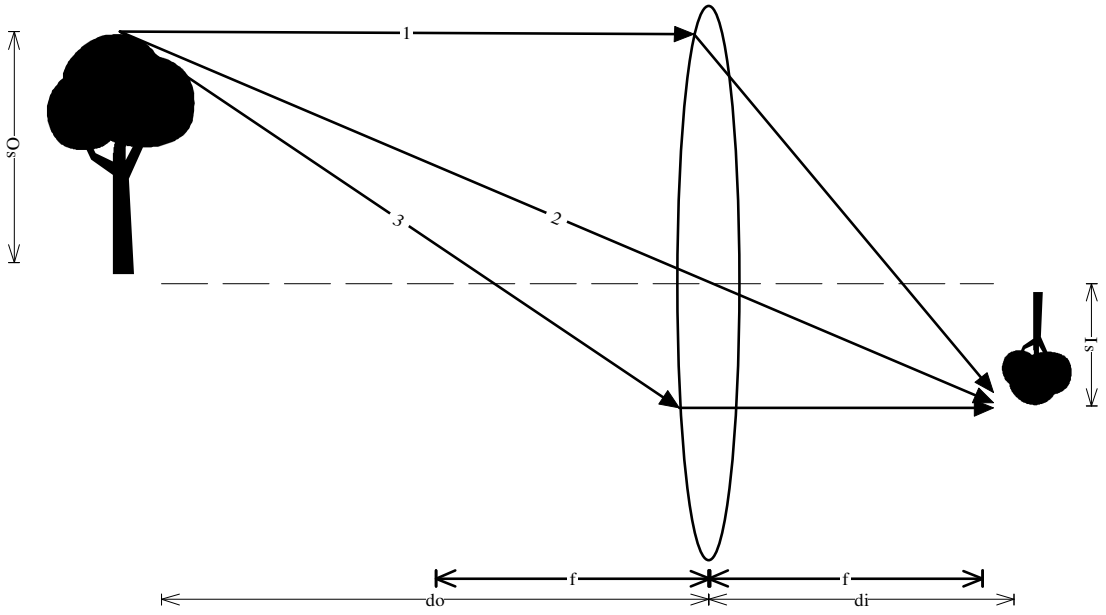


Figure 2-6: Focal plane construction

Figure 2-6 shows how a focal plane can be constructed using three special light ray paths. Light ray 1 which is parallel to the optical axis on the object side passes the focal point on the image side. Light ray 2 which passes the centre of the lens is not refracted. Light ray 3 which passes the focal point on the object side is parallel to the optical axis.

Thus by changing the distance between the lens and the sensor, one can shift the focus on images at different distances from the system.

The focal length is calculated as follows in metres (m) [36, p. 21]:

$$f = \frac{di}{1 + \frac{Os}{Is}} \approx di\frac{Is}{Os} \text{ for } Os >> Is \qquad 2\text{-}8$$

where:

Is = Image size

Os = Object size

di = Distance to image

do = Distance to object.

The magnification factor is as follows:

$$m = \frac{Is}{Os} \qquad 2\text{-}9$$

where:

Is = Image size

Os = Object size.

If the focal length is shortened, then the lens has to reflect the light more. The power of the lens D is therefore reciprocal to the focal length. This is measured in dioptres (dpt). Thus for a focal length of 100 mm one would need a lens refraction power of 10 dpt.

$$d = \frac{1}{f} \qquad 2\text{-}10$$

where:

f = Focal length.

A lens is also classified according to its angle of view, which is an important feature of a lens. The classification consists of a wide angle, standard or telephoto lens and is as follows [24, p. 48]:

$$\theta = 2 \tan^{-2}\left[\frac{Is_{\max}}{2f}\right] \qquad 2\text{-}11$$

where:

$Is_{\max}$ = Size of sensor

f = Focal length.

Distortion of a lens increases with its angle of view, thus this parameter describes the quality of a lens.

If a sensor is positioned to capture a sharp image from a point A, then from a closer point B the image would appear blurred. A small aperture or iris decreases the blur. This is explained by the fact that a small aperture size gives the lens system the same physical properties of a pinhole camera and therefore creates advantages.

An aperture is described as

$$k = \frac{f}{d} \qquad 2\text{-}12$$

where:

f = Focal length

d = Effective diameter.

Measuring the size of an object with a normal lens depends on its distance to the lens. But as stated above, with an addition of an aperture a decrease in perspective distortion can be achieved.

2.1.1.4 Cameras

Digital image processing cameras consist of two parts:

1) The image acquisition unit based on pick-up tubes, CCD or CMOS chips.

2) The output unit which generates the video signals suitable for image-processing devices.

Standard cameras generate two internationally accepted video signals, namely CCIR (Comité Consultatif International des Radiocommunications) and EIA (Electronics Industries Association).

2.1.1.4.1 CMOS camera

There are other cameras available other then CMOS and CCD, but for this study are the focus on CMOS. CMOS sensors are based on photodiodes for photodetection. An area-scan CMOS sensor camera also provides features such as a full-frame shutter and electronic exposure time control.

Normally, exposure time and charge readout are controlled by values transmitted to the camera's control register via the, in this study IEEE 1394, interface. Control registers are available to set exposure time and frame rates. There are also control registers available to set the camera for single-frame or continuous-frame capture.

At readout, accumulated charges are transported from each of the sensor's light-sensitive elements (pixels) to a pixel memory.

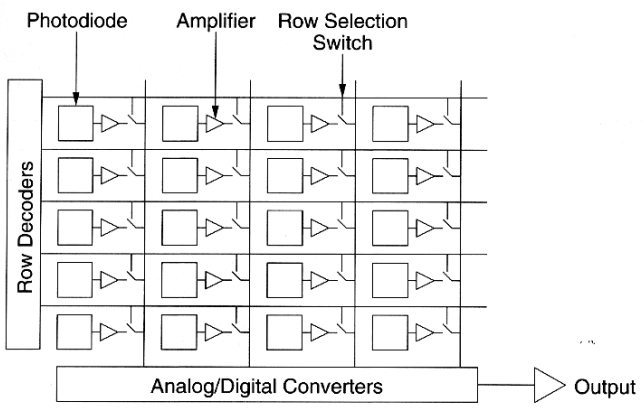


Figure 2-7: An example of CMOS sensor architecture [36, p. 41]

As the charges are moved out of the pixels and into the pixel memories, they are converted to voltages. There is a separate memory for each pixel. Because the sensor has separate memory, the next image can be exposed while the sensor is reading out data from the previously captured image.

The pixel memories can be connected to a bus (there is one bus per vertical column). For readout the pixel memories are addressed row-wise by closing a switch that connects each pixel memory in the addressed row of the column buses. As the voltages leave the column buses they are amplified, an offset is applied and they are digitised by the ADCs (analog-to-digital converters). A variable gain control and a 10-bit ADC are attached to the end of each column bus.

Form the column buses, the digitised signals enter a horizontal output register. The 10-bit digital video data are then clocked out of the output register, through an FPGA (field-programmable gate array) and into an image buffer.

The data leave the image buffer and pass back through the FPGA to an IEEE 1394 link layer controller, for this study, where it is assembled into data packets that comply with the “1394 - digital camera specification” (DCAM). The packets are passed to a 1394 physical layer controller which transmits them synchronously to a 1394 interface board in the host PC.

2.1.2 Image processing

To process the data received from the digital camera, it is necessary to know what type of data are being received. The image from the camera is a two-dimensional matrix [12]. To get a better understanding of this type of data, a comparison can be made between the camera matrix and the eye of a fly.

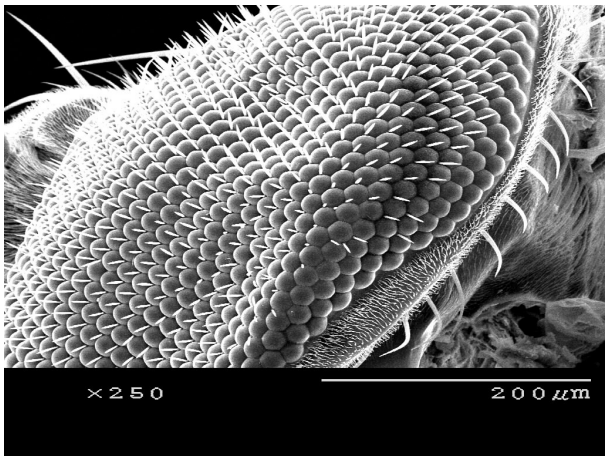


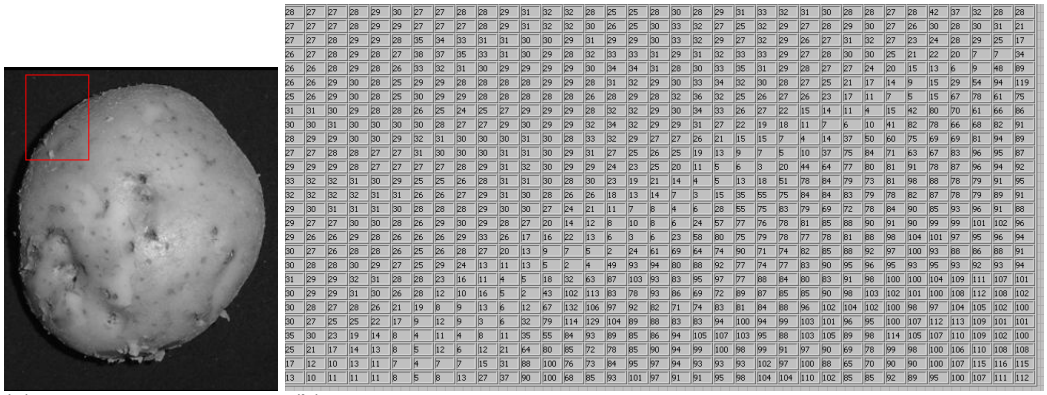
Figure 2-8: Eye of a fly [17]

The close-up of a fly's eye in Figure 2-8 clearly shows the light-sensitive cells. The combination of the light-sensitive cells generates an image of the surroundings to the fly. Similarly, the camera sensor is made up of a matrix of light-sensitive sensors.

Figure 2-9: A series of zoom factors of the same image [16]

The zoomed image in Figure 2-9 shows the different intensity levels measured by the various sensors in the digital camera matrix. As is, the image is not recognisable to the human eye, but if it is zoomed out as shown in Figure 2-9 (a) to (c), the human eye will smooth out the blocks and the image will become

recognisable as the forest depicted in the picture. A computer, on the other hand, does not have the intelligence to see this bigger picture and thus cannot recognise the image. A computer is only concerned with the individual values from the sensors. Observe the image matrix of a potato as seen by the PC.



(a) (b)

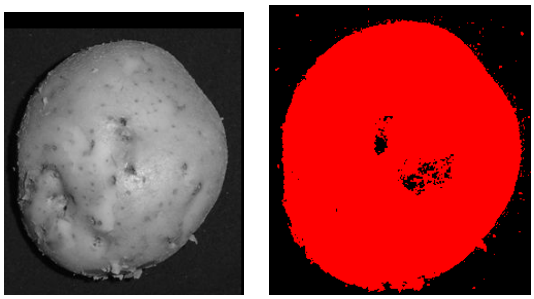
Figure 2-10: An image of a potato (a) and the 2D matrix of the image (b)

Figure 2-9 shows how a computer interprets the image. The computer in this case only “sees” a two-dimensional matrix of values between 0 and 255 (8-bit image): 255 represents white and 0 dark or black (observe the dark edge of the potato in the wire box and then the corresponding low numbers edge in the sensor matrix in Figure 2-9).

There are advantages of not seeing the bigger picture and only individual numerical values. One can apply any mathematical calculation or measurement to these numeric values.

2.1.2.1 Threshold

The threshold of an image is where certain intensity levels are dissipated to binary 0 and others taken to binary 1, resulting in a binary image [14, p. 13] [12].



(a) (b)

Figure 2-11: (b) is the binary image of (a)

Thresholding can be implemented with an **if statement** coded as "if the input value is bigger than a certain threshold make the result a 1, otherwise make it a 0".

2.1.2.2 Erosion

Binary images may contain spectral values on the background due to reflections or low-quality camera sensors. Erosion, like its counterpart in nature, erodes small spectral values away. Erosion is applied to the binary image with a 3 by 3 matrix (this can be any size matrix) and the following equation [12] [14, p. 161] [2, p. 8] [5, p. 965]:

$$a = e1 \bigcap e2 \bigcap e3 \bigcap e4 \bigcap e5 \bigcap e6 \bigcap e7 \bigcap e8 \bigcap e9 = \bigcap_{i}^{9} ei \qquad 2\text{-}13$$

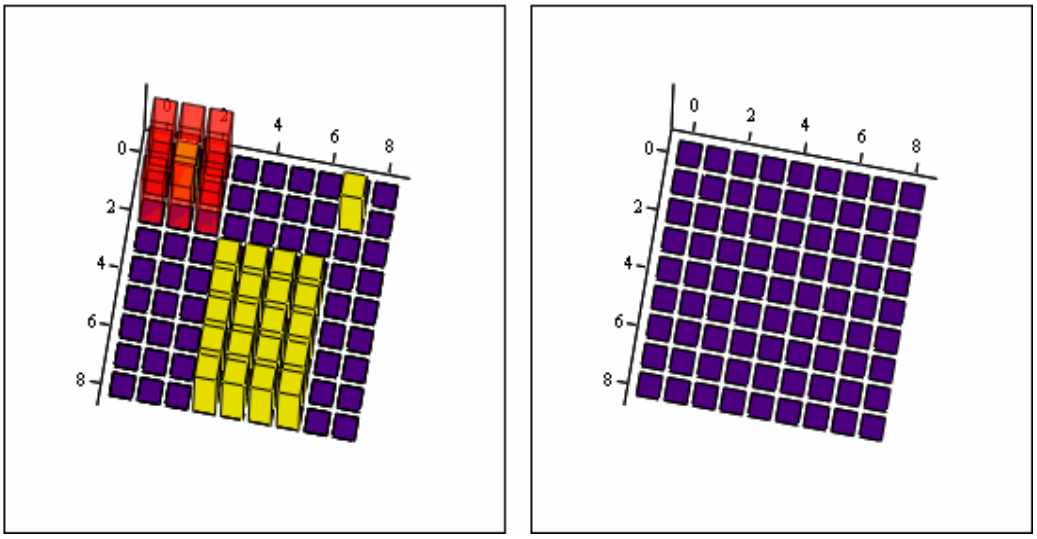
This equation can be rewritten as:

$$e_q = \prod_{0} X_{q,o} \qquad 2\text{-}14$$

where:

$$E = \prod_{q} e_q$$

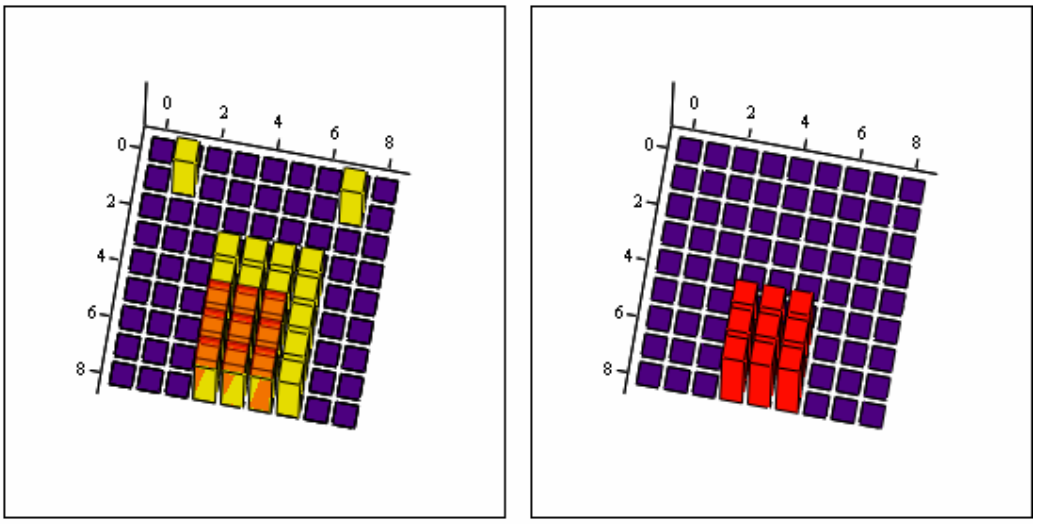
From this it is clear that erosion is a binary AND gate function and the 3 by 3 matrix will only have a 1 as an answer if all the spaces in a 3 by 3 matrix are occupied by a 1.



(a) (b)

Figure 2-12: An eroding 3 by 3 matrix at origin and occupying one spectral 1 and its corresponding result

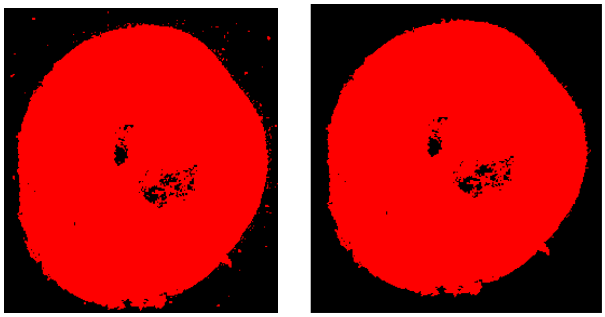
Figure 2-12 shows that the resulting erosion is zero if not all the spaces in a 3 by 3 matrix are occupied by a 1.



(a) (b)

Figure 2-13: An eroding 3 by 3 matrix covering all 1's resulting in the corresponding 1's result

Figure 2-13 shows that the resulting erosion is one if all the spaces in a 3 by 3 matrix are occupied by a 1.



(a) **(b)**

Figure 2-14: Noise removed with erosion

Observe the spectral parts on the areas around the potato in the left image and the lack thereof in the right image after erosion (Figure 2-14).

2.1.2.3 Fill or Dilation

The filling function, also known as dilation, fills holes in the binary image. Filling the binary image is accomplished with a 3 by 3 matrix using the following equation [12] [14, p. 158] [26, p. 158]:

$$a = e1 \bigcup e2 \bigcup e3 \bigcup e4 \bigcup e5 \bigcup e6 \bigcup e7 \bigcup e8 \bigcup e9 = \bigcup_{i}^{9} ei \qquad \text{2-15}$$

This can be rewritten as:

$$e_q = \sum_{o} X_{q,o} \qquad \text{2-16}$$

where:

$$EE = \sum_{q} e_q$$

From this it is clear that filling is a binary OR function and the 3 by 3 matrix will result in 1 if any one of the spaces in the matrix is occupied by a 1.

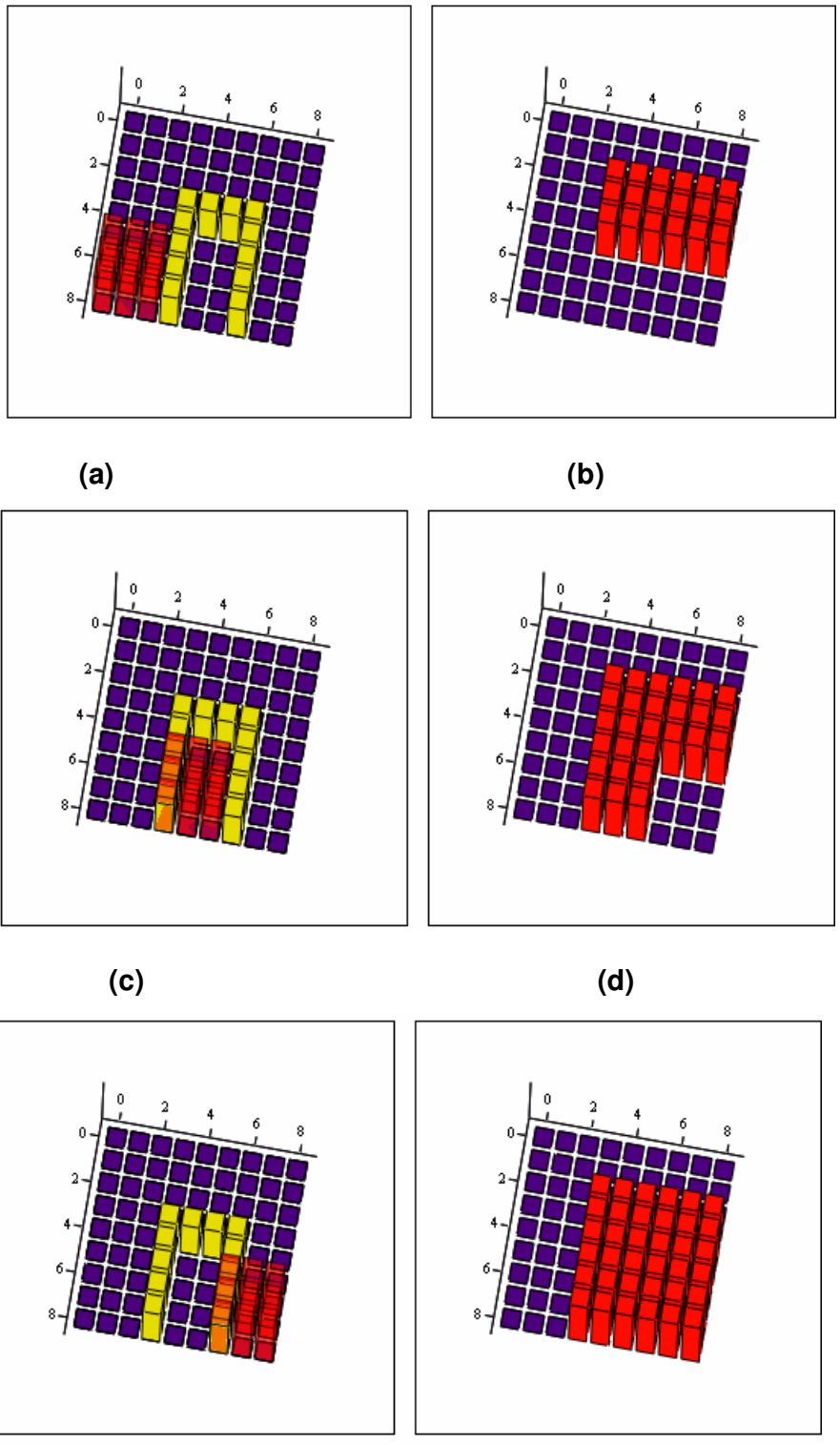
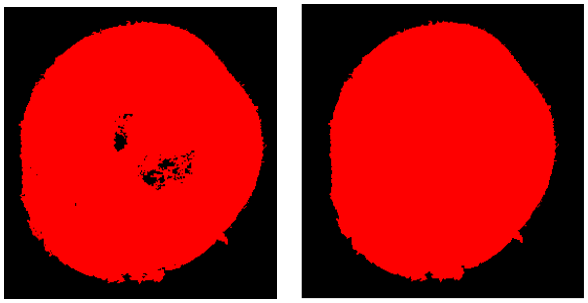


Figure 2-15: Step 3 in filling with a result of 1's since some of the matrix is occupied by a 1

Figure 2-15 (a) and (b) show the first step in filling with a result of 0's, since none of the matrix is occupied by a 1. Figure 2-15 (c) and (d) show step 2 in the filling algorithm with a result of 1's since some of the matrix is occupied by a 1. The same result of 1's will be generated in step 3 as seen in Figure 2-15 (e) and (f), since some of the matrix is occupied by a 1. This demonstrates how the filling function fills the hole in the binary image.



(a) **(b)**

Figure 2-16: Image of the object with holes (a) and the image of the object where the holes have been filled (b)

Observe that all the holes in Figure 2-16 are filled after the function has been performed.

2.1.2.4 Area

The area of the object can be calculated from the processed binary image. The new matrix consists of 1's where the potato occupies the image and 0's for the background. Therefore the area can be calculated by adding all the values of the matrix together as follows:

$$Area = \sum_{x}\sum_{y} pix(x, y) \qquad 2\text{-}17$$

where:

$pix(x, y)$ is the binary image matrix

x is the line position indicator

y is the column position indicator.

2.1.2.5 Moments

Object moments are used in a variety of calculations, for instance:

- Centre of gravity
- Object orientation
- Bonding ovals, etc.

An object's moments are determined by treating each pixel as a physical mass of 1.

The moment of the order i,j is then calculated as [36, p.116] [34, p. 229]:

$$m_{i,j} = \sum_{x} \sum_{y} x^i . y^j . pix(x, y) \qquad 2\text{-}18$$

where:

$pix(x, y)$ represents the binary image matrix

x and y are the coordinates

i and j are the order of moment.

2.1.2.5.1 Centre of gravity

The centre of gravity is used as the reference point and is calculated from the moments of zero and first order:

From this the coordinates of the centre point of gravity are calculated as follows [34, p. 229]:

$$X = \frac{m_{10}}{m_{00}}; Y = \frac{m_{01}}{m_{00}} \qquad 2\text{-}19$$

where:

m are the moments of the object

x and y the coordinates.

2.1.2.5.2 Object orientation

The angle of orientation of an elongated object can be calculated from its central moments. The central moment of an object is the moment of which the origin of the coordinate system is shifted to the centre of gravity.

So the central moments are as follows [34, p. 229]:

$$M_{i,j} = \sum_x \sum_y (x - X)^i . (y - Y)^i . pix(x, y) \qquad 2\text{-}20$$

where:

$pix(x, y)$ is the binary image matrix

X and Y are the coordinates of the centre of gravity

i and j are the order of moment.

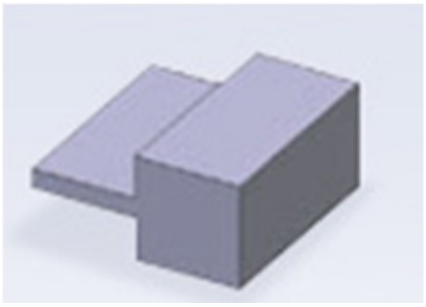
From this, the object orientation can be calculated as follows [36, p. 117]:

$$\theta = \frac{1}{2} \arctan\left(\frac{2M_{11}}{M_{20} - M_{02}} \right) \qquad 2\text{-}21$$

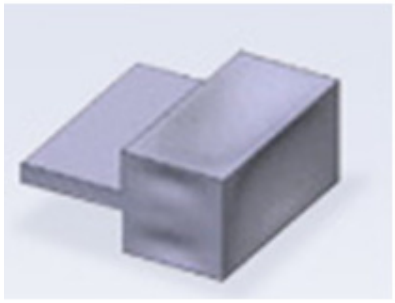
2.1.2.5.3 Bounding oval

The two constants needed to approximate an oval, namely width and height, can be calculated by rotating the solid body around the orientation axis for the one constant and rotating the solid body at 90 degrees to the orientation through the centre point of gravity for the other.

The bounding oval can be used to determine the magnitude of an irregular shape. In products this can be due to shrinkage. When this happens the moulded size is smaller than the designed size.



(a)



(b)

Figure 2-17: Shrinkage before (a) and after (b) cooling [22]

Figure 2-17 shows how a plastic product can shrink with cooling. Everything except water will expand when heated and shrink when cooled. All materials have specific shrinkage rate values. The shrinkage rate is a value predicting how much difference there will be between the plastic product when moulded and when it has cooled to room temperature. This must be inspected to determine whether the acceptable threshold has been exceeded [3, p. 26].

A characteristic that can also be inspected is warping. This occurs when the variation in wall thickness is too big. What then happens is that in the last stage of cooling, shrinkage stress builds up and the part warps, twists or blisters.

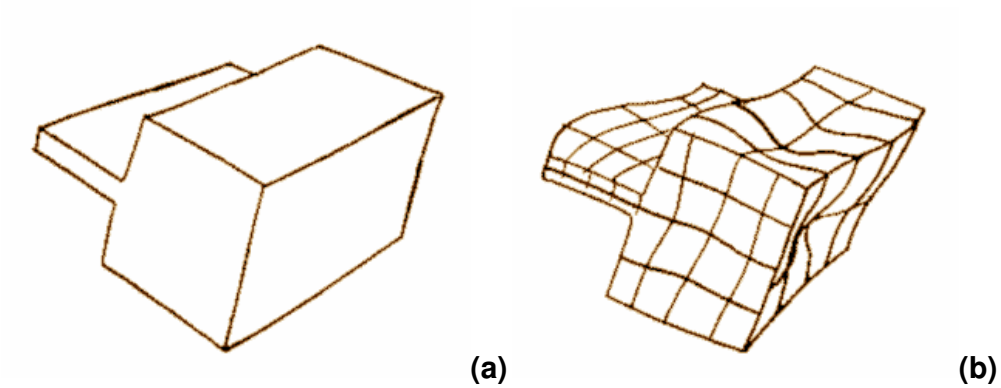


Figure 2-18: Warping before (a) and after (b) cooling [19]

Figure 2-18 shows how a plastic product can warp with cooling. This needs to be inspected to notice that an acceptable threshold is not exceeded [3, p. 15].

In produce irregular shape can be due to irregular grow patterns.

2.2 Neural networks

Artificial Intelligence (AI) of which neural networks is an aspect is the science and engineering of creating intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable [4, p. 4].

The human brain (a neural network) has about twenty billion neurons [35, p. 1]: these neurons are specialised to carry “messages” through an electrochemical process.

Neural networks are based on collections of nodes or neurons that are connected in tree patterns to allow communication between them [4, p. 287]. A single node is a simple processor, which computes by combining the input signals with an activation rule to produce an output signal [4, p. 287].

A single network node would be as follows:

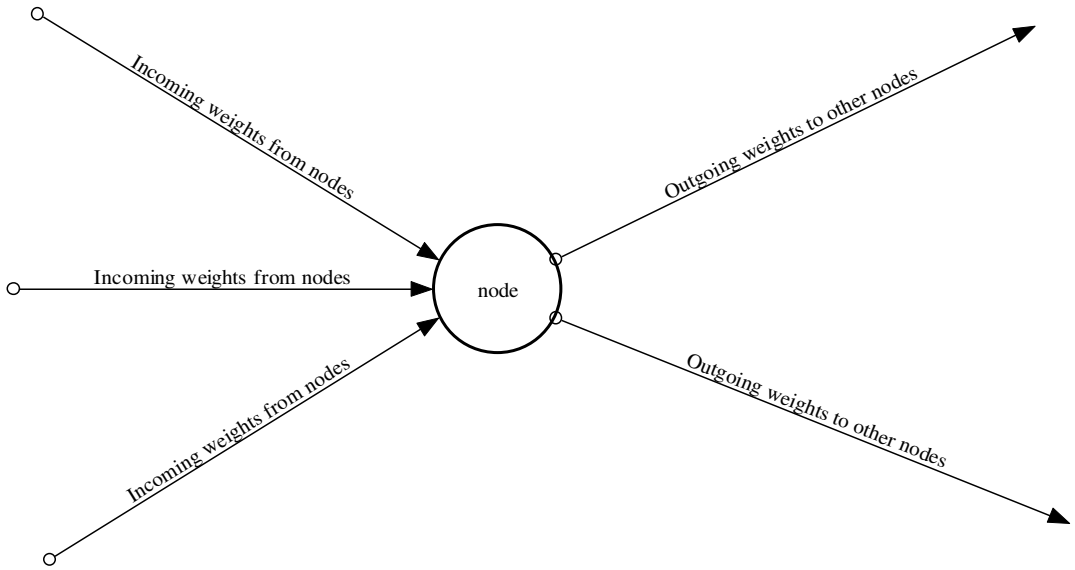


Figure 2-19: A Single network node

The nodes are interconnected with weighted connections. Individually these nodes are limited in operation, but when interconnected, they have the ability to perform complicated tasks. The pattern of connectivity may differ according to where one type of network may be located so that:

- Each neuron or node is connected to every other node.
- It can be arranged into an ordered hierarchy of layers in which connections are allowed only between nodes in immediately adjacent layers.
- Other types of networks allow feedback connections between adjacent layers, or within a layer, or allow nodes to send signals back to themselves.

As stated earlier, these connections carry a variable weight. This weight is a multiplying constant for the connection's input. A multilayered network with supervised training is capable of learning a certain required function. It accomplishes this by calculating the error at each net and slowly adjusting these weights to produce all the required outputs.

This process can be mathematically simulated where the formula of the neuron is as follows [4, p. 290]:

$$net_j := \sum_{i=1}^{N} x_{i,j} w_{i,j} \qquad 2\text{-}22$$

where:

N is the number of inputs

i is the node number for a specific input

j is the number of the net

x is the input value

w is the weight or constant.

If one is working with binary inputs it is best to put the output of a net through a sigmoid function. The sigmoid function is as follows [4, p. 292].

$$f_j := \frac{1}{1 + \left[e^{\left(- net_j \right)} \right]} \qquad 2\text{-}23$$

where:

net is the output of the net

j is the number of the net.

To calculate the error, generalisation of the delta rule is used [4, p. 297]

[6, p. 251].

This is accomplished by starting at the last layer with:

$$\delta_j := \left(t_j - o_j\right) \cdot o_j \left(1 - o_j\right) \qquad 2\text{-}24$$

where:

t is the required output

o is the net output

j is the number of the net.

The error at the hidden layers is calculated next [4, p. 303]:

$$\delta_j := o_j \left(1 - o_j\right) \cdot \sum_k \delta_k w_{j,k} \qquad 2\text{-}25$$

where

o is the net output

j is the number of the net

k is the number of the net from where the error originates

δ_k is the error from the previous layer

l is the number of that specific path.

The weight change for each node is then calculated with [4, p. 303]:

$$\Delta w_{i,j} := \eta \cdot \left(x_{i,j} \cdot \delta_j\right) \qquad 2\text{-}26$$

where:

η is the learning rate

i is the node number for a specific input

j is the number of the net

x is the input value

δ is the error from the each layer.

Thereafter the weights are adjusted as follows [4, p. 304]:

$$W_{i,j} := w_{i,j} + \Delta w_{i,j} \qquad 2\text{-}27$$

where:

Δw is the weight change

w is the old weight.

The workings of a multilayered network with supervised training by backpropagation are best viewed in an example by Callan [4, p. 306]:

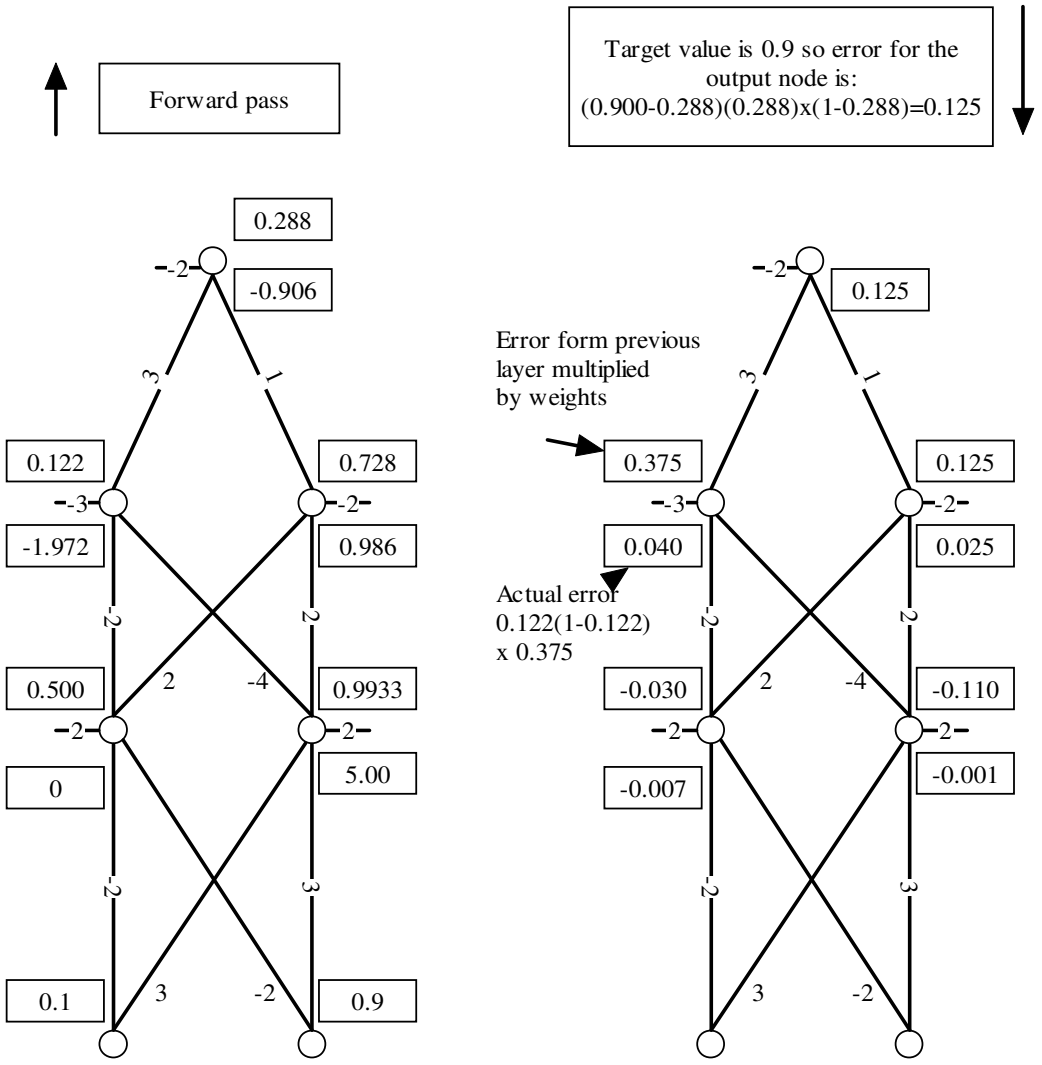


Figure 2-20: A forward and backward sweep of the backpropagation algorithm.

Figure 2-20 shows a two-input, one-output multilayer network. The two inputs are connected through weights to two neurons or nets. These two nets are, in turn, connected through weights to a hidden layer also consisting of two nets. These hidden nets are then connected to a single net producing the output.

Such a multilayered network, with supervised training, consists of two sweeps. The first is forward calculating an output for a specific input. The second sweep is a training cycle which is the backpropagation part. The forward or operation sweep is as follows:

Note that for the forward sweep the inputs are 0.1 and 0.9. To calculate the outputs for the first node on the left, equation 2-22 must be applied, which states that the inputs should be multiplied by their weights and added together as part of the sigma function. Thus 0.1 (input) multiplied by -2 (corresponding weight) added to 0.9 (input) multiplied by -2 (corresponding weight) added to 2 (activation) gives a result of 0. The next action is to step this 0 through the sigmoid function of 2-23, resulting in an answer of 0.5. The same is now applied to the net on the right, where 0.1 (input) is multiplied by 3 (the corresponding weight) added to 0.9 (input) multiplied by 3 (the corresponding weight) added to 2 (activation). This gives a result of 0.5. As before, this result is stepped through the sigmoid function of 2-23, which gives a result of 0.9933.

The values 0.5 and 0.9933 are now the inputs of the next (hidden) layer. As above, the output of the nets are calculated with 2-22, resulting in -1.972 and 0.986. This is also put through the sigmoid function, resulting in 0.122 and 0.728. These are now the inputs for the final layer, which is calculated as before, resulting in an output -0.906 before the sigmoid function and 0.288 after the sigmoid function.

For instance, if the final answer must be 0.9 instead of 0.288, then the error of each net must be calculated with backpropagation. The last net (output net) in the forward pass would now be the first net for calculating the error in backpropagation. This first calculation is done as described with equation 2-24, resulting in (0.900-0.288) (0.288) x (1 – 0.288) = 0.125. From equation 2-25 it is clear that this error from the previous layer is now multiplied by the weights connecting to the hidden layer, resulting in the node on the left being 0.375 and

the one on the right 0.125. Applying the rest of equation 2-25 will result in 0.04 and 0.025.

This is step back to the input nets using equation 2-25, resulting in -0.007 and -0.001. Equation 2-25 is applied from here by calculating the weight change for each node and taking a learning rate into consideration. The weight is then adapted by applying equation 2-26. These newly adapted weights should now be closer to producing the required output for the specific input for the next forward or operation sweep. If the network produces the required outputs for specific inputs, then the backward or training sweep is no longer needed.

2.2.1 Training and test data sets

A training data set is a real-world problem that needs to be mapped. This training data set therefore consists of inputs with the corresponding outputs that will be fed to the neural network for weight adaptation. It is beneficial to randomise the order of the presentation for each training sample [4, p. 307] [13, p. 80].

A test data set that is similar to real-world problems would test the adapted neural network for good generalisation. Good generalisation refers to when the network can produce the correct output for the majority of input samples of the test data set. This would mean that the network can produce smooth non-linear mapping with the ability to interpolate non-exact samples. If the neural network were over-trained, it would be like a memory looking up an output for input, and then interpolation or prediction would not be possible [4, p. 307] [13, p. 80].

3 MATERIALS AND METHODS

The goal of a machine vision system is to create a model of the real world from images. A machine vision system recovers useful information about a scene from its two-dimensional projections [12] [14, p. 1]. Applications of the technology range from vision-guided robot assembly to inspection tasks involving measuring, verification and detecting defects [14, p. 1].

An artificial intelligence machine vision system can be broken down into two basic sections, namely the hardware and the intelligence software sections.

- **Hardware**

A digital camera is connected to a personal computer via Firewire. The software makes certain decisions and these options are connected to the real world via a National Instruments Digital Signal Processor (NI DSP). The NI DSP controls the pneumatics, levers or other devices that select or sort the produce or products on a conveyer belt. There can be four pneumatic actuators labelled A, B, C and D. These are activated with binary from the decision-making software. If the intelligent software is undecided, the object can be left to fall off the end of the conveyer belt where a human can sort the object and add it to the training data set with the aim of improving future performance.

- **Intelligent software**

The software can be broken down into two sections, namely the feature extraction and the image classification sections:

  - Feature extraction

    This part of the software consists of [12] [14, p. 6]:

- Image processing: brightness and contrast, filters and shading correction, etc.
- Image segmentation: thresholding, edge detection, etc.
- Extraction: the calculation of area, centre of gravity, perimeters, etc.

- Image classification

  This is the intelligence part of the software, which classifies the produce with the aid of artificial intelligence, namely, neural networks. The AI software is custom designed, mathematically simulated and then implemented. The AI network uses inputs from the feature extraction section and includes:

  - Colour
  - Size
  - Shape.

As stated earlier, the decisions control the hardware peripherals for sorting via a DSP card.

3.1 Hardware set-up

The ideal would have been to mount the digital camera on a conveyer, but for simulation, design and debugging purposes it was mounted above a turntable. The turntable's function is to simulate the operation of a conveyer belt, but with the advantage of overcoming the problem of constantly reloading objects as they move off the bottom end of the conveyer. After debugging and design changes, the acquisition system could be removed from the turntable and mounted on the conveyer with the same results as those obtained on the turntable.

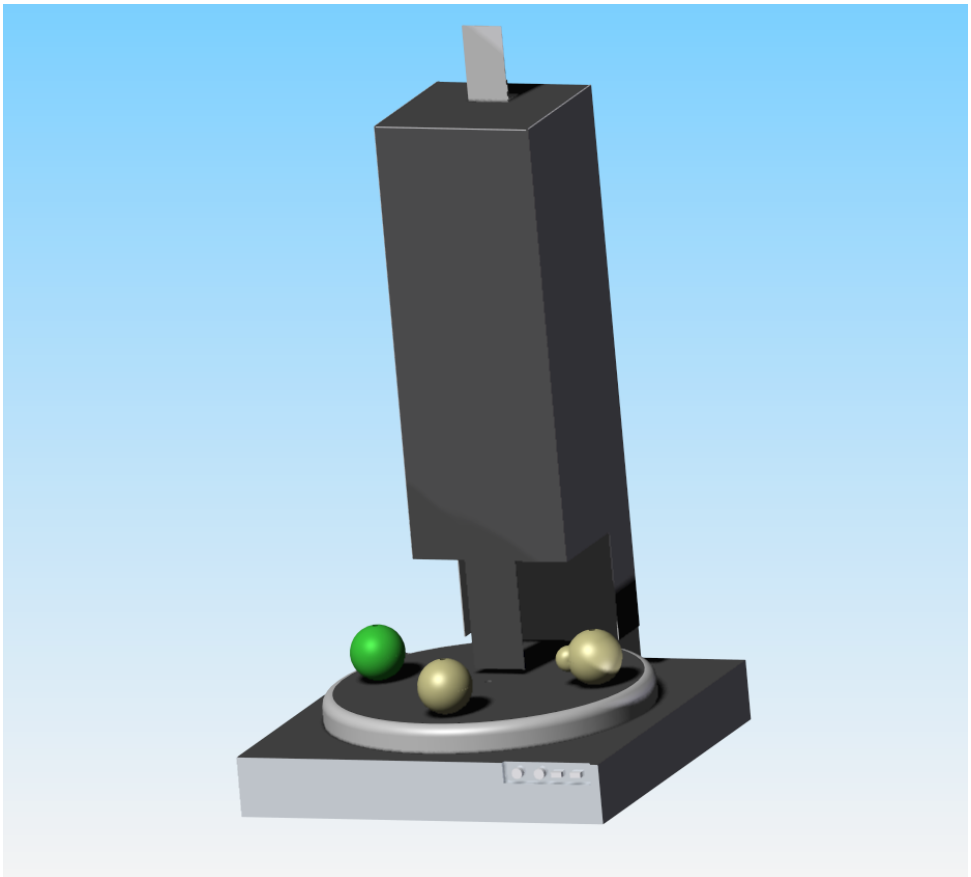


Figure 3-1: Acquisition box mounted on a turntable

Figure 3-1 shows the CAD design of the image acquisition box mounted above a turntable with some objects on it. Inside the image acquisition box are the camera and light source. The reason for the box is to minimise interference from outside lighting, in other words to control the quality and intensity of the illumination.

3.2 Image acquisition

Images were captured with a Basler A601fc digital camera with a Myutron MV-1214 12 mm C-mount lens. The captured frames were transferred via IEEE 1394 Firewire and imported with a Sunix IEEE 1394 Firewire 400 PCI card.

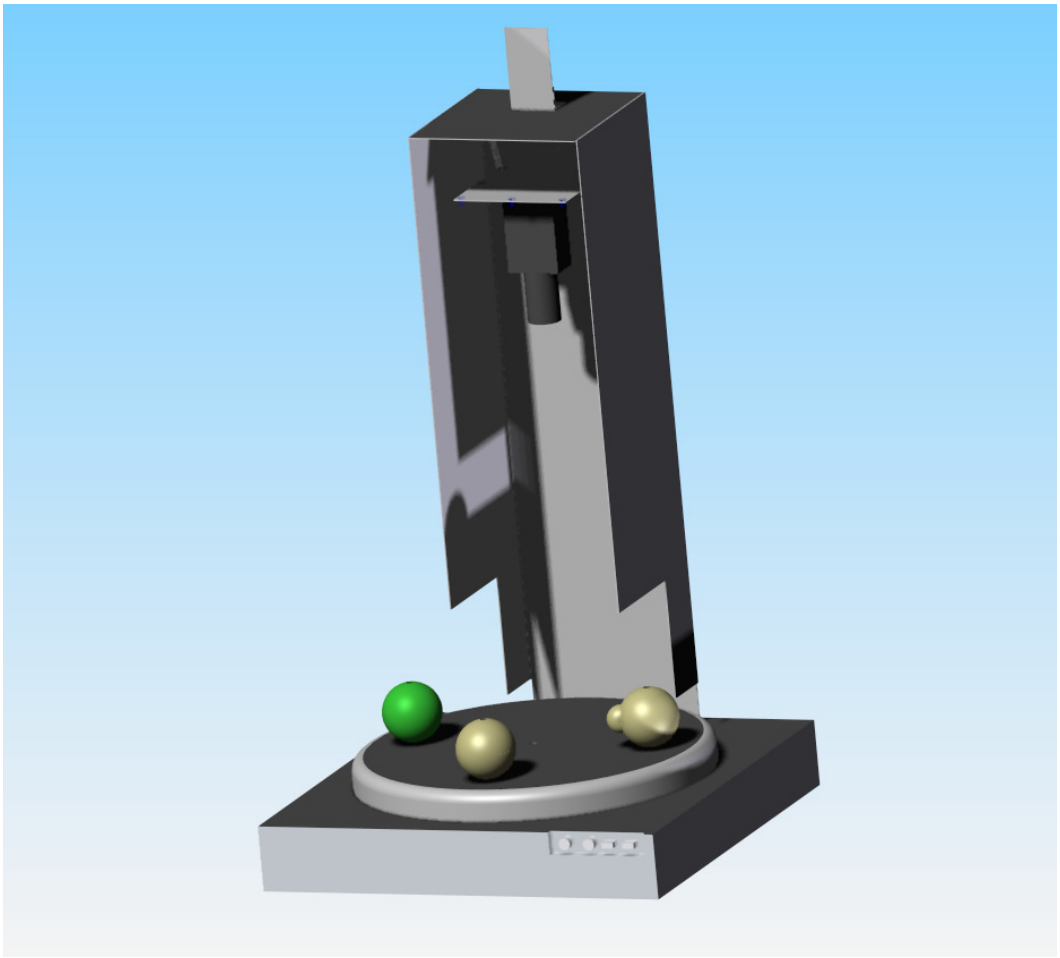
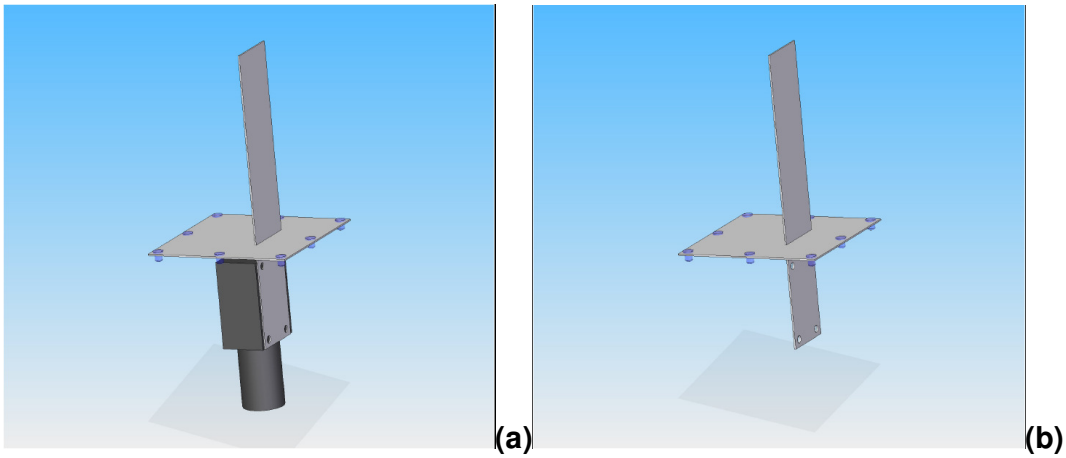


Figure 3-2: The acquisition box opened showing the camera and the LED light source on top of the camera

Figure 3-2 shows the camera mounted on a sliding shaft with the LED light source mounted just above the camera. The reason for the adjustable height is to overcome the problem of an object being too high and out of range of the lens's focal distance.

3.3 Light and illumination

A LED light source was chosen for its advantages as discussed in section 2.1.1.1.1.



(a) (b)

Figure 3-3: The camera mounted on the adjustable shaft with the LED light source above it (a) and the adjustable shaft with the LED light source and the camera removed for a clearer view (b)

From Figure 3-3 it is clear that the light source is connected in a front lighting configuration as discussed in section 2.1.1.1.2.

The inside of the acquisition box is painted silver. This reflects the light off the walls and thus gives diffused lighting as discussed in section 2.1.1.1.2. This improves the overall illumination.

From the CAD designs the implementation is as follows:

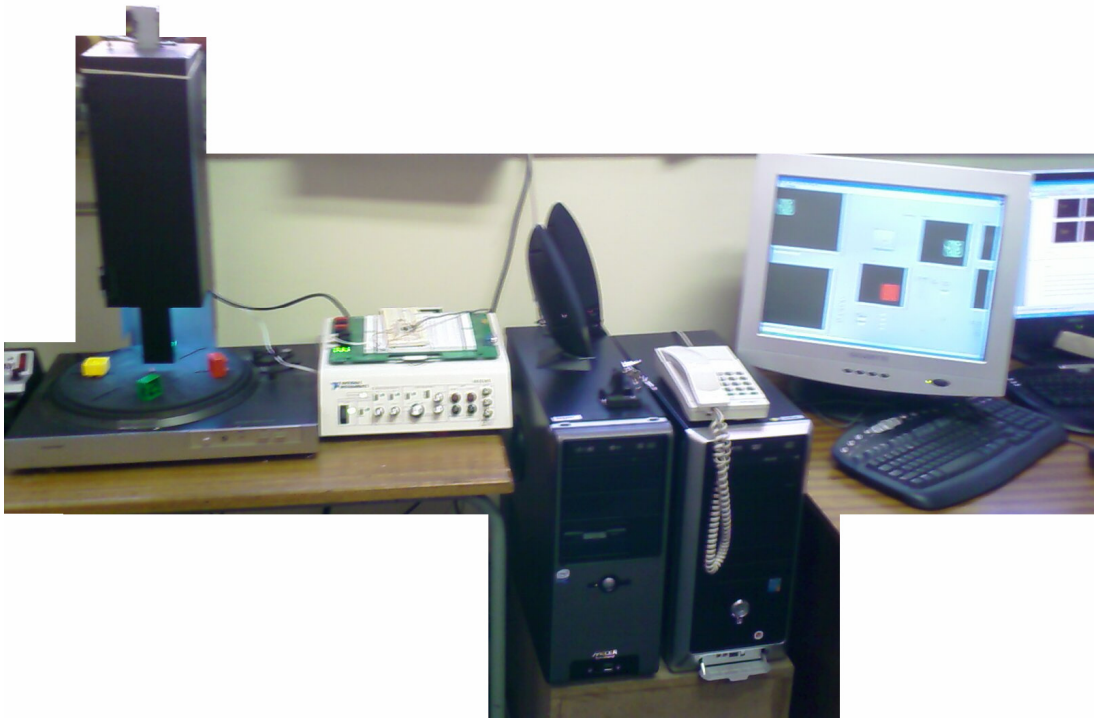


Figure 3-4: The actual acquisition box mounted on a turntable

Figure 3-4 shows how the CAD design was realised. The acquisition box is connected to the PC that runs the software as compiled in LabVIEW.

3.4 Triggering

At the bottom of the acquisition box a detection circuit is mounted that detects when there is an object in the region of interest (ROI). If there is an object the software is triggered to acquire and process the image.

3.5 Neural networks

A multilayered network with supervised training arranged into an ordered hierarchy of layers, in which connections are allowed only between nodes in immediately adjacent layers, is coded for evaluations. Since there are four outputs, it was decided to have four output nodes connected to an input layer with four nodes to which the inputs are connected. This is thus a network with two layers of weights and it has been stated [1, p. 116] that networks with two layers are capable of approximating any continuous functional mapping. Another method would be to input the analog percentages of size, shape, red, green and blue to give a five-input network.

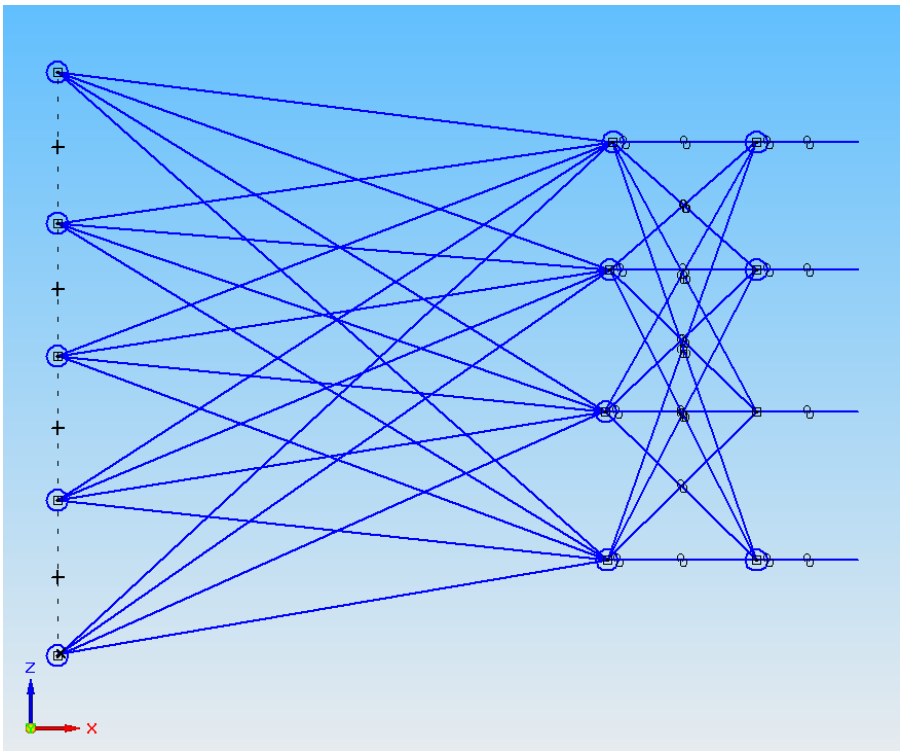


Figure 3-5: A multilayered network for the five analog inputs to the four outputs that drive the actuators with a hidden layer

Figure 3-5 graphically illustrates such an analog input network. The five inputs of percentages of size, shape, red, green and blue are fed in from the left via the weighted connections to the four neurons in the centre. These are then connected via their weighted connections to another four neurons which have one output each to produce the results. By converting the percentages of size, shape, red, green and blue to binary, and depending on the resolution, one can increase the inputs. The reason for this is that a neural network with more weights is like a brain with more weights, making it more intelligent. Neural networks also need a small hamming distance. It was therefore decided to convert the input so that it would consist of incremental levels. To keep the neural network small but with sufficient input resolution, it was decided to step the incremental inputs in 5% intervals, resulting in twenty steps for each input. Thus if the input is 5%, the first input is a 1 and the rest are 0. Then after 10% the two least significant inputs would be 1's and the rest 0's. This could then increase by 5% at a time until 100%, where all twenty inputs would be 1's. This produces a network of one hundred inputs connected to four neurons which, in turn, are connected to four neurons in the output layer connected to the four outputs.

This results in a network shown in Figure 3-6.

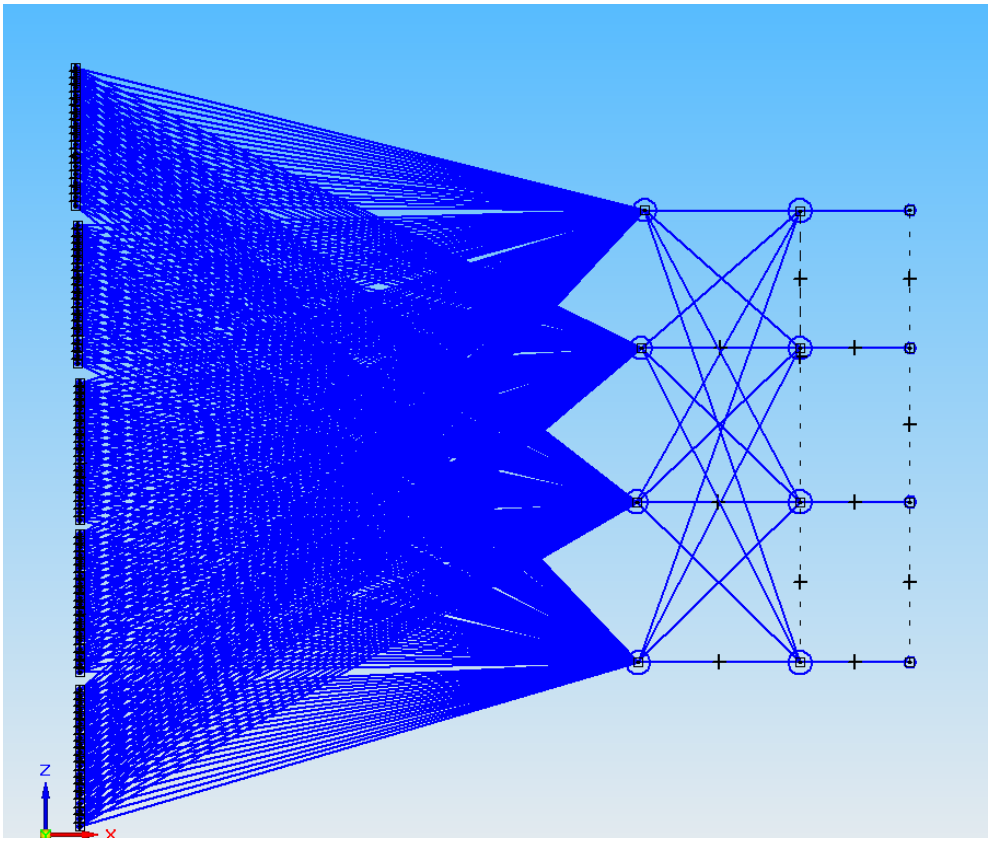


Figure 3-6: A one hundred binary inputs multilayered network, equivalent for the five analog inputs, that will drive the actuators

Figure 3-6 graphically illustrates the digital input network.

The complete network would have one hundred inputs plus one activation input. There are thus one hundred and one inputs connected via weighted connections to each of the four input neurons. The output of each is stepped through a sigmoid function. The next layer has four inputs, each from a neuron (after the sigmoid function) from the previous layer plus an activation input. There are thus five inputs connected via weighted connections to each of the four neurons. From these we get four outputs stepped through sigmoid functions. Due to the nature of the sigmoid function the inputs must be adapted to give a 0.9 for 1 and a 0.1 for a 0. These would produce outputs of values between 0.1 to 0.9, where a value closer to 0.1 would indicate a probability of a 0 and a value closer to 0.9 a

probability of a 1. For the same reasons, the weights are initialised with random numbers between 0 and 1.

The step function with a resolution of 5% per step has an inherent problem when one considers the range of shrinkages as described in Section 1.1.2. Unacceptable shrinking might be less than 5%. It was decided that this input needs to be stepped through a function to improve the resolution in the working range. The size input is a percentage of the area of the object to the total frame. The camera is also set up so that this would be in the 50% range. A sigmoid function is applied since it gives the best resolution in this range. The same must be done for the shape (warping) input, but the shape percentages must be more sensitive in the lower ranges, thus a log function would be more suitable.

The colour inputs have a linear relationship to each other and can thus not be transformed.

3.5.1 Training and test data sets

As stated in section 2.2.1, the network is fed with training data to adapt the weights for the required operation. To prove the ability of the network to sort a variety of objects, the experiment tested three data sets:

The first data set was plastic building blocks to simulate a product.

(a) **(b)** **(c)**

Figure 3-7: The training data set of plastic building blocks (a), (b) and (c) at all the angles at which it may lie

Figure 3-7 shows a product in the form of a plastic building block. Images (a), (b) and (c) show all possible orientations of the object. Since the extraction part calculates only size, shape and colour, the specific orientation in images (a), (b) or (c) does not matter. This technique was repeated for red and green blocks as well. The idea is to sort the blocks according to colour regardless of their orientation. The second training data set consists of samples of potatoes. With this data set one can evaluate the ability of the system not only to sort produce (previously a difficult object for image processing), but also the ability to go from sorting products to produce in a short space of time. Examples of the potato training set are as follows:

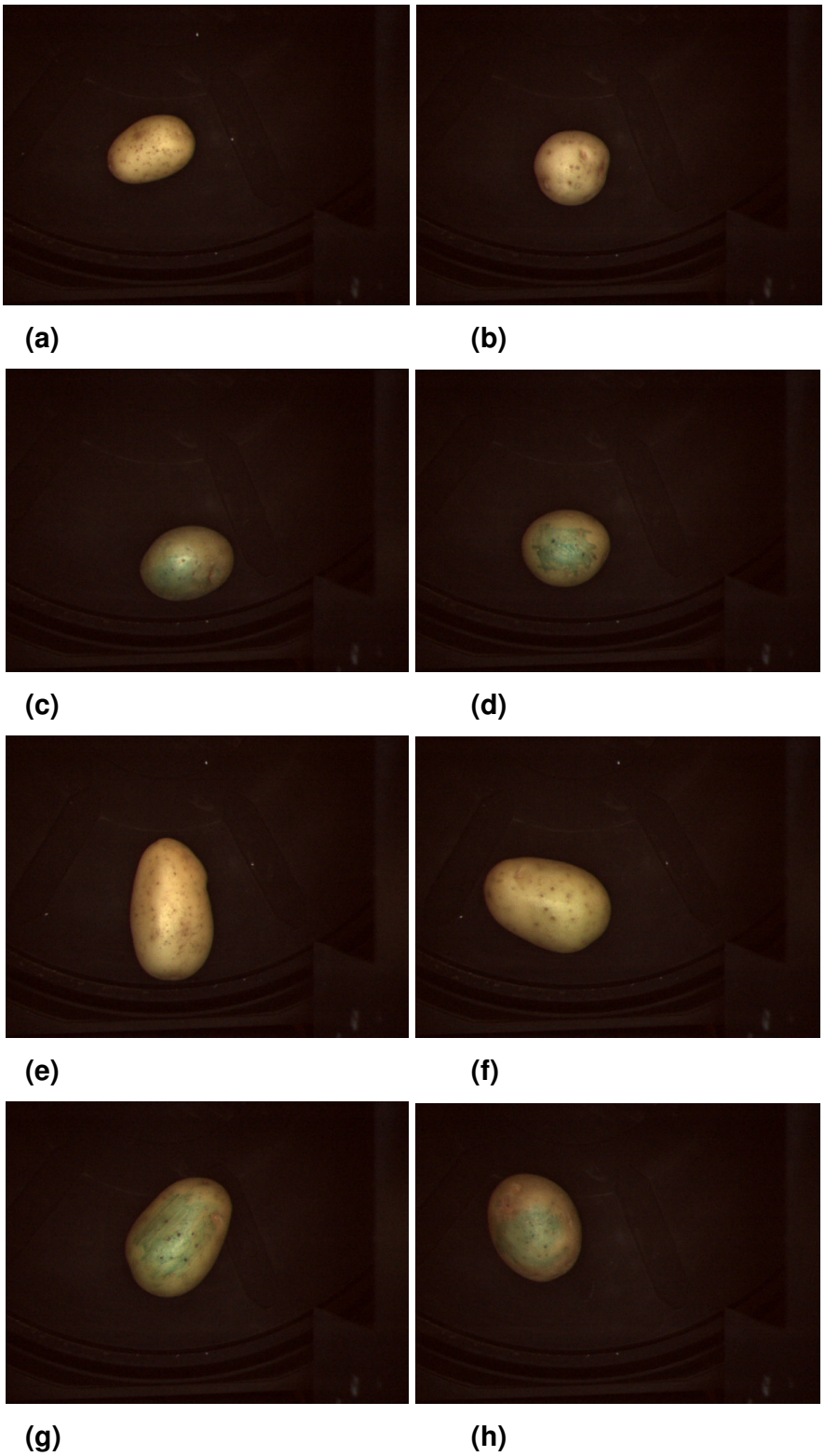


Figure 3-8: In the training data set of potatoes (a) and (b) are samples of small, good quality potatoes, (c) and (d) are samples of small green potatoes, (e) and (f) are samples of large potatoes and (g) and (h) are large green potatoes.

Figure 3-8 shows the training data set of potatoes and how they would be sorted, namely into small, small and green, large or large and green. The irregular-shaped samples are shown in Figure 3-9:

(a) **(b)** **(c)**

Figure 3-9: The training set data of irregular-shaped potatoes - (a) and (b): with irregular large potatoes and (c): with a sample of an irregular small potato

These samples are mixed in with the training data and are left to fall off the end of the conveyer belt.

The last objects for training and evaluation were fresh produce again, as these are more difficult objects to grade by machine vision. This experiment took the form of re-training the network for grading oranges:

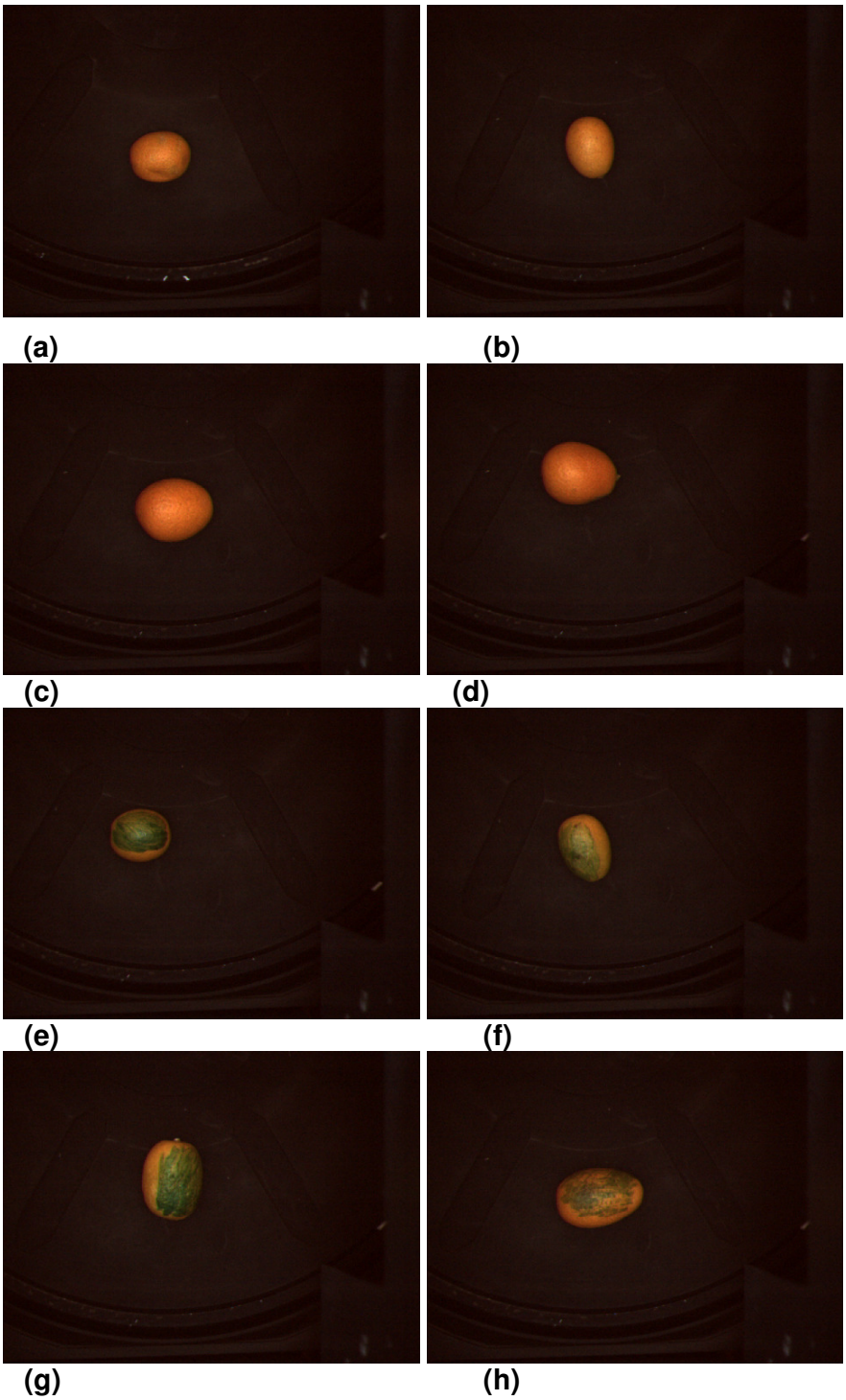


Figure 3-10: The training set data of oranges - (a) and (b), samples of small good-quality oranges, (c) and (d), samples of small green oranges, (e) and (f), samples of large oranges, and (g) and (h), large green oranges

This figure shows the training data set of oranges and how they would be sorted as small, small and green, large or large and green.

The generalisation for all the above is evaluated with test data.

4 MATHCAD ANALYSIS AND DESIGN

4.1 Feature extraction

The aim of feature extraction is to extract useful feature information, which includes size, colour and irregular shapes of an object on a conveyer belt for quality control purposes.

4.1.1 Binary image (Thresholding)

As explained in section 2.1.2.1, a binary image consists of 1's and 0's which represents a greyscale image that was put through a thresholding algorithm [14, p. 13] [12].

The first step was to import a greyscale image for simulation and design purposes. The sample picture is in RGB format and must first be converted into a greyscale picture. This is usually done by reading in the green information of the RGB picture [12]. However, it was felt that this method might not give a true representation of the object. A better way to generate a greyscale image is to convert the RGB to greyscale with the NTSC formula of 0.299 of red, 0.587 of green and 0.11 of blue, which closely represents the average person's perception of the brightness of red, green and blue [11, p. 907].

This function is implemented in MathCAD as follows:

$$sampleA := READ\_RED("c:\DATA\2.bmp") \quad 4\text{-}1$$

This is to read in the red data of the object.

$$sampleB := READ\_GREEN("c:\DATA\2.bmp") \quad 4\text{-}2$$

This is to read in the green data of the object.

$$sampleC := READ\_BLUE("c:\DATA\2.bmp") \quad 4\text{-}3$$

This is to read in the blue data of the object.

This is then put through the NTSC algorithm to ensure the correct amplitude representation of each colour:

$$sample := 0.299sampleA + 0.587sampleB + 0.114sampleC \quad 4\text{-}4$$

where:

Sample A is the red data of the object

Sample B is the green data of the object

Sample C is the blue data of the object.

Figure 4-1: RGB image of the object, in this case a potato

Figure 4-1 shows the object before the NTSC algorithm is applied.

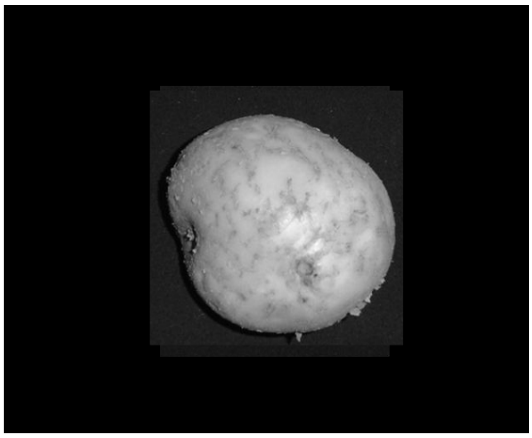


Figure 4-2: The greyscale image of the object in Figure 4-1

Figure 4-2 shows the result of applying the NTSC algorithm.

The next step is to threshold the image. This is usually done by a programming loop ("if" statement) as explained in section 2.1.2.1, but for MathCAD the built-in function is used as follows:

$$\text{bin\_sample} := \text{binarize}(\text{sample}, 40) \qquad 4\text{-}5$$

This function returns a binarised version of a matrix sample, with pixels above a threshold of 40, set to 1 and below 40 to 0.

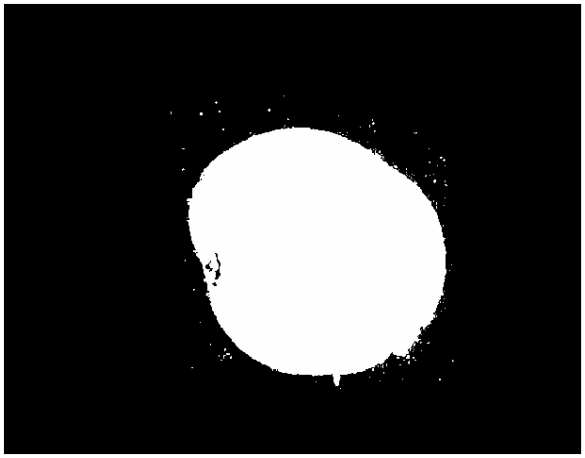


Figure 4-3: The binary image

Figure 4-3 shows the result of the greyscale image stepped through the threshold function. Note that there are some spectral components left. To remove this it was experimentally determined that a threshold of 45 was the optimum to extract the objects for this specific camera and illumination set-up. This now entails that all the grey values smaller than 45 are dissipated to 0 and the values from 45 to 255 are superimposed to binary 1. Figure 4-3 shows that the bin_sample is multiplied by 255. This is done just to make it visible to the human eye on an image frame, as 0 is black and 255 is white, and so 1 would have been just one shade lighter than black. For calculation purposes the top value would remain 1.

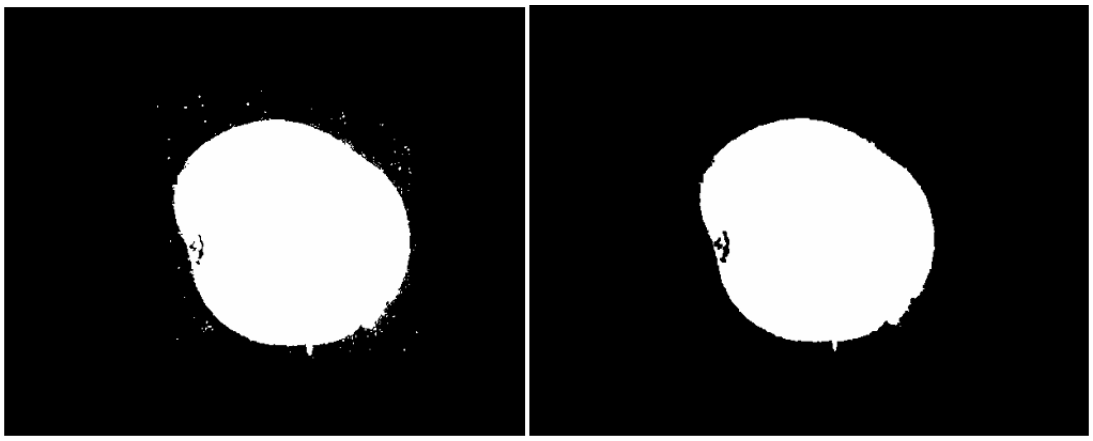
4.1.2 Erode

As in nature and as discussed in section 2.1.2.2 on erosion, small spectral values erode away. This is the next step in the simulation or CAD design. The binary image in section 4.1.1 has some noise in the background due to reflections or low-quality camera sensors that need to be eroded for better quality extraction.

Erosion is accomplished as explained in section 2.1.2.2 with equations 2-13 and 2-14. However, for simulation purposes, the built-in MathCAD function was used:

$$G := \text{erode8}(\text{bin\_sample}, b) \qquad 4\text{-}6$$

This function performs erosion on the matrix bin_sample at a threshold of b using 8 neighbours.



(a) **(b)**

Figure 4-4: The erosion result (b) of the binary image of (a)

Observe that the spectral noise in the background of Figure 4-4 (a) is completely eroded without deforming the object to be extracted. The result is a better quality extraction of the desired object.

4.1.3 Fill

As discussed in section 2.1.2.3, the fill function will fill holes in the binary image. This final binary image is used as the ROI, thus any dark spots in the object need to be filled. Filling is accomplished as explained in section 2.1.2.3 with equations 2-15 and 2-16. However for simulation purposes the built-in MathCAD function was used:

$$D8 := \text{dilate8}(G, d) \qquad \text{4-7}$$

This function performs dilation on a matrix G (the eroded image) at a threshold of 1 using 8 neighbours.

(a) **(b)**

Figure 4-5: The filled result (b) of the binary image of (a)

Observe how the hole on the left of the object is filled. This result was achieved after three fillings. This binary, eroded and filled image can now be used as a ROI since it only consists of information on the object and no background.

4.1.4 Area

The area of the object, in this case a potato, can now be calculated from the processed binary image. The new matrix now consists of 1's where the potato occupies the image and 0's for the background. Therefore the area can be calculated by adding all the values of the matrix together as follows:

$$\text{totalarea} := \sum_{x} \sum_{y} D888_{x,y} \qquad 4\text{-}8$$

where:

$D888_{x,y}$ represents the binary image matrix

x and y are the coordinates.

In this case the answer was 5.354×10^4 pixels. In normal image processing,

calculations must be done to determine what the relation is between pixels and millimetres. However, as this information is fed to a neural network it can be left as is. If the network is taught that one million pixels is a large object and half a million is a small object, then it would not be necessary to be concerned with millimetres, which would only be understandable to human perception.

4.1.5 Shape

The scale of the irregular shape is calculated with the aid of a bounding oval and the area of the binary image outside of it.

The bounding oval has the same area as the object. The two constants needed (width and height) to approximate this oval, as explained in section 2.1.2.5.3, can be calculated by calculating the maximum inertia rotating the solid body around the orientation axis for the one constant. To calculate the other constant, the orientation is at a 90-degree angle through the centre point of gravity.

To illustrate this better a more suitable image was read in and processed as the sample above, resulting in a binary image as follows:

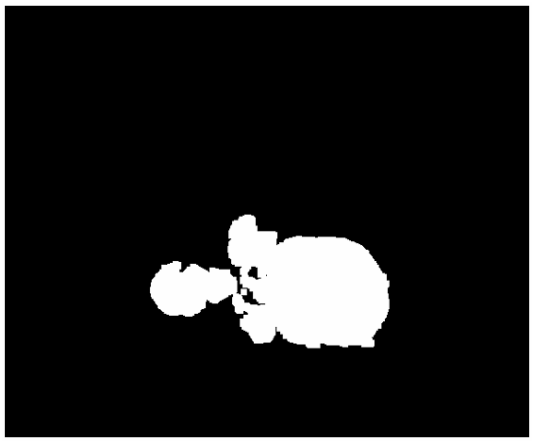


Figure 4-6: A more suitable image to illustrate irregular shape

The potato sample in Figure 4-6 has a prominent irregular shape, and a bounding oval should make an approximation of what it should have been.

For example, note the following figure:

Figure 4-7: Masking the bounding oval out of the object to calculate the overgrown area

Figure 4-7 illustrates the results of the calculations which are given below, where the bounding oval is calculated to approximate a perfect specimen with the same area, centre of gravity and orientation. By using this approximation as a subtraction mask, only the irregularity is left. The next step is to calculate the area of this irregularity with relation to the whole area. With this an indication of the irregular shape can be determined.

4.1.5.1 Centre point of gravity

As explained in section 2.1.2.5.1, the centre of gravity is used as the reference point and is calculated from the moments of zero and first order as given in equation 2-19.

Therefore the moment of 0,0 in MathCAD is as follows:

$$m_{0,0} := \sum_x \sum_y x^i \cdot y^j \cdot D888_{x,y} \qquad 4\text{-}9$$

where:

$D888_{x,y}$ represents the binary image matrix

x and y are the coordinates

i and j are the order of moment.

The moment of 1,0 in MathCAD is as follows:

$$m_{1,0} := \sum_x \sum_y x \cdot D888_{x,y} \qquad 4\text{-}10$$

where:

$D888_{x,y}$ represents the binary image matrix

x and y are the coordinates

i and j are the order of moment.

The moment of 0,1 in MathCAD is as follows:

$$m_{0,1} := \sum_x \sum_y y \cdot D888_{x,y} \qquad 4\text{-}11$$

where:

$D888_{x,y}$ represents the binary image matrix

x and y are the coordinates

i and j are the order of moment.

The next step is to substitute these values into the centre point of gravity coordinate, equation 2-19 as explained in section 2.1.2.5.1. In MathCAD this is as follows:

$$X := \frac{m_{1,0}}{m_{0,0}} \quad Y := \frac{m_{0,1}}{m_{0,0}} \qquad 4\text{-}12$$

where:

m is the moments of a specified order as calculated in equations 4-9, 4-10 and 4-11

X and Y are the coordinates of the centre point of gravity.

To illustrate, lines are drawn through these points as follows:

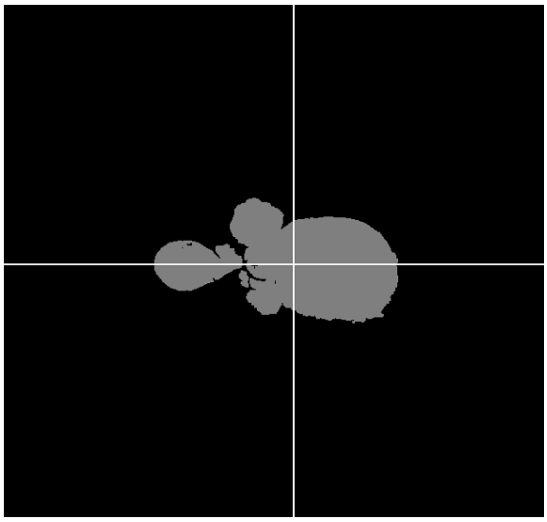


Figure 4-8: The object with an indication of the calculated centre point of gravity

The lines in Figure 4-8 cross at the centre point of gravity calculated above.

4.1.5.2 Orientation

If an object is elongated, the orientation can be calculated even if the object is a perfect circle. Noise in the image would then result in one angle of orientation. The orientation is calculated using equation 2-18 and is implemented in MathCAD as follows:

$$\text{orentation} := \frac{1}{2}\cdot\text{atan}\left[\frac{2\cdot\left[\sum_x\sum_y (x-X)^1\cdot(y-Y)^1\cdot D888_{x,y}\right]}{\sum_x\sum_y (x-X)^2\cdot(y-Y)^0\cdot D888_{x,y} - \sum_x\sum_y (x-X)^0\cdot(y-Y)^2\cdot D888_{x,y}}\right]$$

4-13

where:

$D888_{x,y}$ represents the binary image matrix

x and y are the coordinates

i and j are the order of moment.

This angle can now be added as a gradient to a line for illustration.

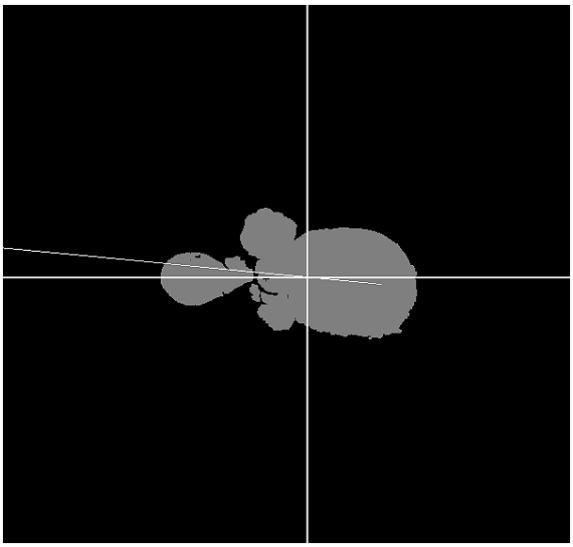


Figure 4-9: The object with a cross through the centre of gravity and an indication line of the orientation through the centre point of gravity

Figure 4-9 illustrates the object with the orientation line drawn through the centre of gravity.

4.1.5.3 Maximum and minimum inertia

Maximum and minimum inertias are not he only feature to measure shape of binary images but it is used in this setup.

As stated in sections 2.1.2.5.3 and 4.1.5, two constants are needed, namely width and height, to approximate an oval, and these constants are the maximum and minimum inertias.

To illustrate the calculation of the first constant, the image is rotated with the orientation angle as follows:

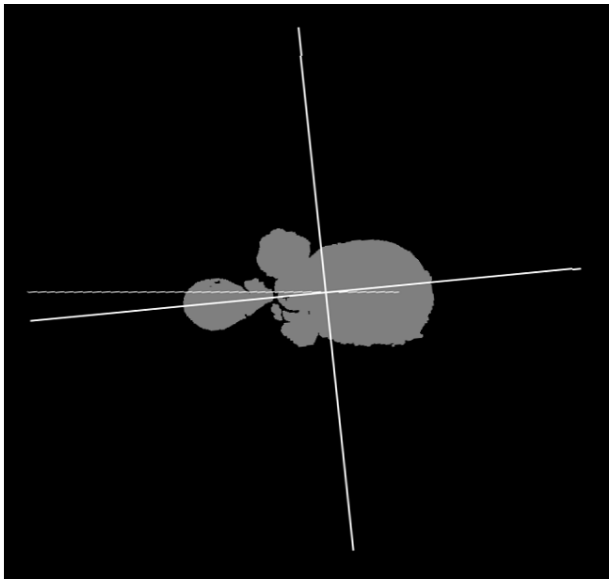


Figure 4-10: The object with the orientation angle as reference

Figure 4-10 is the result after the image was rotated to make the orientation the reference or new x axis. The maximum and minimum inertia for the one constant would now be the coordinates of the pixel furthest away from the orientation axes.

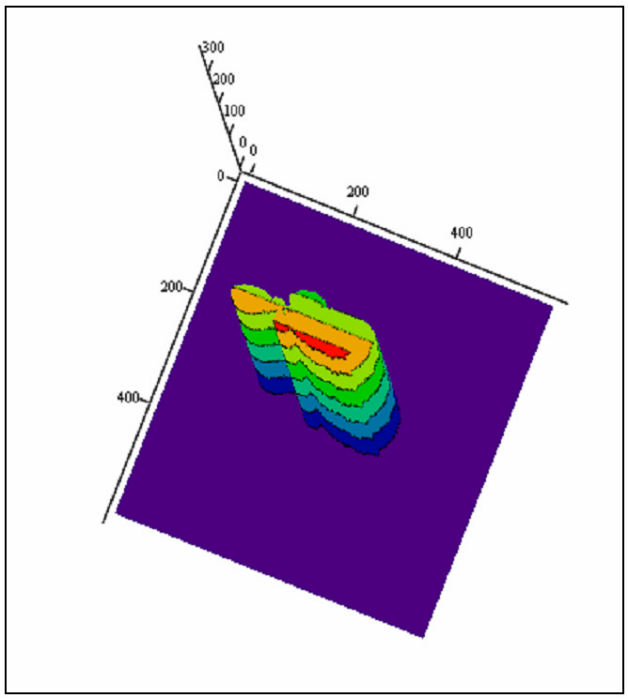


Figure 4-11: A 3D image of the object where the amplitude indicates the distance from the reference axes indicating the maximum inertia

Figure 4-11 illustrates the distance of a pixel to the reference axis. The x axes and y axes are coordinates and the y axes indicate the distance. Hence, the further a certain pixel should be from the reference axes, the bigger the magnitude of the y axes.

The maximum inertia is calculated by determining the maximum distance from the reference axes, and a possible candidate for the one constant of the oval is determined. The reason for a possible candidate is that the minimum inertia might be bigger. To determine this, the image is rotated 180 degrees and the maximum is calculated.

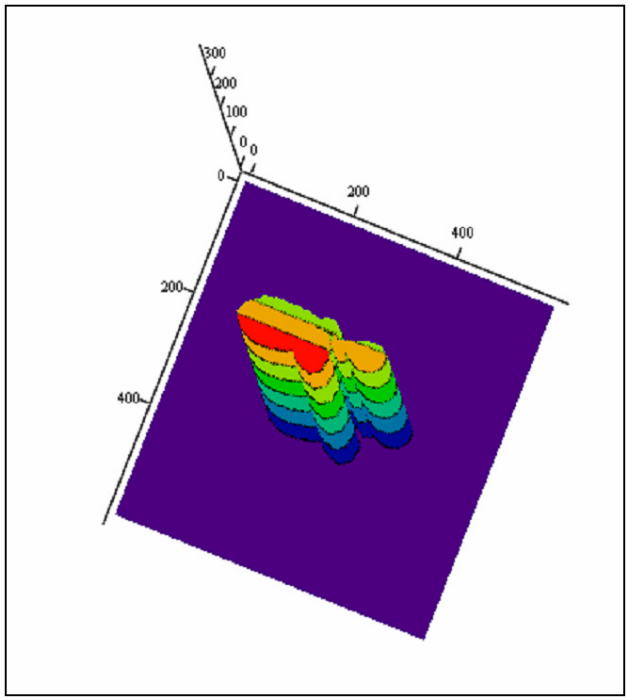


Figure 4-12: 180-degree rotation of a 3D image of the object where the amplitude indicates the distance from the reference axes indicating the minimum inertia

Figure 4-12 shows a 180-degree rotation of Figure 4-11. This illustrates the distance of a pixel to the reference axis, and would be an indication of a minimum. The x axes and y axes are coordinates, and the y axes indicate the distance. Hence, the further a certain pixel should be from the reference axes, the bigger the magnitude of the y axes will be.

The next step is to determine which of the minimum or maximum distances from the reference is the biggest. The bigger one is used as the one constant, for example the width of the oval.

The same must now be done for the other constant, for example the height. To calculate this, the image is rotated 90 degrees to the orientation around the centre of gravity point.

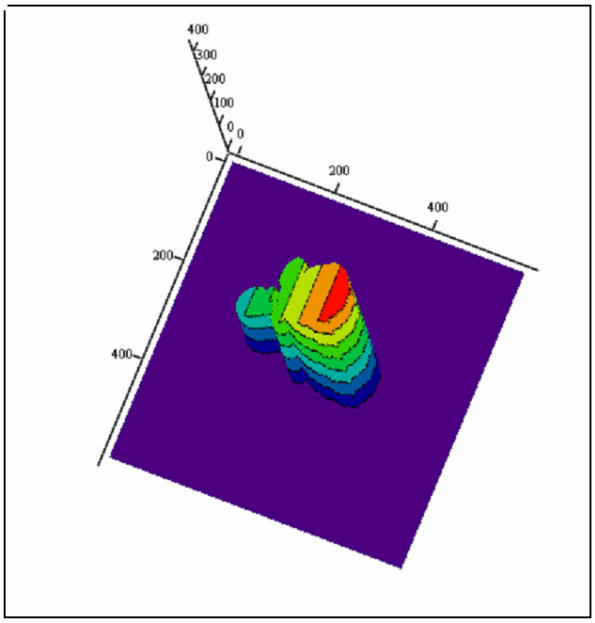


Figure 4-13: A 3D image of the object at a 90-degree rotation, where the amplitude indicates the distance from the reference axes indicating the maximum inertia

Figure 4-13 shows a 90-degree rotation of Figure 4-11 which illustrates the distance of a pixel to the reference axis. This now shows the maximum of the second constant needed. The x axes and y axes are coordinates and the y axes indicate the distance. Hence, the further a certain pixel should be from the reference axes, the bigger the magnitude of the y axes.

To determine the maximum distance from the reference axes, the maximum inertia is calculated as a possible candidate for the one constant of the oval.

As above, the reason that it is only a possible candidate is that the minimum inertia may be bigger.

To determine this, the image is rotated 180 degrees and the maximum is calculated.

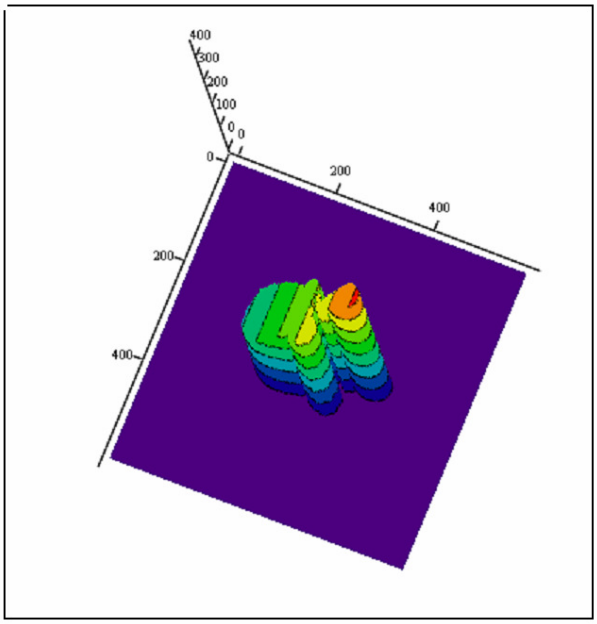


Figure 4-14: 180-degree rotation of the 3D image of the object in Figure 4-13 where the amplitude indicates the distance from the reference axes indicating the minimum inertia

Figure 4-14 is a 180-degree rotation of Figure 4-13 which illustrates the distance of a pixel to the reference axis, but this would be an indication of a minimum. The x axes and y axes are coordinates and the y axes indicate the distance. Hence the further a certain pixel should be from the reference axes, the bigger the magnitude of the y axes.

As above, the next step is to determine the minimum or maximum distance from the reference and which one is the biggest. The bigger one is used as the other constant, for example the height of the oval.

4.1.5.4 Masking oval

The oval and its inside should be used as a mask, therefore the inside of the oval must be filled. This was accomplished by generating a finite number of ovals with constants from zero to the values calculated above. Owing to the limitations of

MathCAD, the oval is divided into two halves, namely the top and bottom halves or the parts above and below the orientation axes halves. The two halves are added together at the end to generate the full oval as needed.

The calculation for the top half is done as follows

$$\text{topy}(h,b) := \left| b \cdot \sqrt{1 - \frac{(h)^2}{a^2}} + (XX) \right| \qquad 4\text{-}14$$

where:

b is the height

a is the width

XX is the x coordinate of the centre point of gravity as calculated with equation 4- 12 but on the rotated image.

$h := -a .. a$

The calculation for the bottom half is done as follows

$$\text{boty}(h,b) := \left| -b \cdot \sqrt{1 - \frac{(h)^2}{a^2}} + (XX) \right| \qquad 4\text{-}15$$

where:

b is the height

a is the width

XX is the x coordinate of the centre point of gravity as calculated with equation 4- 12 but on the rotated image.

$h := -a .. a$

However, as stated, the whole oval must be filled. This is done by generating a range of values for the constant b from 0 to the maximum as calculated above.

$$b := 0..\text{round}\left[\frac{(mxx + XX) - (XX - mxx)}{2}\right] \qquad 4\text{-}16$$

where:

XX is the x coordinate of the centre point of gravity as calculated with formula 4- 12 but on the rotated image mxx is the maximum distance of a pixel from the orientation axes.

The value for a can be calculated as follows:

$$a := \text{round}\left[\frac{(mxy + YY) - (YY - mxy)}{2}\right] \qquad 4\text{-}17$$

where:

YY is the y coordinate of the centre point of gravity as calculated with equation 4-12 but on the rotated image

mxy is the maximum distance of a pixel from the orientation axes.

If this oval is added to the image matrix the result is as follows:

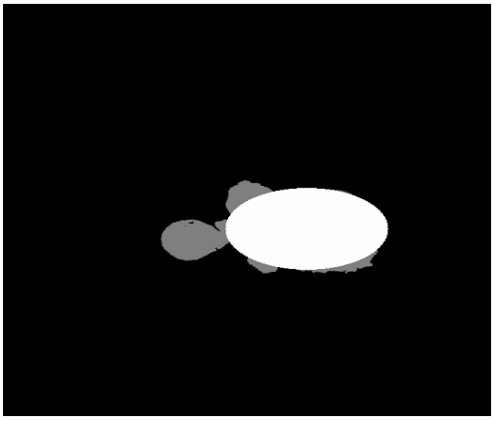


Figure 4-15: The oval added to the image with the object

In Figure 4-15 it is clear that this oval can serve as a mask for the calculation of the magnitude of the overgrown parts.

To mask an image matrix one can multiply the image matrix by the inverse of the oval as in Figure 4-15.

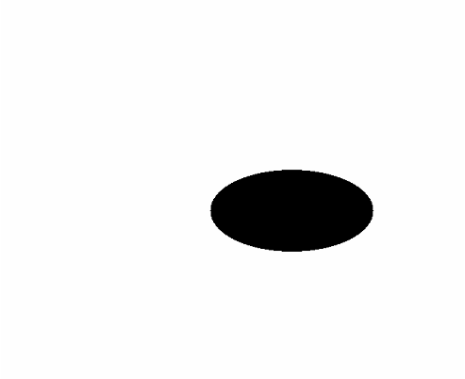


Figure 4-16: The oval inverted to serve as a mask

Figure 4-16 shows the inverse to be multiplied by the image matrix. This entails multiplying the surrounding parts by 1, letting their amplitudes through, and multiplying the parts inside the oval by 0, dissipating it completely.

Figure 4-17: The oval mask applied to the object image

Figure 4-17 shows the result of the applied mask, and all that is left is the overgrown part. The area of the overgrown part is now calculated, and this can be expressed as a percentage of the overall area.

$$\text{overgrown}\% := \frac{\text{area\_over}}{\text{totalarea}} \cdot 100 \qquad \text{4-18}$$

where:

area_over is the overgrown area

totalarea is the total area of the object.

4.1.6 Colour analyses

A colour processing setup is always critical and dependent on the noise and white balance of the camera. The settings of the camera must thus be the same for the training and evaluation set. The colour of the potato is analysed with respect to its area of hue [36, p. 423] [28, p. 1591] [9, p. 12].As stated in section 2.1.1.2, hue is a colour-processing technique, in which all the possible colours are plotted around a circle at certain radians. This implies that a certain colour would be represented by a corresponding angle. To simplify the decision-making process, this colour representation is broken up into areas of green (from yellowish-green to greenish-blue), blue (from blue-green to violet) and red (from violet to reddish-yellow). For example observe the following colour wheel:

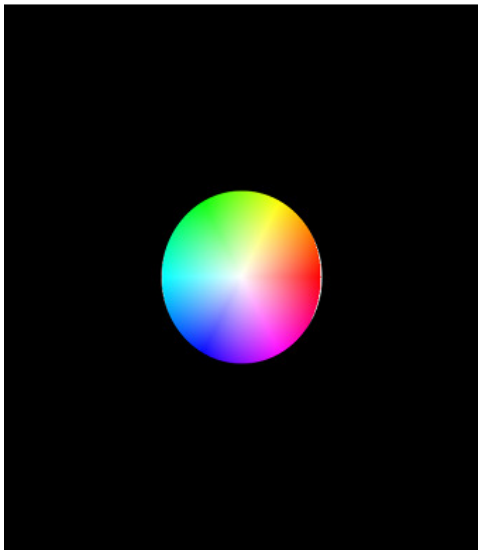


Figure 4-18: Sample colour wheel

Figure 4-18 shows a test colour wheel that would now be divided into the three colour groups as stated above. This image is read into MathCAD as follows:

$$\text{colour\_sample} := \text{READ\_RGB}(S) \qquad \text{4-19}$$

where:

S is the path of the image's file.

The next step is to convert the RGB colour scheme to the required HSV scheme for hue extraction. The application in MathCAD is as follows:

$$\text{hue} := \text{extract}(\text{rgb\_to\_hsv}(\text{colour\_sample}), 1) \qquad \text{4-20}$$

where:

extract is a function that returns the nth (1, 2, or 3) colour component of a packed colour matrix of $\text{rgb\_to\_hsv}(\text{colour\_sample})$

$\mathrm{rgb\_to\_hsv}\,(\mathrm{colour\_sample})$ is the function that returns array RGB in RGB colour representation converted to HSV colour representation

$\mathrm{colour\_sample}$ is the read-in sample image matrix.

The next step is to normalise the hue; this results in an 8-bit black-and-white image that can be thresholded:

$$\mathrm{huenorm} := \frac{\mathrm{hue}}{\max(\mathrm{hue})} \cdot 255 \qquad 4\text{-}21$$

where:

hue is the hue as calculated in 4-19

$\max(\mathrm{hue})$ is a function returning the maximum of the hue.

The result is as follows:

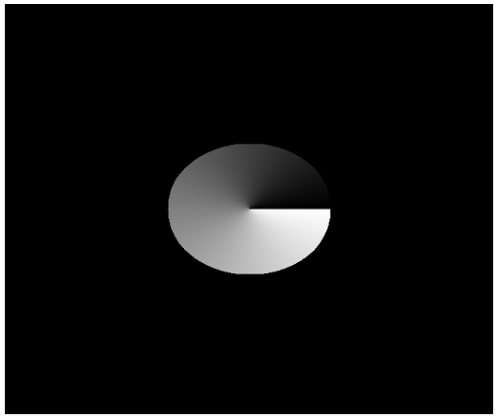


Figure 4-19: The angle of colour (hue) of the sample colour wheel

Figure 4-19 shows the angle as amplitude where 0 is 0 degrees and 255 is equal to 360 degrees. The reason for the range of 0 to 255 is more for illustration purposes than for mathematical sense. The angle can be displayed in an image frame since 0 represents black and 255 represents white and the values in

between would be all the shades of grey. Of course, 255 can also be processed with 8 bits.

As stated above, the colours must be divided into three subgroups for simplification of the decision-making part.

For the area of green (from yellowish-green to greenish-blue), the threshold function is as follows:

$$\text{green}_{i,j} := \text{if}\left(35 \le \text{huenorm}_{i,j} \le 120, 1, 0\right) \qquad \text{4-22}$$

where:

$\text{huenorm}_{i,j}$ is the normalised hue matrix

x and y are the coordinates.

This result is the following:

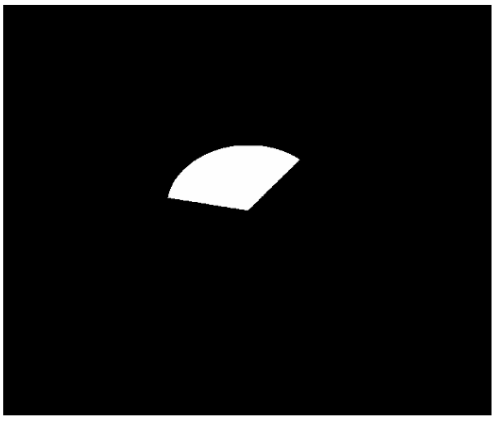


Figure 4-20: The area of green (from yellowish-green to greenish-blue)

This part is then applied as a mask to the colour test wheel. The result is as follows:

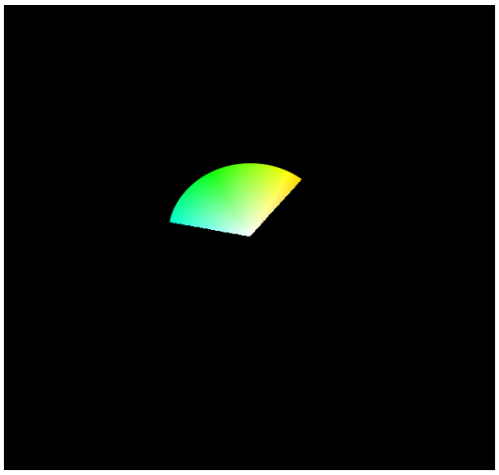


Figure 4-21: The area of green (from yellowish-green to greenish-blue) applied as a mask to the colour image

So for the area of blue (from blue-green to violet) the threshold function is as follows:

$$\text{blue}_{x,y} := \text{if}\left(120 \le \text{huenorm}_{x,y} \le 205, 1, 0\right) \qquad \text{4-23}$$

where:

$\text{huenorm}_{i,j}$ is the normalised hue matrix

x and y are the coordinates.

The result is the following:

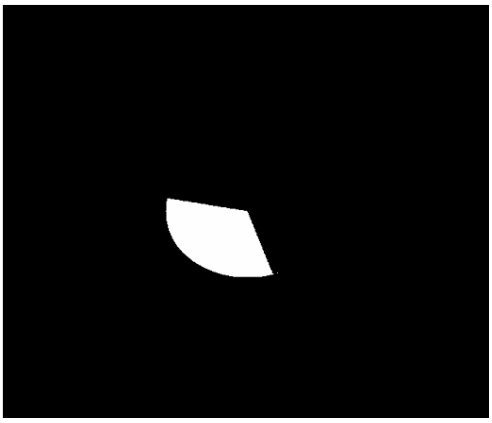


Figure 4-22: The area of blue (from blue-green to violet)

This part is then applied as a mask to the colour test wheel. The result is as follows:

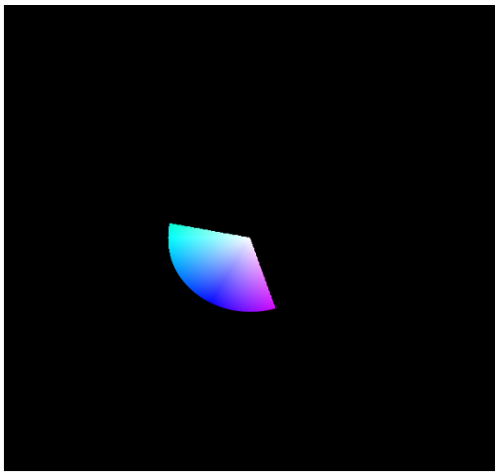


Figure 4-23: The area of blue (from blue-green to violet) applied as a mask to the colour image

Red is slightly different since it consists of a low and a high range of angles. For the first section area of red, the threshold function is as follows:

$$red1_{x,y} := if\left(1 \le huenorm_{x,y} \le 35, 1, 0\right) \quad \text{4-24}$$

where:

$huenorm_{i,j}$ is the normalised hue matrix

x and y are the coordinates.

The result is the following:

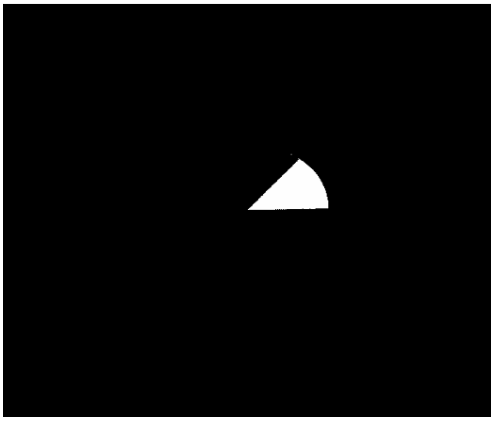


Figure 4-24: The first section of red

For the second section area of red, the threshold function is as follows:

$$red2_{x,y} := if\left(205 \le huenorm_{x,y} \le 255, 1, 0\right) \quad \text{4-25}$$

where:

$huenorm_{i,j}$ is the normalised hue matrix

x and y are the coordinates.

The result is the following:

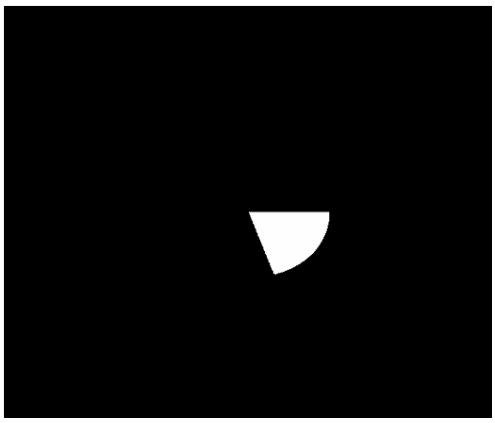


Figure 4-25: The second section of red

The total area of red (from violet to reddish-yellow) is as follows:

$$red_{x,y} := red1_{x,y} + red2_{x,y} \qquad 4\text{-}26$$

where:

$red1_{x,y}$ is the red area matrix as calculated in equation 4-23

$red2_{x,y}$ is the red area matrix as calculated in equation 4-24.

The result is the following:

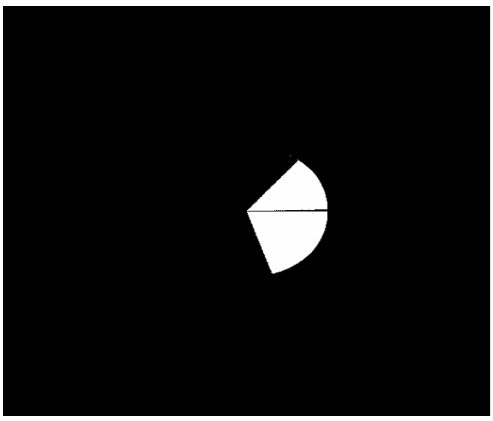


Figure 4-26: The total area of red (from violet to reddish-yellow)

These parts are then applied as a mask to the colour test wheel. The result is as follows:

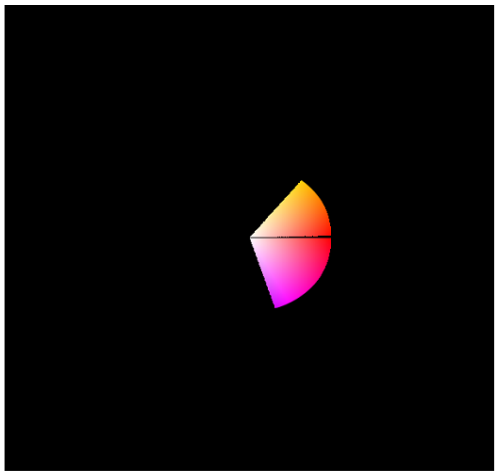


Figure 4-27: The area of red (from violet to reddish-yellow) applied as a mask to the colour image

Figure 4-21, Figure 4-23 and Figure 4-27 show the three divisions of the decision-making part.

The next step is to calculate the area of each section to calculate the magnitude of the colour of a sample.

This is better explained by illustration. These algorithms are now applied to a potato sample:

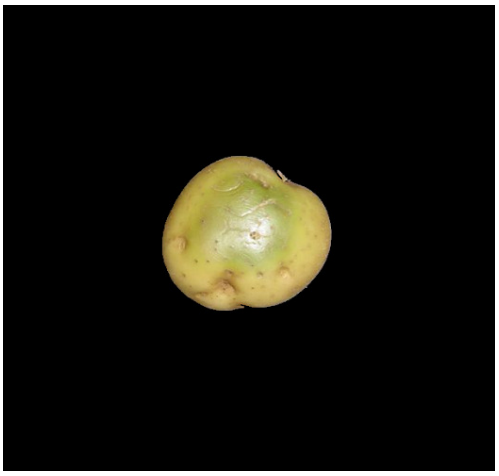


Figure 4-28: The potato sample that will be colour processed

The image is read in as in equation 4-18 and converted to the HSV scheme as in equation 4-19. This is then normalised as in equation 4-20. This image is now thresholded into the three sections of green, blue and red. The result of the green thresholding is as follows:

Figure 4-29: The area of green (from yellowish-green to greenish-blue)

This area must be calculated to determine the magnitude of green in the sample.

For explanatory purposes this is now applied as a mask to the colour image:

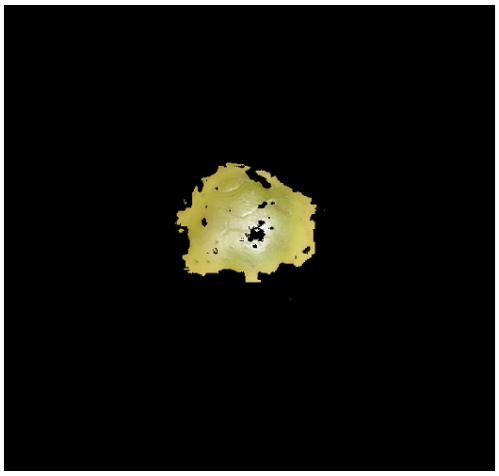


Figure 4-30: The area of green (from yellowish-green to greenish-blue) applied as a mask to the colour image

From Figure 4-30 it is clear that the algorithm is successful as only the parts that are yellowish-green to greenish-blue are shown.

Now follows the thresholding for blue:

Figure 4-31: The area of blue (from blue-green to violet)

This area must be calculated to determine the magnitude of blue in the sample.

For explanatory purposes this is now applied as a mask to the colour image:

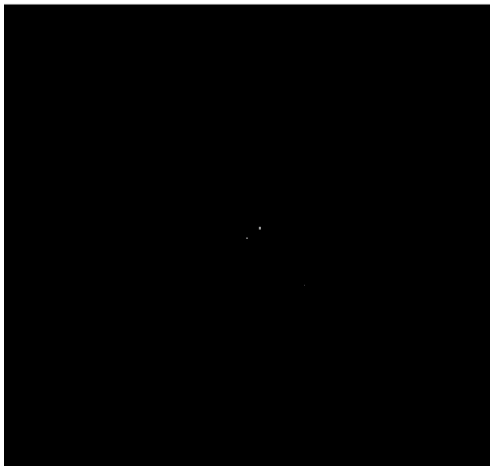


Figure 4-32: The area of blue (from blue-green to violet) applied as a mask to the colour image

As noted the specimen does not consist of blue-green to violet components. This result can be fed into an intelligent decision-making system. It may be that some other components will be graded. One cannot ignore this colour section. Next follows the thresholding of the red:

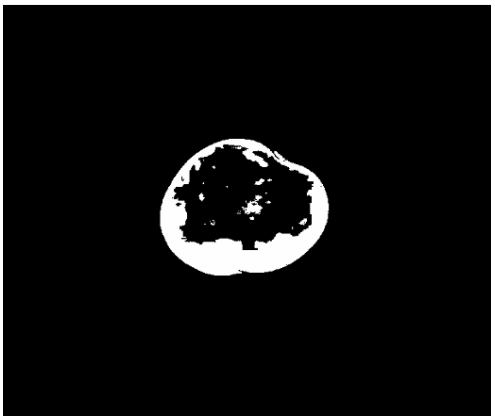


Figure 4-33: The total area of red (from violet to reddish-yellow)

This area must be calculated to determine the magnitude from violet to reddish-yellow in the sample. For explanatory purposes this is now applied as a mask to the colour image:

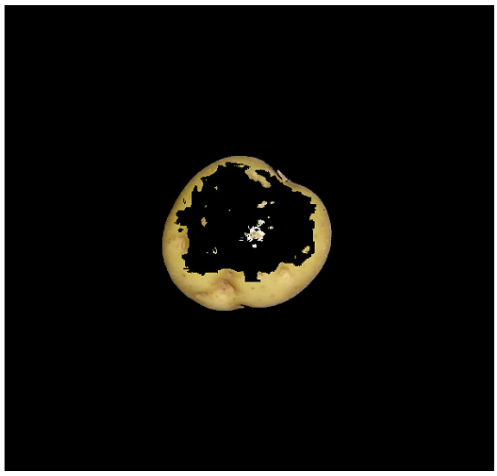


Figure 4-34: The area of red (from violet to reddish-yellow) applied as a mask to the colour image

Figure 4-34 shows the violet to reddish-yellow section, of which the reddish yellow dominates.

As stated above, the area of each section should be calculated to give an indication of the colour of the specimen.

For simplification, the area is expressed as a percentage of the total area where the total area is calculated using equation 4-8.

The algorithm to calculate the green area is as follows:

$$\text{areagreen} := \sum_{x} \sum_{y} \text{green}_{x,y} \qquad \text{4-27}$$

where:

$\text{green}_{x,y}$ represents the binary image matrix of the threshold Section of green

x and y the coordinates.

As stated, this is expressed as a percentage of the total area:

$$\text{green\%} := \frac{\text{areagreen}}{\text{totalarea}} \cdot 100 \qquad \text{4-28}$$

where:

areagreen is the area of the green section

totalarea is the total area of the potato.

The algorithm to calculate the blue is as follows:

$$\text{areablue} := \sum_{x} \sum_{y} \text{blue}_{x,y} \qquad \text{4-29}$$

where:

$\text{blue}_{x,y}$ represents the binary image matrix of the threshold section of green

x and y are the coordinates. As stated, this is expressed as a percentage of the total area:

$$\text{blue\%} := \frac{\text{areablue}}{\text{totalarea}} \cdot 100 \qquad \text{4-30}$$

where:

areablue is the area of the blue section

totalarea is the total area of the potato.

The algorithm to calculate the red area is as follows:

$$\text{areared} := \sum_{x} \sum_{y} \text{red}_{x,y} \qquad \text{4-31}$$

where:

$\text{red}_{x,y}$ represents the binary image matrix of the threshold section of green

x and y are the coordinates.

As stated, this is expressed as a percentage of the total area:

$$\text{red}\% := \frac{\text{areared}}{\text{totalarea}} \cdot 100 \qquad \text{4-32}$$

where:

areared is the area of the blue section

totalarea is the total area of the potato.

These percentages can now be fed into a decision-making system. This may lead to functions in the decision-making process along the line of: “if the area of green is larger than that of red, reject potato”.

4.2 Neural networks

As described in section 3.5, a neural network must be implemented with one hundred inputs consisting of percentages of size, shape, red, green and blue via the weighted connections to the four neurons in the centre. These are then connected via their weighted connections to another four neurons that have one output each producing the result.

4.2.1 Transfer functions

The percentages of size, shape, red and green need to be transferred to binary as required by the network. This is done as described in section 3.5 through an incrementing step function.

The first step in design is to initialise a range in MathCAD to simulate the input percentages from 0 to 100.

$$size := 0 .. 100 \qquad \text{4-33}$$

Next follows the incrementing step function, which is a simple IF statement code that will divide the inputs up into steps from binary 0000000000 0000000000 for 0 up to 5 (but not including 5) and binary 0000000000 0000000001 for 5 up to 10 (but not including 10) and so forth, up to 100 which will be binary 1111111111 1111111111. This IF statement also returns a 0.1 for binary 0 and 0.9 for binary 1.

The IF statement function is as follows:

$$\begin{aligned}
&\text{step1}(\text{size}) := \text{if}(1 \le \text{size_out}(\text{size}), 0.9, 0.1) \\
&\text{step2}(\text{size}) := \text{if}(5 \le \text{size_out}(\text{size}), 0.9, 0.1) \\
&\text{step3}(\text{size}) := \text{if}(10 \le \text{size_out}(\text{size}), 0.9, 0.1) \\
&\text{step4}(\text{size}) := \text{if}(15 \le \text{size_out}(\text{size}), 0.9, 0.1) \\
&\text{step5}(\text{size}) := \text{if}(20 \le \text{size_out}(\text{size}), 0.9, 0.1) \\
&\text{step6}(\text{size}) := \text{if}(25 \le \text{size_out}(\text{size}), 0.9, 0.1) \\
&\text{step7}(\text{size}) := \text{if}(30 \le \text{size_out}(\text{size}), 0.9, 0.1) \\
&\text{step8}(\text{size}) := \text{if}(35 \le \text{size_out}(\text{size}), 0.9, 0.1) \\
&\text{step9}(\text{size}) := \text{if}(40 \le \text{size_out}(\text{size}), 0.9, 0.1) \\
&\text{step10}(\text{size}) := \text{if}(45 \le \text{size_out}(\text{size}), 0.9, 0.1) \\
&\text{step11}(\text{size}) := \text{if}(50 \le \text{size_out}(\text{size}), 0.9, 0.1) \\
&\text{step12}(\text{size}) := \text{if}(55 \le \text{size_out}(\text{size}), 0.9, 0.1) \\
&\text{step13}(\text{size}) := \text{if}(60 \le \text{size_out}(\text{size}), 0.9, 0.1) \\
&\text{step14}(\text{size}) := \text{if}(65 \le \text{size_out}(\text{size}), 0.9, 0.1) \\
&\text{step15}(\text{size}) := \text{if}(70 \le \text{size_out}(\text{size}), 0.9, 0.1) \\
&\text{step16}(\text{size}) := \text{if}(75 \le \text{size_out}(\text{size}), 0.9, 0.1) \\
&\text{step17}(\text{size}) := \text{if}(80 \le \text{size_out}(\text{size}), 0.9, 0.1) \\
&\text{step18}(\text{size}) := \text{if}(85 \le \text{size_out}(\text{size}), 0.9, 0.1) \\
&\text{step19}(\text{size}) := \text{if}(90 \le \text{size_out}(\text{size}), 0.9, 0.1) \\
&\text{step20}(\text{size}) := \text{if}(95 \le \text{size_out}(\text{size}), 0.9, 0.1)
\end{aligned} \qquad 4\text{-}34$$

where:

$\text{size\_out}(\text{size})$ is the transfer function of these percentages.

This would then convert the percentages to incriminating adapted binary inputs.

However, as explained in section 3.5, more processing is needed on the percentages of size and shape. These have to go through a sigmoid and log function respectively before the step function is applied.

4.2.1.1 Adapted sigmoid transfer function for the size percentages

The transfer function for the size percentages must be sigmoid and, as stated in section 3.5, must be adapted so that the best resolution is in the 50% range.

The adaptation is designed to give the better resolution between 20% and 80%, of which the best part is at 50%. This transfer function is as follows:

$$\text{size\_out}(\text{size}) := \frac{100}{\left[1 + e^{-1\left[\frac{(\text{size}-50)}{11}\right]}\right]} \qquad 4\text{-}35$$

where:

size is the area percentage of the object.

The size input for this evaluation given in equation 4-32 is from 0 to 100, indicating 0% to 100%, of which the result of the function 4-34 is as follows:

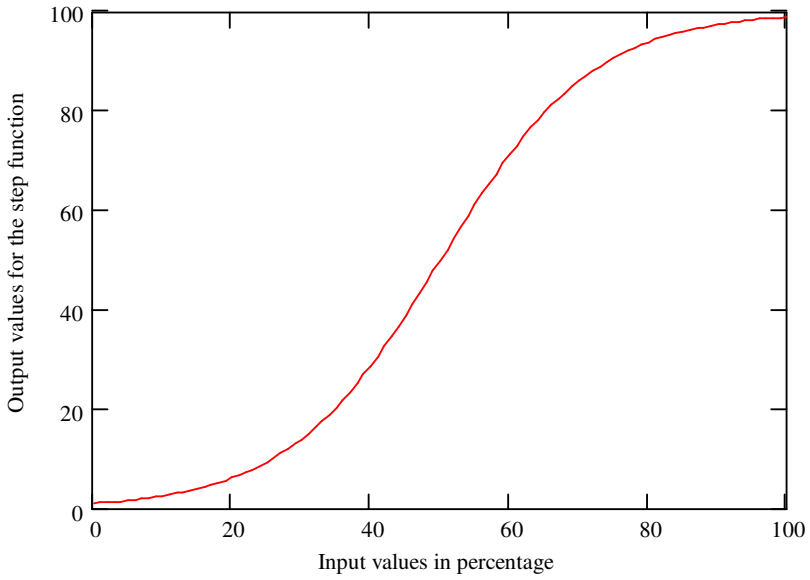


Figure 4-35: The sigmoid transfer function for the size percentage input

In Figure 4-35 the x axis represents the input values in percentage of the size. The y axis represents the output values that will be fed to the incrementing step function as in equation 4-33. To illustrate this concept better, one can plot the result in a three-dimensional space.

$$\text{step3d}_{\text{size},1} := \text{step1}(\text{size}) \qquad \text{4-36}$$
$$\text{step3d}_{\text{size},2} := \text{step2}(\text{size})$$
$$\text{step3d}_{\text{size},3} := \text{step3}(\text{size})$$
$$\text{step3d}_{\text{size},4} := \text{step4}(\text{size})$$
$$\text{step3d}_{\text{size},5} := \text{step5}(\text{size})$$
$$\text{step3d}_{\text{size},6} := \text{step6}(\text{size})$$
$$\text{step3d}_{\text{size},7} := \text{step7}(\text{size})$$
$$\text{step3d}_{\text{size},8} := \text{step8}(\text{size})$$
$$\text{step3d}_{\text{size},9} := \text{step9}(\text{size})$$
$$\text{step3d}_{\text{size},10} := \text{step10}(\text{size})$$
$$\text{step3d}_{\text{size},11} := \text{step11}(\text{size})$$
$$\text{step3d}_{\text{size},12} := \text{step12}(\text{size})$$
$$\text{step3d}_{\text{size},13} := \text{step13}(\text{size})$$
$$\text{step3d}_{\text{size},14} := \text{step14}(\text{size})$$
$$\text{step3d}_{\text{size},15} := \text{step15}(\text{size})$$
$$\text{step3d}_{\text{size},16} := \text{step16}(\text{size})$$
$$\text{step3d}_{\text{size},17} := \text{step17}(\text{size})$$
$$\text{step3d}_{\text{size},18} := \text{step18}(\text{size})$$
$$\text{step3d}_{\text{size},19} := \text{step19}(\text{size})$$
$$\text{step3d}_{\text{size},20} := \text{step20}(\text{size})$$

These declarations would now plot the number of times a single step would be active until the input values have incremented enough to activate the next step as well. The 3D plot is as follows:

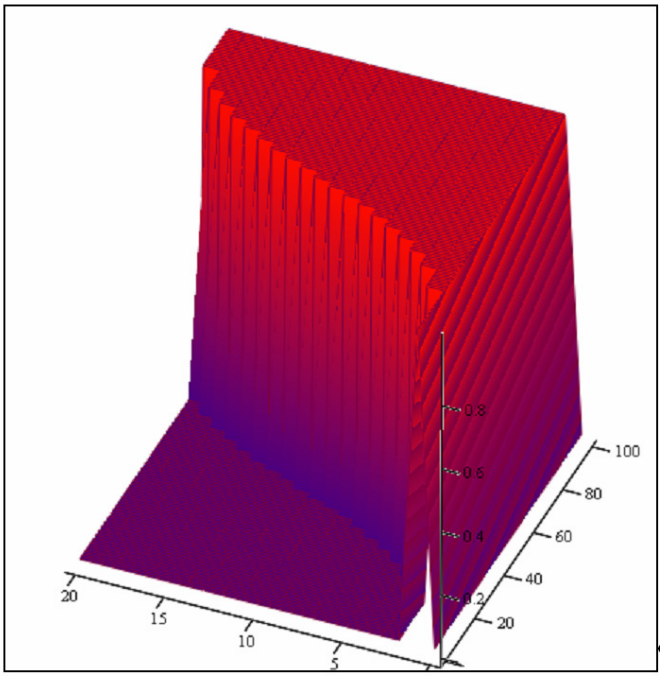


Figure 4-36: The sigmoid transfer function for the size percentage input plotted in 3D, illustrating the transfer resolution of the step function

In Figure 4-36 the x axis (from 0 to 100) represents the input size percentage, and the y axis (0 to 20) represents the steps or activation inputs of the neural network. The z axis (0.1 to 0.9) represents the binary magnitude of which binary 0 is 0.1 and binary 1 is 0.9. Note the small distance in activation in the 50% range. Next observe when it is turned so that it is seen directly from above:

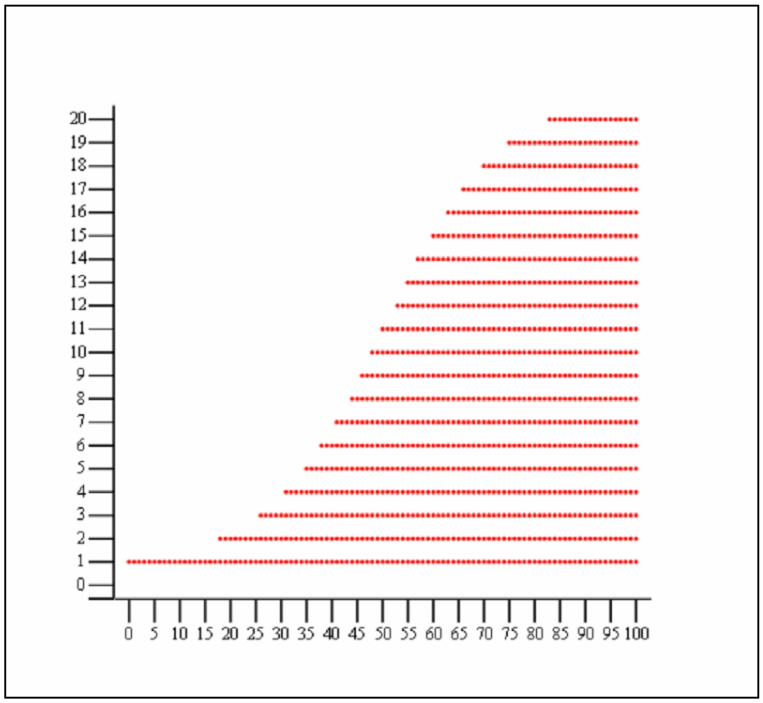


Figure 4-37: The sigmoid transfer function for the size percentage input plotted in 3D, illustrating the transfer resolution of the step function as seen directly from above

In Figure 4-37 the x axis (from 0 to 100) represents the input size percentage, and the y axis (0 to 20) represents the steps or activation inputs of the neural network. The z axis (0.1 to 0.9), which cannot be seen clearly due to the view, is the binary magnitude, of which binary 0 is 0.1 and binary 1 is a 0.9. Figure 4-37 is set up so that a dot represents a binary 1. The resolution transfer is now clearly seen. Note that from 0% to 1% none of the activation steps on the y axes are on.. The first activation step is on from 1% to the end, and the only time the second one comes on is when it is closer to the 20% marker. Thereafter the distance

between the steps becomes smaller until the best resolution is reached at 50%. After this the distances increase again. As such high resolution is not needed at the high end. This would keep the network's inputs to a minimum without losing sensitivity in the operating range.

4.2.1.2 Adapted log transfer function for the shape percentages

As explained in section 3.5, the transfer function of the shape percentages must be a log function. It was also stated in section 3.5 that it needs to be adapted so that the best resolution is in the lower ranges. The adaptation in design is to give the better resolution between 0% and 50%, of which the best part is between 0% and 20%. This transfer function is as follows:

$$\text{size\_out}(\text{size}) := 49.9 \log(\text{size}) \qquad \text{4-37}$$

where:

size is the percentage of the irregular shape of the object. The shape input for this evaluation is, as seen in equation 4-32, from 0 to 100, indicating 0% to 100% ,of which the result of the function 4-36 is as follows:

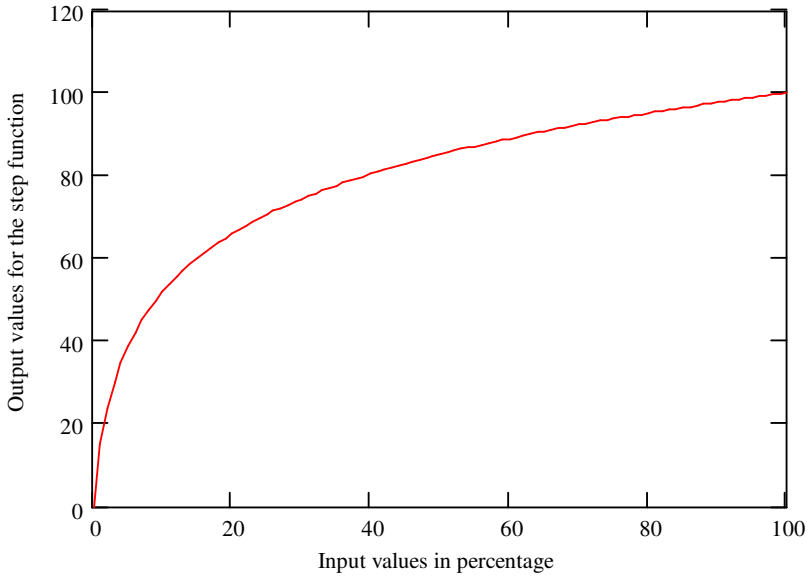


Figure 4-38: The log transfer function for the shape percentage input

In Figure 4-38 the x axis represents the input values in percentage of the shape. The y axis represents the output values that will be fed to the incrementing step function as in equation 4-33. To illustrate this concept better, one can plot the result in a three-dimensional space as before, using equation 4-35.

These declarations would now plot the number of times a single step would be active until the input values have incremented enough to activate the next step as well. This 3D plot is as follows:

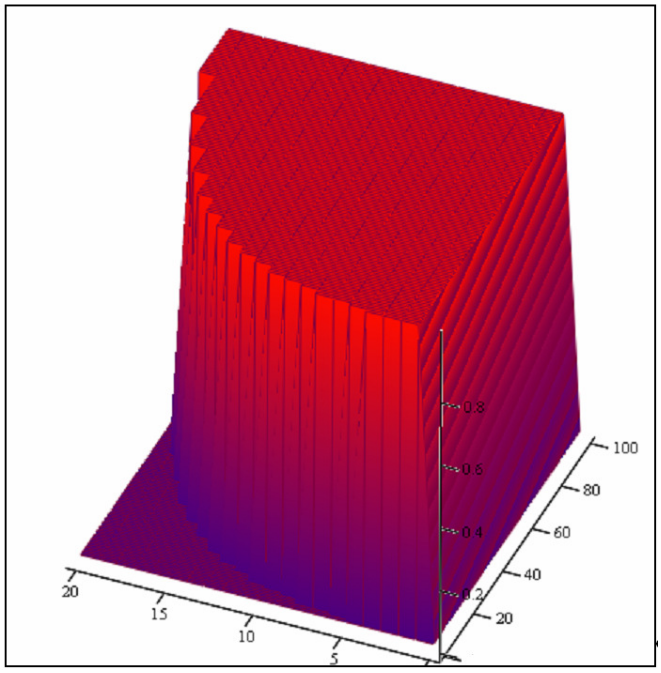


Figure 4-39: The log transfer function for the shape percentage input plotted in 3D, illustrating the transfer resolution of the step function

In Figure 4-39 the x axis (from 0 to 100) represents the input shape percentage, and the y axis (0 to 20) represents the steps or activation inputs of the neural network. The z axis (0.1 to 0.9) represents the binary magnitude, of which binary 0 is 0.1 and binary 1 is 0.9. Note the small distance in activations in the 0% to 20% range. Next observe when it is turned so that it is seen directly from above:

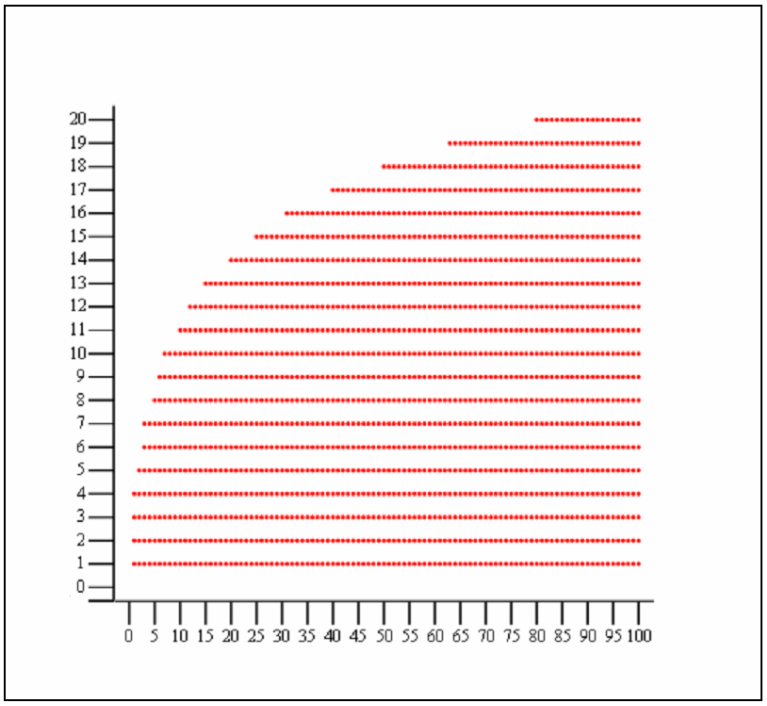


Figure 4-40: The log transfer function for the shape percentage input plotted in 3D, illustrating the transfer resolution of the step function as seen directly from above

In Figure 4-40 the x axis (from 0 to 100) represents the input shape percentage, and the y axis (0 to 20) represents the steps or activation inputs of the neural network. The z axis (0.1 to 0.9), which cannot be seen clearly due to the view, is the binary magnitude of which binary 0 is 0.1 and binary 1 is a 0.9. Figure 4-40 is set up so that a dot represents a binary 1. The resolution transfer is now clearly seen. Note that from 0% to 1% none of the activation steps on the y axis are on. The first activation step is on from 1% to the end and the second one comes on almost immediately and so fourth until closer to the 20% marker where the

distance starts to increase. This keeps the network's inputs to a minimum without losing sensitivity in the operating range.

4.2.1.3 Linear function for the colour percentages

As stated in section 3.5, the inputs of red, green and blue cannot be transformed since they have a linear relationship with each other. These are therefore simulated by applying the declaration 4-32, again generating the values of 0 to 100. This then simulates the possible inputs of 0% to 100%.

Since there is no transfer function, the inputs are directly put through the step function 4-33. One can state that:

$$\text{size\_out}\ (\text{size}) := \text{size} \qquad \text{4-38}$$

where:

size is the red, green or blue percentages of the object.

The linear relationship of the red, green or blue for this evaluation is:

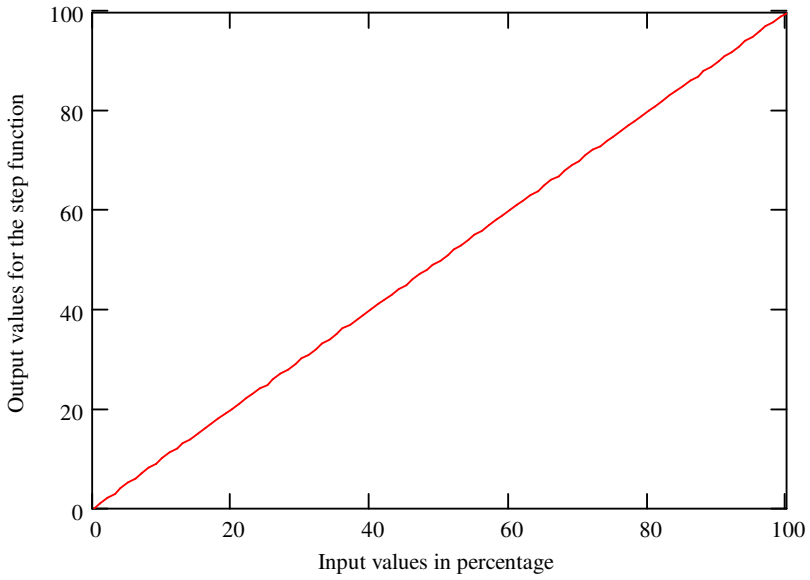


Figure 4-41: The transfer function for the size percentage inputs

In Figure 4-41 the x axis represents the input values in percentages of the separate colours. The y axis represent the output values that will be fed to the incrementing step. For purposes of illustration, this is placed in a 3D space with 4-35.

These declarations now plot the number of times a single step would be active until the input values have incremented sufficiently to activate the next step as well. This 3D plot is as follows:

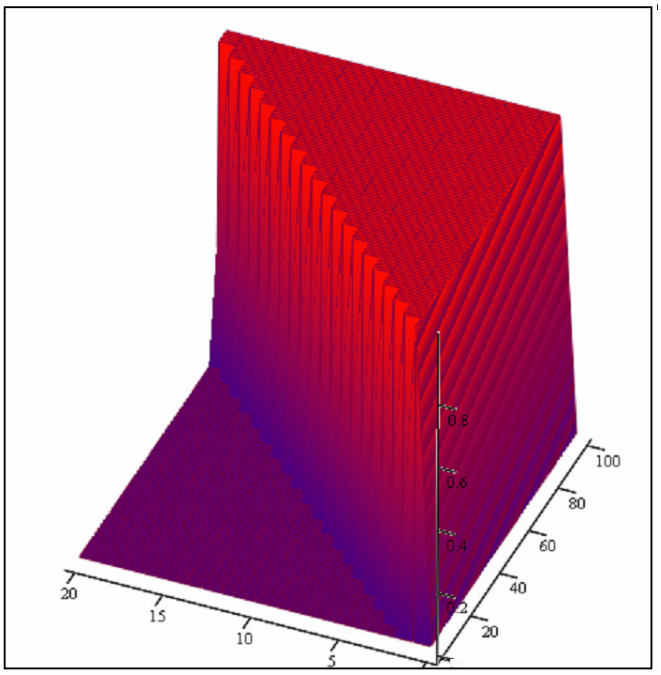


Figure 4-42: The transfer function for the different colour percentage input plotted in 3D, illustrating the resolution of the step function

In Figure 4-36 the x axis (from 0 to 100) represents the red, green or blue percentage input, and the y axis (0 to 20) represents the steps or activation inputs of the neural network. The z axis (0.1 to 0.9) represents the binary magnitude of which binary 0 is 0.1 and binary 1 is 0.9. Note the linear distance between steps. Next observe when it is turned so that it is seen directly from above:

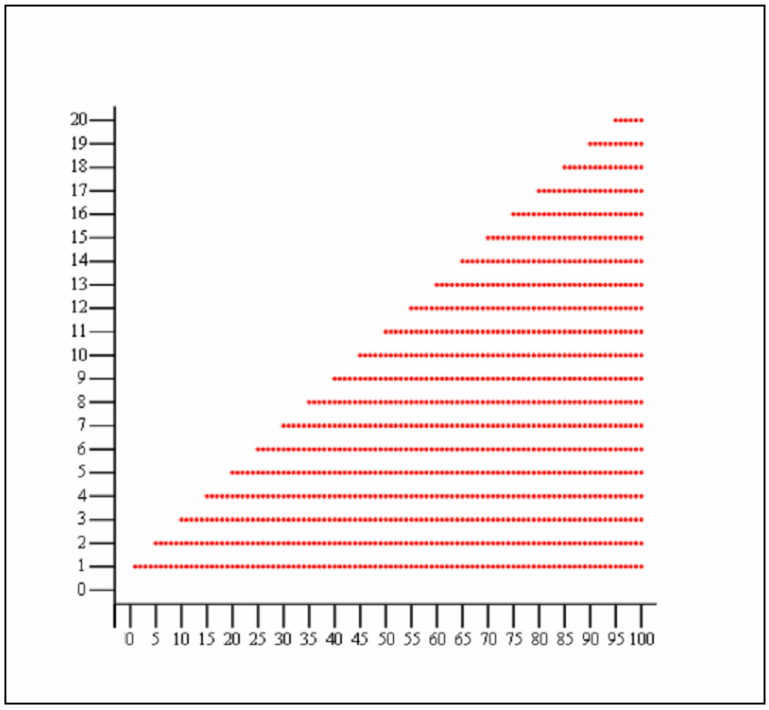


Figure 4-43: The transfer function for the colour percentage input plotted in 3D, illustrating the resolution of the step function as seen directly from above

In Figure 4-43 the x axis (from 0 to 100) represents the percentage input of red, green or blue, and the y axis (0 to 20) represents the steps or activation inputs of the neural network. The z axis (0.1 to 0.9), which cannot be seen clearly due to the view, is the binary magnitude of which binary 0 is 0.1 and binary 1 is a 0.9. Figure 4-43 is set up so that a dot represents a binary 1. Note that from 0% to 1% none of the activation steps on the y axis are on. The first activation step is on from 5% to the end and the second one comes on at 10%. From here on the distance remains at 5% increments between steps.

4.2.2 A one hundred input to four neurons to another four neurons with a four-output neural network

As described in section 3.5, a neural network is implemented with one hundred inputs of percentages of size, shape, red, green and blue, of which size and shape first go through a transfer function. This is now coded in MathCAD to demonstrate and evaluate the learning ability.

The first step is to define a few variables. Consider the first neuron. This neuron would have one activation input followed by one hundred real-world inputs. Thus weight 1 is activation input one. Weights 2 to 21 would be the twenty inputs from the incrementing step function of area; weights 22 to 41 would be the twenty inputs from the incrementing step function of shape; weights 42 to 61 would be the twenty inputs from the incrementing step function of red; weights 62 to 81 would be the twenty inputs from the incrementing step function of green, and weights 82 to 101 would be the twenty inputs from the incrementing step function of blue. This would be repeated for all four input neurons. This series must be divided for later use in the neural network matrix.

The first series:

$$A := 2..21 \qquad \text{4-39}$$

The second series:

$$B := 22..41 \qquad \text{4-40}$$

The third series:

$$C := 42..61 \qquad \text{4-41}$$

The fourth series:

$$D := 62..81 \qquad 4\text{-}42$$

The fifth series:

$$E := 82..101 \qquad 4\text{-}43$$

Next the series for the ranges of the first neurons follows:

$$i := 1..101 \qquad 4\text{-}44$$

The series for the ranges of the next layer of neurons then follows. The previous layer that connects to this one has only four neurons, meaning that a neuron in this layer would have four inputs plus one activation input, ending in a total of five input weights. This series is as follows:

$$ii := 1..5 \qquad 4\text{-}45$$

The next series numbers the total of eight neurons:

$$j := 1..8 \qquad 4\text{-}46$$

The purpose of the following steps is to deduce possible inputs to evaluate whether the network is adaptable. The input of size is:

$$area_A := .9 \qquad 4\text{-}47$$

where:

$area$ is the input matrix values of a simulated area input, ranging with the values of A as defined in equation 4-38.

The input of shape is:

$$shape_B := .5 \qquad 4\text{-}48$$

where:

shape is the input matrix values of a simulated shape input, ranging with the values of B as defined in equation 4-39.

The input of red is:

$$\text{red}_C := .6 \qquad 4\text{-}49$$

where:

red is the input matrix values of a simulated red input, ranging with the values of C as defined in equation 4-40.

The input of green is:

$$\text{green}_D := .2 \qquad 4\text{-}50$$

where:

green is the input matrix values of a simulated green input, ranging with the values of D as defined in equation 4-41.

The input of blue is:

$$\text{blue}_E := .1 \qquad 4\text{-}51$$

where:

blue is the input matrix values of a simulated blue input, ranging with the values of E as defined in equation 4-42.

The next step is to generate initial constant values for the weights of the various neuron inputs. As stated, these must be between zero and a one.

The matrix of the weights of the first neuron is:

$$w_{i,1} := 0.2 \qquad 4\text{-}52$$

where:

i is the range as defined in equation 4-43

1 is the number of the neuron.

Thus for the rest of the four input neuron weights the matrix declaration is as follows:

$$w_{i,2} := .3 \qquad 4\text{-}53$$

$$w_{i,3} := 0.1 \qquad 4\text{-}54$$

$$w_{i,4} := 0.5 \qquad 4\text{-}55$$

where:

i is the range as defined in equation 4-43

2, 3 or 4 is the number of the neuron.

The next step is to generate initial constants for the weights of the various neuron inputs in the second layer. As stated, these must be between zero and one. The matrix of the weights of the first neuron of the second layer or neuron number 5 is calculated as follows:

$$w_{ii,5} := 0.7 \qquad 4\text{-}56$$

where:

ii is the range as defined in equation 4-44

5 is the number of the neuron.

Thus for the rest of the four input neurons weights the matrix declaration is as follows:

$$w_{ii,6} := .3 \qquad 4\text{-}57$$

$$w_{ii,7} := 0.4 \qquad 4\text{-}58$$

$$w_{ii,8} := 0.1 \qquad 4\text{-}59$$

where:

ii is the range as defined in equation 4-43.

6, 7 or 8 is the number of the neuron:

The total matrix of the initial generated constants of all the weights is thus as given in Table 2:

Table 2: The initial values of the weights for all the neurons

w =

	1	2	3	4	5	6	7
1	0.2	0.3	0.1	0.5	0.7	0.3	0.4
2	0.2	0.3	0.1	0.5	0.7	0.3	0.4
3	0.2	0.3	0.1	0.5	0.7	0.3	0.4
4	0.2	0.3	0.1	0.5	0.7	0.3	0.4
5	0.2	0.3	0.1	0.5	0.7	0.3	0.4
6	0.2	0.3	0.1	0.5	0	0	0
7	0.2	0.3	0.1	0.5	0	0	0
8	0.2	0.3	0.1	0.5	0	0	0
9	0.2	0.3	0.1	0.5	0	0	0
10	0.2	0.3	0.1	0.5	0	0	0
11	0.2	0.3	0.1	0.5	0	0	0
12	0.2	0.3	0.1	0.5	0	0	0
13	0.2	0.3	0.1	0.5	0	0	0
14	0.2	0.3	0.1	0.5	0	0	0
15	0.2	0.3	0.1	0.5	0	0	0
16	0.2	0.3	0.1	0.5	0	0	0
17	0.2	0.3	0.1	0.5	0	0	0
18	0.2	0.3	0.1	0.5	0	0	0
19	0.2	0.3	0.1	0.5	0	0	0
20	0.2	0.3	0.1	0.5	0	0	0
21	0.2	0.3	0.1	0.5	0	0	0
22	0.2	0.3	0.1	0.5	0	0	0
23	0.2	0.3	0.1	0.5	0	0	0
24	0.2	0.3	0.1	0.5	0	0	0
25	0.2	0.3	0.1	0.5	0	0	0
26	0.2	0.3	0.1	0.5	0	0	0
27	0.2	0.3	0.1	0.5	0	0	0
28	0.2	0.3	0.1	0.5	0	0	0
29	0.2	0.3	0.1	0.5	0	0	0
30	0.2	0.3	0.1	0.5	0	0	0
31	0.2	0.3	0.1	0.5	0	0	0
32	0.2	0.3	0.1	0.5	0	0	0
33	0.2	0.3	0.1	0.5	0	0	0
34	0.2	0.3	0.1	0.5	0	0	0
35	0.2	0.3	0.1	0.5	0	0	0
36	0.2	0.3	0.1	0.5	0	0	0
37	0.2	0.3	0.1	0.5	0	0	0
38	0.2	0.3	0.1	0.5	0	0	0
39	0.2	0.3	0.1	0.5	0	0	0
40	0.2	0.3	0.1	0.5	0	0	0
41	0.2	0.3	0.1	0.5	0	0	0
42	0.2	0.3	0.1	0.5	0	0	0
43	0.2	0.3	0.1	0.5	0	0	0
44	0.2	0.3	0.1	0.5	0	0	0
45	0.2	0.3	0.1	0.5	0	0	0
46	0.2	0.3	0.1	0.5	0	0	0
47	0.2	0.3	0.1	0.5	0	0	0
48	0.2	0.3	0.1	0.5	0	0	0
49	0.2	0.3	0.1	0.5	0	0	0
50	0.2	0.3	0.1	0.5	0	0	0
51	0.2	0.3	0.1	0.5	0	0	0

w =

	1	2	3	4	5	6	7	8
51	0.2	0.3	0.1	0.5	0	0	0	0
52	0.2	0.3	0.1	0.5	0	0	0	0
53	0.2	0.3	0.1	0.5	0	0	0	0
54	0.2	0.3	0.1	0.5	0	0	0	0
55	0.2	0.3	0.1	0.5	0	0	0	0
56	0.2	0.3	0.1	0.5	0	0	0	0
57	0.2	0.3	0.1	0.5	0	0	0	0
58	0.2	0.3	0.1	0.5	0	0	0	0
59	0.2	0.3	0.1	0.5	0	0	0	0
60	0.2	0.3	0.1	0.5	0	0	0	0
61	0.2	0.3	0.1	0.5	0	0	0	0
62	0.2	0.3	0.1	0.5	0	0	0	0
63	0.2	0.3	0.1	0.5	0	0	0	0
64	0.2	0.3	0.1	0.5	0	0	0	0
65	0.2	0.3	0.1	0.5	0	0	0	0
66	0.2	0.3	0.1	0.5	0	0	0	0
67	0.2	0.3	0.1	0.5	0	0	0	0
68	0.2	0.3	0.1	0.5	0	0	0	0
69	0.2	0.3	0.1	0.5	0	0	0	0
70	0.2	0.3	0.1	0.5	0	0	0	0
71	0.2	0.3	0.1	0.5	0	0	0	0
72	0.2	0.3	0.1	0.5	0	0	0	0
73	0.2	0.3	0.1	0.5	0	0	0	0
74	0.2	0.3	0.1	0.5	0	0	0	0
75	0.2	0.3	0.1	0.5	0	0	0	0
76	0.2	0.3	0.1	0.5	0	0	0	0
77	0.2	0.3	0.1	0.5	0	0	0	0
78	0.2	0.3	0.1	0.5	0	0	0	0
79	0.2	0.3	0.1	0.5	0	0	0	0
80	0.2	0.3	0.1	0.5	0	0	0	0
81	0.2	0.3	0.1	0.5	0	0	0	0
82	0.2	0.3	0.1	0.5	0	0	0	0
83	0.2	0.3	0.1	0.5	0	0	0	0
84	0.2	0.3	0.1	0.5	0	0	0	0
85	0.2	0.3	0.1	0.5	0	0	0	0
86	0.2	0.3	0.1	0.5	0	0	0	0
87	0.2	0.3	0.1	0.5	0	0	0	0
88	0.2	0.3	0.1	0.5	0	0	0	0
89	0.2	0.3	0.1	0.5	0	0	0	0
90	0.2	0.3	0.1	0.5	0	0	0	0
91	0.2	0.3	0.1	0.5	0	0	0	0
92	0.2	0.3	0.1	0.5	0	0	0	0
93	0.2	0.3	0.1	0.5	0	0	0	0
94	0.2	0.3	0.1	0.5	0	0	0	0
95	0.2	0.3	0.1	0.5	0	0	0	0
96	0.2	0.3	0.1	0.5	0	0	0	0
97	0.2	0.3	0.1	0.5	0	0	0	0
98	0.2	0.3	0.1	0.5	0	0	0	0
99	0.2	0.3	0.1	0.5	0	0	0	0
100	0.2	0.3	0.1	0.5	0	0	0	0
101	0.2	0.3	0.1	0.5	0	0	0	0

Table 2 shows the values for the weights as generated from equations 4-51 to 4-58.

The following declares the activation input with the value 1 for the first layer of neurons:

$$x_{1,j} := 1 \qquad 4\text{-}60$$

where

$j := 1..4$ is the series of numbers for the number of first-layer neurons.

The next steps generate an input matrix for the defined real-world inputs for area, shape, red, green and blue:

$$x_{A,j} := area_A \qquad 4\text{-}61$$

$$x_{B,j} := shape_B \qquad 4\text{-}62$$

$$x_{C,j} := red_C \qquad 4\text{-}63$$

$$x_{D,j} := green_D \qquad 4\text{-}64$$

$$x_{E,j} := blue_E \qquad 4\text{-}65$$

These are fed into the formula of neural networks as in equation 2-22. The answers for the four nets are:

$$net_1 = 9.4$$

$$net_2 = 14.1$$

$$net_3 = 4.7$$

$$net_4 = 23.5$$

After this the sigmoid function of 2-23 is applied to the answers, resulting in the following:

$f_1 = 1$

$f_2 = 1$

$f_3 = 0.991$

$f_4 = 1$

These values are now the inputs for the following layer of four neurons with activation input as in equation 4-59. This is applied to the function of equation 2-22, where $j := 5..8$ indicates the number series of the second layer of neurons of which the answers are:

$net_5 = 3.494$

$net_6 = 1.497$

$net_7 = 1.996$

$net_8 = 0.499$

Once they have passed through the sigmoid function of 2-23 they become:

$f_5 = 0.971$

$f_6 = 0.817$

$f_7 = 0.88$

$f_8 = 0.622$

To prove the ability of the network to learn, it must be able to change the outputs of this final layer to produce user-required values.

Let us take these values to be in the form required by the backpropagation equation 2-24, such as the following:

$$t_5 := 0.1 \quad \text{4-66}$$

$$t_6 := 0.1 \quad \text{4-67}$$

$$t_7 := 0.1 \quad \text{4-68}$$

$$t_8 := 0.9 \quad \text{4-69}$$

As explained in section 2.2, when equation 2-24 is applied the values of error of the first layer result as follows:

$\delta_j =$

-0.025
-0.107
-0.082
0.065

This is fed into the formula for the first layer (equation 2-25) resulting in:

$\delta_j =$

$-6.28 \cdot 10^{-6}$
$-5.713 \cdot 10^{-8}$
$-6.782 \cdot 10^{-4}$
$-4.726 \cdot 10^{-12}$
-0.025
-0.107
-0.082
0.065

This includes the errors of the second layer. An argument learning rate of 1 is used:

$$\eta := 10 \quad \text{4-70}$$

By applying this to equation 2-26, the weight changes can be calculated.

These are then used with equation 2-27 to adapt all the weights as given in Table 3:

Table 3: The updated values of the weights for all the neurons

W =

	1	2	3	4	5	6	7
1	0.2	0.3	0.093	0.5	0.451	-0.771	-0.422
2	0.2	0.3	0.094	0.5	0.451	-0.771	-0.422
3	0.2	0.3	0.094	0.5	0.451	-0.771	-0.422
4	0.2	0.3	0.094	0.5	0.453	-0.762	-0.414
5	0.2	0.3	0.094	0.5	0.451	-0.771	-0.422
6	0.2	0.3	0.094	0.5	0	0	0
7	0.2	0.3	0.094	0.5	0	0	0
8	0.2	0.3	0.094	0.5	0	0	0
9	0.2	0.3	0.094	0.5	0	0	0
10	0.2	0.3	0.094	0.5	0	0	0
11	0.2	0.3	0.094	0.5	0	0	0
12	0.2	0.3	0.094	0.5	0	0	0
13	0.2	0.3	0.094	0.5	0	0	0
14	0.2	0.3	0.094	0.5	0	0	0
15	0.2	0.3	0.094	0.5	0	0	0
16	0.2	0.3	0.094	0.5	0	0	0
17	0.2	0.3	0.094	0.5	0	0	0
18	0.2	0.3	0.094	0.5	0	0	0
19	0.2	0.3	0.094	0.5	0	0	0
20	0.2	0.3	0.094	0.5	0	0	0
21	0.2	0.3	0.094	0.5	0	0	0
22	0.2	0.3	0.097	0.5	0	0	0
23	0.2	0.3	0.097	0.5	0	0	0
24	0.2	0.3	0.097	0.5	0	0	0
25	0.2	0.3	0.097	0.5	0	0	0
26	0.2	0.3	0.097	0.5	0	0	0
27	0.2	0.3	0.097	0.5	0	0	0
28	0.2	0.3	0.097	0.5	0	0	0
29	0.2	0.3	0.097	0.5	0	0	0
30	0.2	0.3	0.097	0.5	0	0	0
31	0.2	0.3	0.097	0.5	0	0	0
32	0.2	0.3	0.097	0.5	0	0	0
33	0.2	0.3	0.097	0.5	0	0	0
34	0.2	0.3	0.097	0.5	0	0	0
35	0.2	0.3	0.097	0.5	0	0	0
36	0.2	0.3	0.097	0.5	0	0	0
37	0.2	0.3	0.097	0.5	0	0	0
38	0.2	0.3	0.097	0.5	0	0	0
39	0.2	0.3	0.097	0.5	0	0	0
40	0.2	0.3	0.097	0.5	0	0	0
41	0.2	0.3	0.097	0.5	0	0	0
42	0.2	0.3	0.096	0.5	0	0	0
43	0.2	0.3	0.096	0.5	0	0	0
44	0.2	0.3	0.096	0.5	0	0	0
45	0.2	0.3	0.096	0.5	0	0	0
46	0.2	0.3	0.096	0.5	0	0	0
47	0.2	0.3	0.096	0.5	0	0	0
48	0.2	0.3	0.096	0.5	0	0	0
49	0.2	0.3	0.096	0.5	0	0	0
50	0.2	0.3	0.096	0.5	0	0	0
51	0.2	0.3	0.096	0.5	0	0	0

W =

	1	2	3	4	5	6	7	8
51	0.2	0.3	0.096	0.5	0	0	0	0
52	0.2	0.3	0.096	0.5	0	0	0	0
53	0.2	0.3	0.096	0.5	0	0	0	0
54	0.2	0.3	0.096	0.5	0	0	0	0
55	0.2	0.3	0.096	0.5	0	0	0	0
56	0.2	0.3	0.096	0.5	0	0	0	0
57	0.2	0.3	0.096	0.5	0	0	0	0
58	0.2	0.3	0.096	0.5	0	0	0	0
59	0.2	0.3	0.096	0.5	0	0	0	0
60	0.2	0.3	0.096	0.5	0	0	0	0
61	0.2	0.3	0.096	0.5	0	0	0	0
62	0.2	0.3	0.099	0.5	0	0	0	0
63	0.2	0.3	0.099	0.5	0	0	0	0
64	0.2	0.3	0.099	0.5	0	0	0	0
65	0.2	0.3	0.099	0.5	0	0	0	0
66	0.2	0.3	0.099	0.5	0	0	0	0
67	0.2	0.3	0.099	0.5	0	0	0	0
68	0.2	0.3	0.099	0.5	0	0	0	0
69	0.2	0.3	0.099	0.5	0	0	0	0
70	0.2	0.3	0.099	0.5	0	0	0	0
71	0.2	0.3	0.099	0.5	0	0	0	0
72	0.2	0.3	0.099	0.5	0	0	0	0
73	0.2	0.3	0.099	0.5	0	0	0	0
74	0.2	0.3	0.099	0.5	0	0	0	0
75	0.2	0.3	0.099	0.5	0	0	0	0
76	0.2	0.3	0.099	0.5	0	0	0	0
77	0.2	0.3	0.099	0.5	0	0	0	0
78	0.2	0.3	0.099	0.5	0	0	0	0
79	0.2	0.3	0.099	0.5	0	0	0	0
80	0.2	0.3	0.099	0.5	0	0	0	0
81	0.2	0.3	0.099	0.5	0	0	0	0
82	0.2	0.3	0.099	0.5	0	0	0	0
83	0.2	0.3	0.099	0.5	0	0	0	0
84	0.2	0.3	0.099	0.5	0	0	0	0
85	0.2	0.3	0.099	0.5	0	0	0	0
86	0.2	0.3	0.099	0.5	0	0	0	0
87	0.2	0.3	0.099	0.5	0	0	0	0
88	0.2	0.3	0.099	0.5	0	0	0	0
89	0.2	0.3	0.099	0.5	0	0	0	0
90	0.2	0.3	0.099	0.5	0	0	0	0
91	0.2	0.3	0.099	0.5	0	0	0	0
92	0.2	0.3	0.099	0.5	0	0	0	0
93	0.2	0.3	0.099	0.5	0	0	0	0
94	0.2	0.3	0.099	0.5	0	0	0	0
95	0.2	0.3	0.099	0.5	0	0	0	0
96	0.2	0.3	0.099	0.5	0	0	0	0
97	0.2	0.3	0.099	0.5	0	0	0	0
98	0.2	0.3	0.099	0.5	0	0	0	0
99	0.2	0.3	0.099	0.5	0	0	0	0
100	0.2	0.3	0.099	0.5	0	0	0	0
101	0.2	0.3	0.099	0.5	0	0	0	0

Compare the values in Table 3 with those in Table 2 and note how the weights change to produce outputs that are closer to the user-specified outputs. These outputs, after one training cycle, are the following:

$f_5 = 0.971$

$f_6 = 0.817$

$f_7 = 0.88$

$f_8 = 0.622$

Compare these to the original outputs. They are still not:

$t_5 := 0.1$

$t_6 := 0.1$

$t_7 := 0.1$

$t_8 := 0.9$

However, they are closer to the required output. Observe the outputs after five training cycles:

$f_5 = 0.09$

$f_6 = 0.027$

$f_7 = 0.102$

$f_8 = 0.97$

The margin of error is smaller, and the values are much closer to the required outputs of:

$t_5 := 0.1$

$t_6 := 0.1$

$t_7 := 0.1$

$t_8 := 0.9$

After twenty-five cycles the outputs are:

$f_5 = 0.1$

$f_6 = 0.1$

$f_7 = 0.1$

$f_8 = 0.9$

These represent the required outputs exactly.

The final weights (Table 4) were:

Table 4: The final values of the weights for all the neurons

W =

	1	2	3	4	5	6	7
1	0.2	0.3	0.085	0.5	-0.443	-0.442	-0.442
2	0.2	0.3	0.087	0.5	-0.443	-0.442	-0.442
3	0.2	0.3	0.087	0.5	-0.443	-0.442	-0.442
4	0.2	0.3	0.087	0.5	-0.431	-0.437	-0.435
5	0.2	0.3	0.087	0.5	-0.443	-0.442	-0.442
6	0.2	0.3	0.087	0.5	0	0	0
7	0.2	0.3	0.087	0.5	0	0	0
8	0.2	0.3	0.087	0.5	0	0	0
9	0.2	0.3	0.087	0.5	0	0	0
10	0.2	0.3	0.087	0.5	0	0	0
11	0.2	0.3	0.087	0.5	0	0	0
12	0.2	0.3	0.087	0.5	0	0	0
13	0.2	0.3	0.087	0.5	0	0	0
14	0.2	0.3	0.087	0.5	0	0	0
15	0.2	0.3	0.087	0.5	0	0	0
16	0.2	0.3	0.087	0.5	0	0	0
17	0.2	0.3	0.087	0.5	0	0	0
18	0.2	0.3	0.087	0.5	0	0	0
19	0.2	0.3	0.087	0.5	0	0	0
20	0.2	0.3	0.087	0.5	0	0	0
21	0.2	0.3	0.087	0.5	0	0	0
22	0.2	0.3	0.093	0.5	0	0	0
23	0.2	0.3	0.093	0.5	0	0	0
24	0.2	0.3	0.093	0.5	0	0	0
25	0.2	0.3	0.093	0.5	0	0	0
26	0.2	0.3	0.093	0.5	0	0	0
27	0.2	0.3	0.093	0.5	0	0	0
28	0.2	0.3	0.093	0.5	0	0	0
29	0.2	0.3	0.093	0.5	0	0	0
30	0.2	0.3	0.093	0.5	0	0	0
31	0.2	0.3	0.093	0.5	0	0	0
32	0.2	0.3	0.093	0.5	0	0	0
33	0.2	0.3	0.093	0.5	0	0	0
34	0.2	0.3	0.093	0.5	0	0	0
35	0.2	0.3	0.093	0.5	0	0	0
36	0.2	0.3	0.093	0.5	0	0	0
37	0.2	0.3	0.093	0.5	0	0	0
38	0.2	0.3	0.093	0.5	0	0	0
39	0.2	0.3	0.093	0.5	0	0	0
40	0.2	0.3	0.093	0.5	0	0	0
41	0.2	0.3	0.093	0.5	0	0	0
42	0.2	0.3	0.091	0.5	0	0	0
43	0.2	0.3	0.091	0.5	0	0	0
44	0.2	0.3	0.091	0.5	0	0	0
45	0.2	0.3	0.091	0.5	0	0	0
46	0.2	0.3	0.091	0.5	0	0	0
47	0.2	0.3	0.091	0.5	0	0	0
48	0.2	0.3	0.091	0.5	0	0	0
49	0.2	0.3	0.091	0.5	0	0	0
50	0.2	0.3	0.091	0.5	0	0	0
51	0.2	0.3	0.091	0.5	0	0	0

W =

	1	2	3	4	5	6	7	8
51	0.2	0.3	0.091	0.5	0	0	0	0
52	0.2	0.3	0.091	0.5	0	0	0	0
53	0.2	0.3	0.091	0.5	0	0	0	0
54	0.2	0.3	0.091	0.5	0	0	0	0
55	0.2	0.3	0.091	0.5	0	0	0	0
56	0.2	0.3	0.091	0.5	0	0	0	0
57	0.2	0.3	0.091	0.5	0	0	0	0
58	0.2	0.3	0.091	0.5	0	0	0	0
59	0.2	0.3	0.091	0.5	0	0	0	0
60	0.2	0.3	0.091	0.5	0	0	0	0
61	0.2	0.3	0.091	0.5	0	0	0	0
62	0.2	0.3	0.097	0.5	0	0	0	0
63	0.2	0.3	0.097	0.5	0	0	0	0
64	0.2	0.3	0.097	0.5	0	0	0	0
65	0.2	0.3	0.097	0.5	0	0	0	0
66	0.2	0.3	0.097	0.5	0	0	0	0
67	0.2	0.3	0.097	0.5	0	0	0	0
68	0.2	0.3	0.097	0.5	0	0	0	0
69	0.2	0.3	0.097	0.5	0	0	0	0
70	0.2	0.3	0.097	0.5	0	0	0	0
71	0.2	0.3	0.097	0.5	0	0	0	0
72	0.2	0.3	0.097	0.5	0	0	0	0
73	0.2	0.3	0.097	0.5	0	0	0	0
74	0.2	0.3	0.097	0.5	0	0	0	0
75	0.2	0.3	0.097	0.5	0	0	0	0
76	0.2	0.3	0.097	0.5	0	0	0	0
77	0.2	0.3	0.097	0.5	0	0	0	0
78	0.2	0.3	0.097	0.5	0	0	0	0
79	0.2	0.3	0.097	0.5	0	0	0	0
80	0.2	0.3	0.097	0.5	0	0	0	0
81	0.2	0.3	0.097	0.5	0	0	0	0
82	0.2	0.3	0.099	0.5	0	0	0	0
83	0.2	0.3	0.099	0.5	0	0	0	0
84	0.2	0.3	0.099	0.5	0	0	0	0
85	0.2	0.3	0.099	0.5	0	0	0	0
86	0.2	0.3	0.099	0.5	0	0	0	0
87	0.2	0.3	0.099	0.5	0	0	0	0
88	0.2	0.3	0.099	0.5	0	0	0	0
89	0.2	0.3	0.099	0.5	0	0	0	0
90	0.2	0.3	0.099	0.5	0	0	0	0
91	0.2	0.3	0.099	0.5	0	0	0	0
92	0.2	0.3	0.099	0.5	0	0	0	0
93	0.2	0.3	0.099	0.5	0	0	0	0
94	0.2	0.3	0.099	0.5	0	0	0	0
95	0.2	0.3	0.099	0.5	0	0	0	0
96	0.2	0.3	0.099	0.5	0	0	0	0
97	0.2	0.3	0.099	0.5	0	0	0	0
98	0.2	0.3	0.099	0.5	0	0	0	0
99	0.2	0.3	0.099	0.5	0	0	0	0
100	0.2	0.3	0.099	0.5	0	0	0	0
101	0.2	0.3	0.099	0.5	0	0	0	0

The adapted weights given in Table 4 are able to reproduce the user-defined outputs for the simulated inputs. This network thus has the ability to learn.

5 LABVIEW IMPLEMENTATION

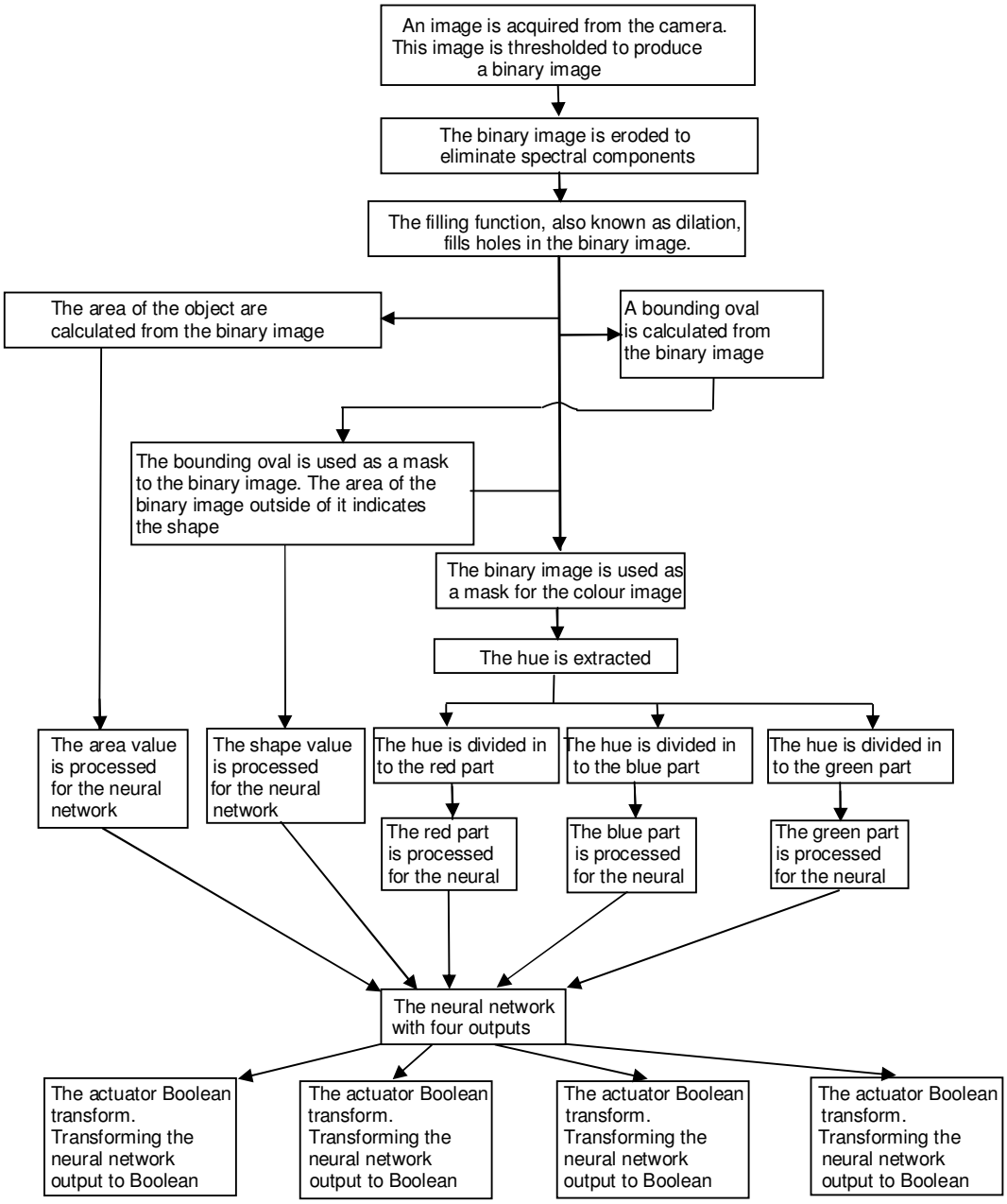
LabVIEW is a third-generation compiler. It is a graphical programming language with high-level tools and configuration utilities designed specifically for test, measurement and control applications. In lower-level programming languages, the code is as much of a concern as the application. The syntax such as commas, semi-colons, and brackets are important and must be correct for such programming languages to operate. In contrast, in LabVIEW graphical icons are wired together to represent functions that determine the flow of data through the program, similar to creating flowcharts.

LabVIEW has the breadth and depth of a general-purpose programming language, but is more intuitive and easier to use. This makes users more productive by decreasing the time required to develop software [23].

A building block of a LabVIEW program is called a VI, which consists of two windows. One of these windows is called the front panel and the other the block diagram. Programming in this way is the same as building a hardware application. The front panel represents the outside of the box with all the lights, knobs and displays, and the block diagram represents the inside of the box with its wiring, code and functions.

The block diagram is where users develop their graphical code. There LabVIEW provides all the functionality of a typical general-purpose programming language including data types, looping structures and event handlers. The advantage of using LabVIEW is that it includes hundreds of functions specific to engineering

tasks while it reduces the complexity of programming through interactive tools and by abstracting away many programming details. For example, LabVIEW performs all necessary memory management automatically, ensuring that it is used efficiently and safely. This frees engineers and scientists to focus on their work without having to worry about the details of memory allocation, which is fraught with potential errors [23]. To realize the system must the following, as explained thus far, be coded in LabVIEW:



5.1 Feature extraction

First, the implementation software is coded without the real-time camera, but with the aid of pre-snapped bitmap images. These images are similar to those used in the simulations in section 4. The camera block is then placed where the bitmap read block was. Since the software is triggered it delivers one picture at a time. This is then the same operation as reading a single bitmap file at a time.

As stated earlier, the aim of feature extraction is to extract useful feature information which includes size, colour and irregular shapes of an object on a conveyer belt for quality control purposes.

The fist step is to read in the file from a previously acquired bitmap. The reason for this is twofold. First is the development and debugging of the software. The same images are read in and the lighting and external interferences are kept the same. Second, these are the same images that were used in section 4 of the mathematical simulations, which simplifies the evaluation of the software. After the software is coded and evaluated, the read block is replaced with an “acquire image from camera” block which will turn this application into a real-time program.

The graphical code for reading in a bitmap as coded in the block diagram part of LabVIEW is shown in Figure 5-1.

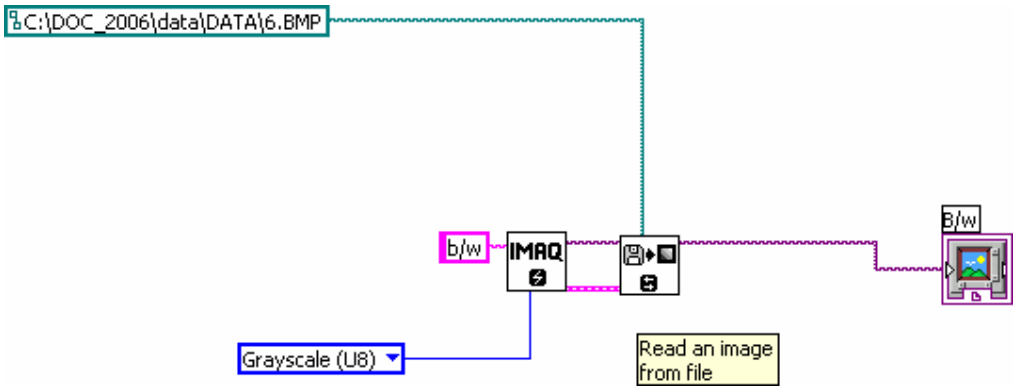


Figure 5-1: Code of the read function

This figure shows the combination of the programming blocks to display the black and white image required for further processing as explained in section 2.1.2 and simulated in section 4.

IMAQ Create is the first block as shown in Figure 5-1.. It generates the basis of an image.

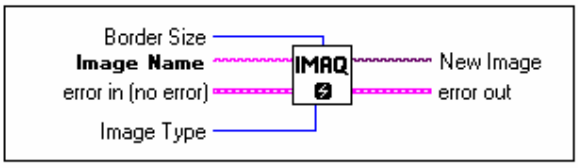


Figure 5-2: IMAQ Create

In Figure 5-2:

- **Border Size** determines the width, in pixels, of the border to be created around an image.
- **Image Name** is the name associated with the created image. Each image created must have a unique name.
- **Error in (no error)** describes error conditions that occur before this VI or function runs. The default is no error.
- **Image Type** specifies the image type.
- **New Image** is the **Image** reference that is supplied as input to all subsequent (downstream) functions used by NI Vision. Multiple images can be created in a LabVIEW application.
- **Error out** contains error information.

IMAQ Create is connected to the IMAQ ReadFile which reads an image file. The file format can be standard (BMP, TIFF, JPEG, JPEG2000, PNG and AIPD) or non-standard known to the user. In all cases, the read pixels are converted automatically into the image type passed by IMAQ Create.

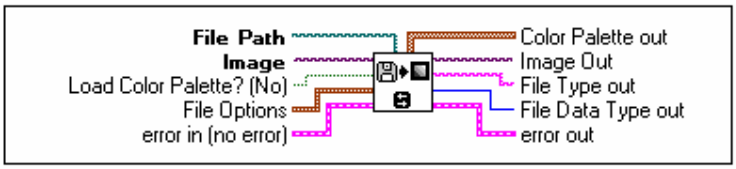


Figure 5-3: IMAQ ReadFile code block

In Figure 5-3:

- **File Path** is the complete pathname - including drive, directory, and filename - of the file to be read.
- **Image** is a reference to the image to which data from the image file is applied.
- **Load Colour Palette? (No)** determines whether to load the colour table present in the file (if the file has one).
- **File Options** is a cluster of user-optional values that can be used to read non-standard file formats.
- **Error in (no error)** describes error conditions that occur before this VI or function runs.
- **Colour Palette out** contains the RGB colour table (if the file has one).
- **Image Out** refers to the destination image.
- **File Type out** indicates the file type that is read.
- **File Data Type out** indicates the pixel size defined in the header for standard image file types.
- **Error out** contains error information.

As explained, this is not used at first. The result of reading in the bitmap and converting it to greyscale is displayed with the block labelled B/w which is connected to the front panel.

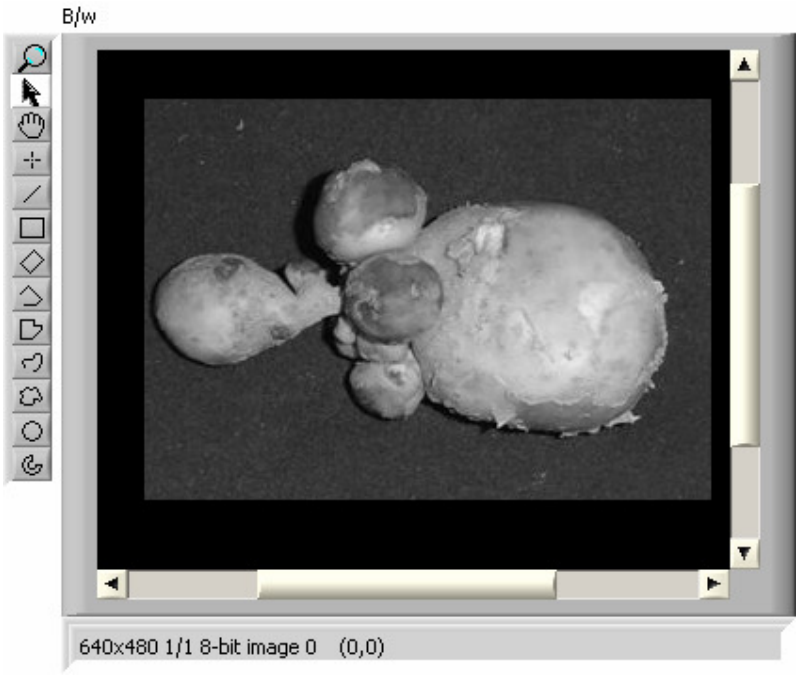


Figure 5-4: Front panel

Figure 5-4 shows the view the user would have of the image on the front panel.

5.1.1 Binary image (Thresholding)

As explained in section 2.1.2.1 and simulated in section 4.1.1, a binary image consists of 1's and 0's and represents a greyscale image that was put through a thresholding algorithm [14 p.13] [12].

As was done in the simulation, the first step is to import a greyscale image. The sample picture is in RGB format and must first be converted to a greyscale image. Since LabVIEW is a third-generation programming language, implementing this is as easy as placing the correct code block: the read-in image is now connected to the input of the IMAQ Threshold block which, as in the simulation, applies a threshold to an image.

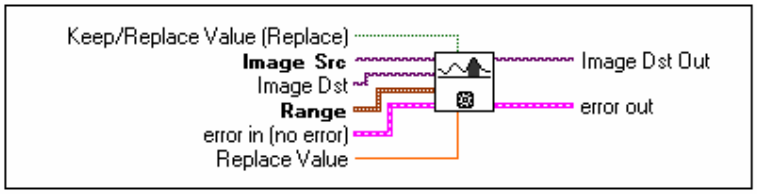


Figure 5-5: IMAQ Threshold

In Figure 5-5:

- **Keep/Replace Value (Replace)** determines whether to replace the value of the pixels existing in the range between **Lower value** and **Upper value**.
- **Image Src** is a reference to the source image.
- **Image Dst** is a reference to the destination image. If **Image Dst** is connected, it must be the same type as the **Image Src**.
- **Range** is a cluster specifying the threshold range. In this case it will be user defined.
- **Error in (no error)** describes error conditions that occur before this VI or function runs.
- **Replace Value** is the value used to replace pixels between the **Lower value** and **Upper value**. As in the simulation the default is 1.
- **Image Dst Out** refers to the destination image.
- **Error out** contains error information.

The result of the coding is as follows:

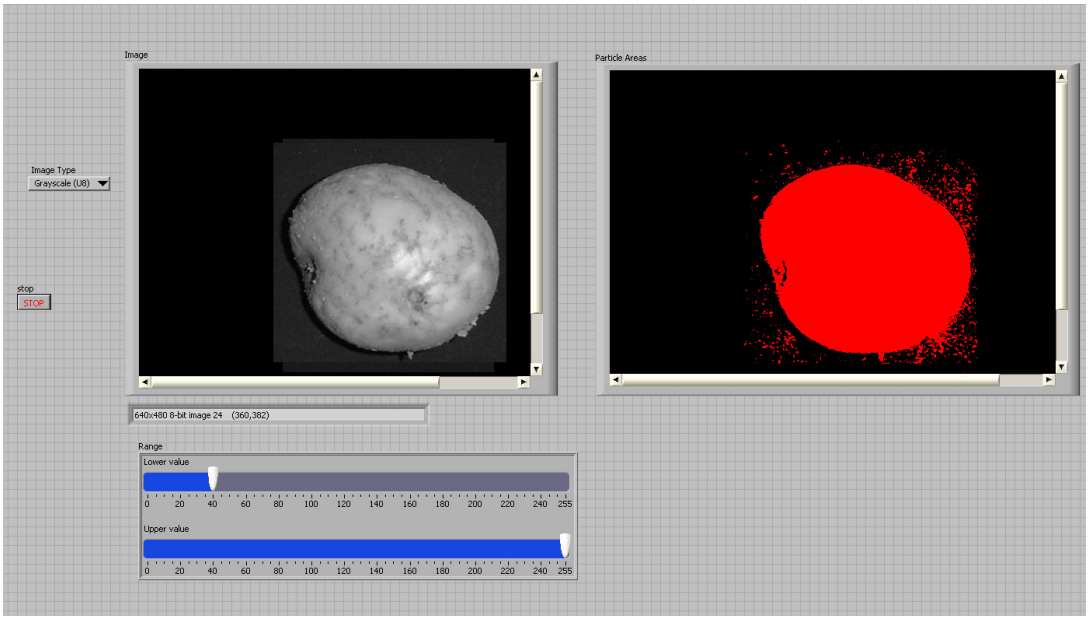


Figure 5-6: The view of user interface to the implementation code

In Figure 5-6 the two images represent the read-in greyscale image (on the left) and the threshold image (on the right). The control bars at the bottom are the manual settings of the threshold lower and the upper level. In this example it is set to the same values as those of the simulation. Note the good correlation between the simulation image in Figure 4-3 and the result of the implemented image in Figure 5-6. The simulation was therefore successfully implemented.

5.1.2 Erode

The next step is the erode function as explained in section 2.1.2.2 and simulated in section 4.1.2. The binary image is now connected to the erode code block (Figure 5-7).

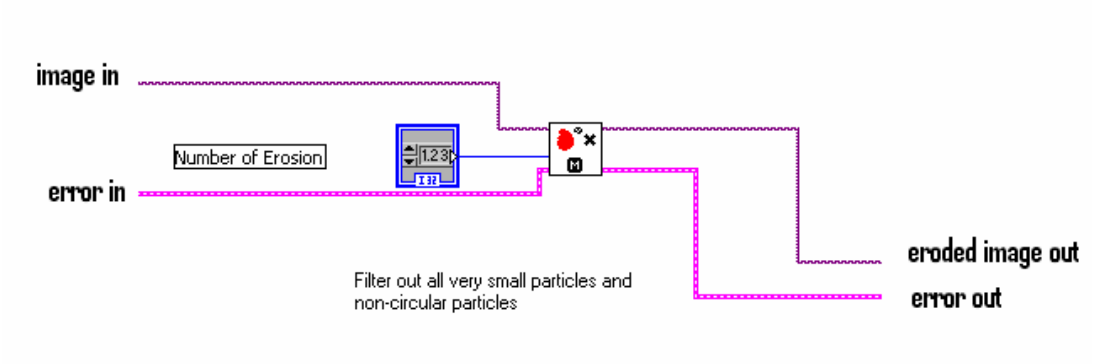


Figure 5-7: The erode function block is added and connected to the threshold function block

In Figure 5-7 the output of the threshold is now connected to the erode block. In LabVIEW it is labelled IMAQ RemoveParticle. It eliminates or keeps particles resistant to a specified number of 3 x 3 erosions. The particles that are kept are exactly the same shape as those in the original source image.

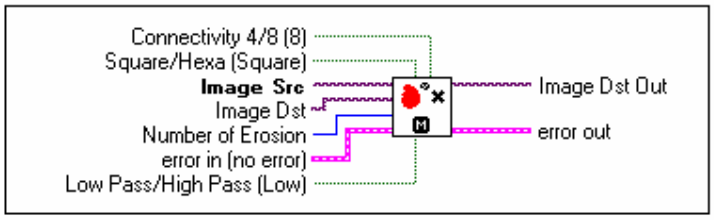


Figure 5-8: Block diagram of the IMAQ RemoveParticle

In Figure 5-8:

- **Connectivity 4/8 (8)** specifies the type of connectivity used by the algorithm for particle detection.
- **Square/Hexa (Square)** specifies whether to treat the pixel frame as square or hexagonal during the transformation. The default is square.
- **Image Src** refers to the source image.
- **Image Dst** refers to the destination image.
- **Number of Erosion** specifies the number of 3 x 3 erosions to apply to the image. The default is 2.
- **Error in (no error)** describes error conditions that occur before this VI or function runs.
- **Low Pass/High Pass (Low)** specifies whether the objects resistant to n erosions are discarded or kept (default).
- **Image Dst Out** refers to the destination image.
- **Error out** contains error information.

This is applied as in the simulation in section 4.1.2 and is as follows:

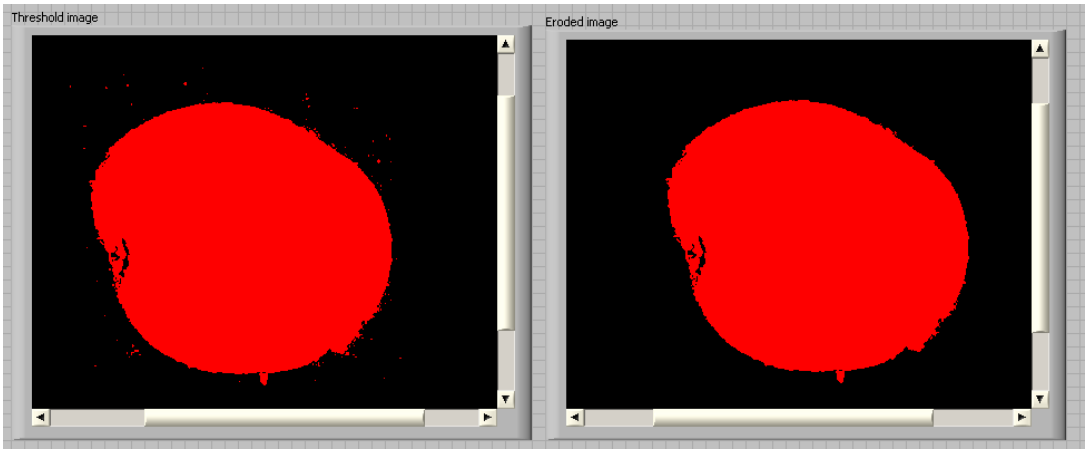


Figure 5-9: The user interface to the implementation code showing the result of the erode function

Figure 5-9 shows the result of the erode function. Note how the small spectral dots are removed when one compares the threshold image with the erode image. Note the good visual correlation between this and the simulation in Figure 4-4.

5.1.3 Fill

The next part has the fill function as explained in section 2.1.2.3 and simulated in section 4.1.3. The “image in” of the fill block is connected to the “image out” of the erode block as shown in Figure 5-10.

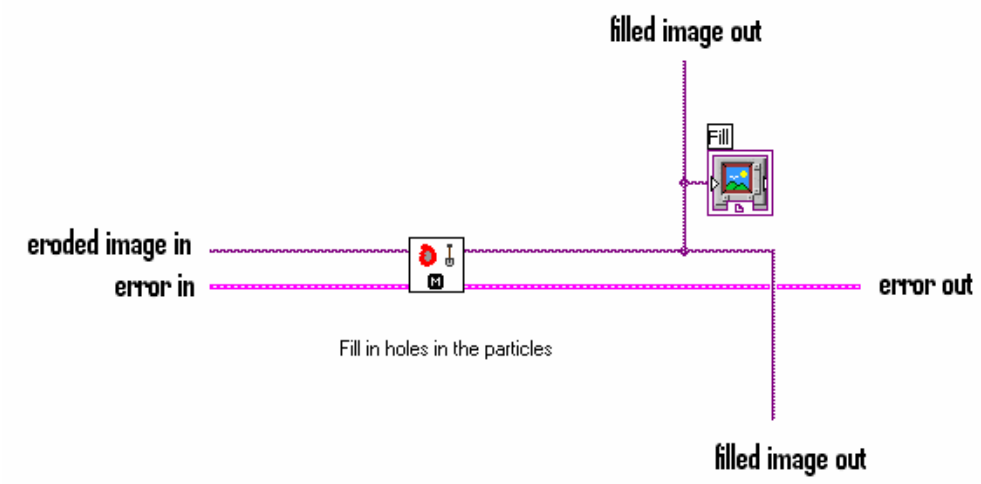


Figure 5-10: The fill function block is added and connected to the erode function block

In Figure 5-10 the IMAQ FillHole fills the holes found in a particle. The holes are filled with a pixel value of 1 as explained in section 2.1.2.3 and simulated in section 4.1.3.

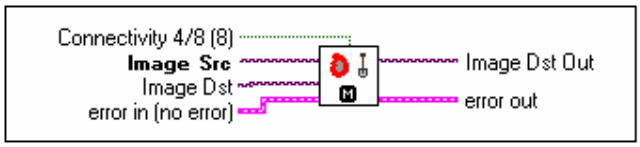


Figure 5-11: Block diagram of the IMAQ FillHole

In Figure 5-11:

- **Connectivity 4/8 (8)** specifies the type of connectivity used by the algorithm for particle detection.
- **Image Src** refers to the source image.
- **Image Dst** refers to the destination image.
- **Error in (no error)** describes error conditions that occur before this VI or function runs.
- **Image Dst Out** refers to the destination image.
- **Error out** contains error information.

The result as seen on the front panel is shown in Figure 5-12.

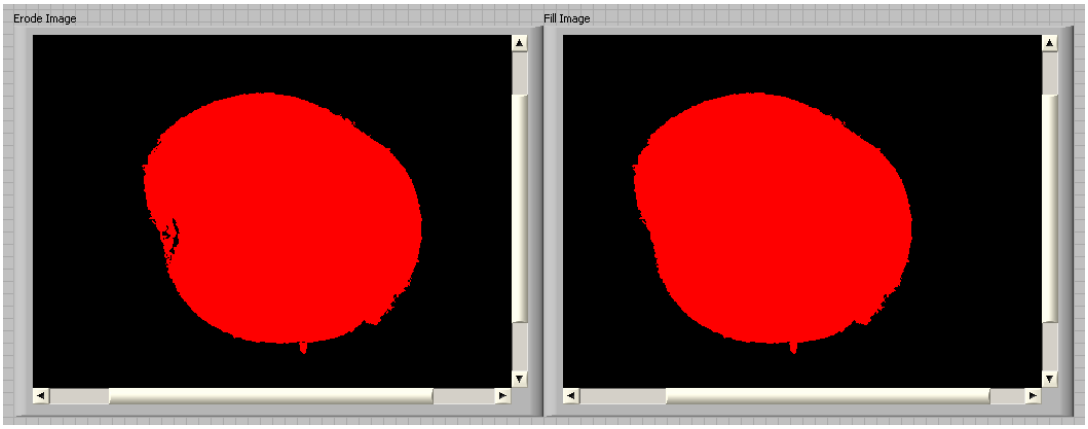


Figure 5-12: A section of the user interface of the implementation code showing the result of the fill function

Figure 5-12 shows the result of the fill function. Note how the holes in the object are filled if one compares the erode image with the fill image. The good visual correlation between this and Figure 4-5 in the simulation in section 4.1.3 indicates successful implementation.

5.1.4 Area

As explained in section 4.1.4, the area of the object can be calculated from the processed binary image. The binary image consists of 1's where the object occupies the image and 0's for the background. Therefore the area can be calculated by adding all the values of the matrix together as follows: The "image out" of the fill block is connected to the "image in" of the IMAQ Particle Analysis as in Figure 5-13.

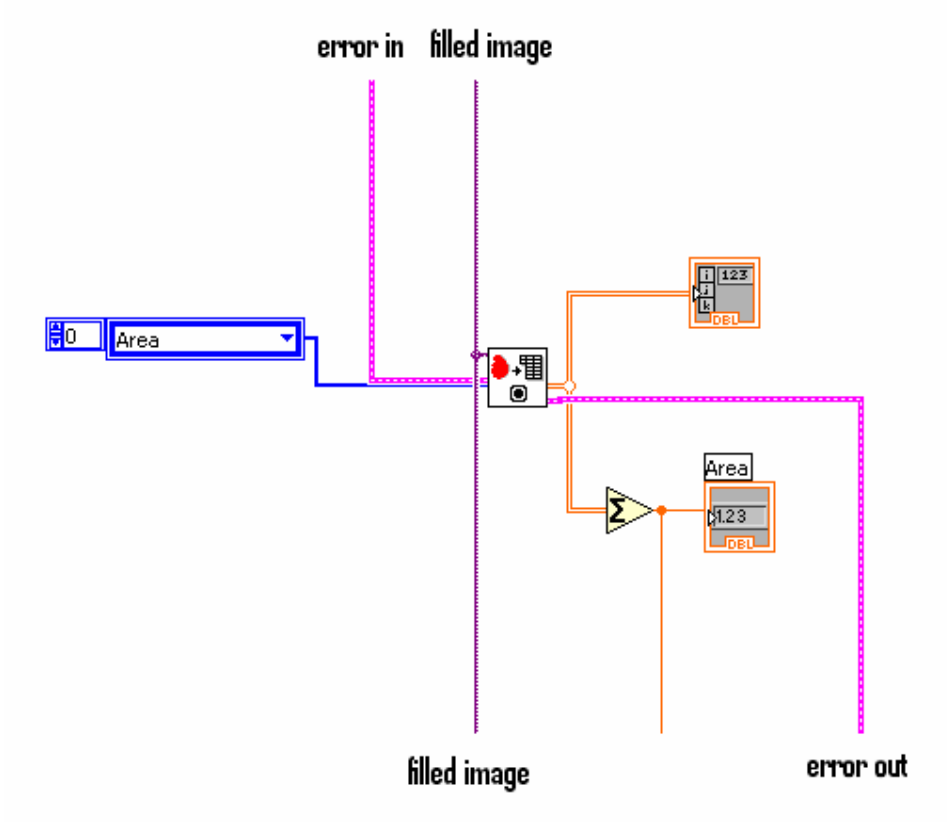


Figure 5-13: Connecting the IMAQ Particle Analysis to a sigma function to add up all the 1's in the matrix

The IMAQ Particle Analysis returns the number of particles detected in a binary image and a 2D array of requested measurements of the particle.

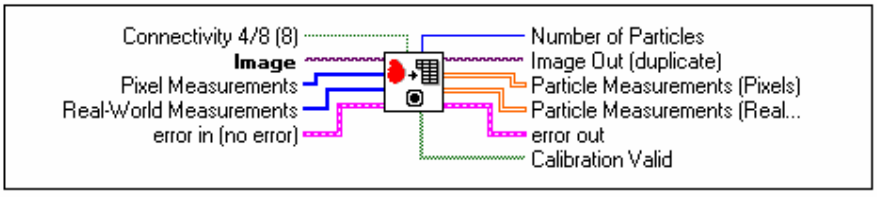


Figure 5-14: The IMAQ Particle Analysis Block

In Figure 5-14:

- **Connectivity 4/8 (8)** specifies the type of connectivity used by the algorithm for particle detection. The connectivity mode directly determines

whether an adjacent pixel belongs to the same particle or a different particle. The default is 8. The following values are possible: 8 (TRUE): particle detection is performed in connectivity mode 8. 4 (FALSE): Particle detection is performed in connectivity mode 4.

- **Image** refers to the source image.
- **Pixel Measurements** is an array of measurement parameters that can be requested for each particle. The parameters are returned as uncalibrated pixel measurements.
- **Real-World Measurements** is an array of measurement parameters that can be requested for each particle. The parameters are returned as calibrated real-world measurements. If Image does not have any attached calibration information, the VI returns pixel measurements.
- **Error in (no error)** describes error conditions that occur before this VI or function runs.
- **Number of Particles** indicates the number of particles detected in the image.
- **Image Out (duplicate)** is a reference to Image In.
- **Particle Measurements (Pixels)** is a 2D array that returns the requested pixel measurements from the detected particles. The array has one column for each measurement requested in Pixel Measurements and one row for each particle detected.
- **Particle Measurements (Real-World)** is a 2D array that returns the requested calibrated real-world measurements from the detected particles.

The array has one column for each measurement requested in Real-World Measurements and one row for each particle detected. If Image does not have any attached calibration information, the VI returns pixel measurements.

- **Error out** contains error information.
- **Calibration Valid** indicates whether the calibration information for a particle is valid. Calibration Valid has one Boolean for each particle (row) in Particle Measurements (Real-World). If the calibration information is invalid for any pixel in the particle, the corresponding Calibration Valid Boolean is FALSE.

This value can now be divided by a user-defined constant and multiplied by 100 to generate a percentage of area. This would be ideally set so that the objects to be sorted are in the 50% range.

5.1.5 Shape

As explained in section 2.1.2.5.3 and simulated in section 4.1.5, the scale of the irregular shape is calculated with the aid of a bounding oval and the area of the binary image outside of it. The first step is to calculate the bounding oval coordinates. This is done with the IMAQ Particle Analysis block.

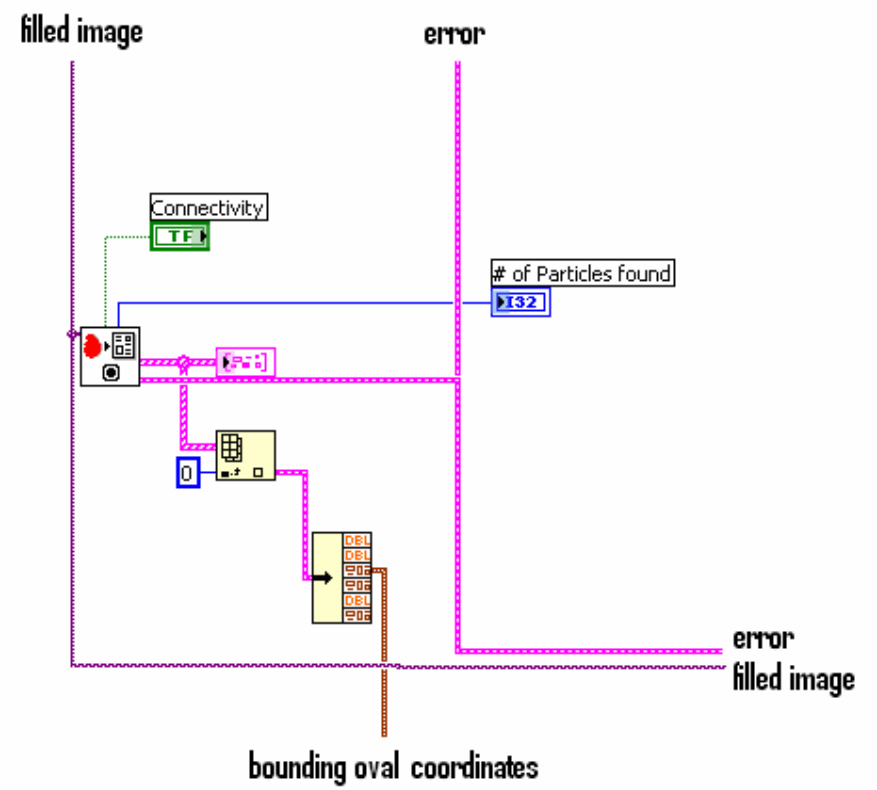


Figure 5-15: The connection of the IMAQ Particle Analysis Report to generate the bounding oval coordinates

The IMAQ Particle Analysis returns the number of particles detected in a binary image and an array of reports containing the most commonly used particle measurements mathematically as described in section 4.1.5.

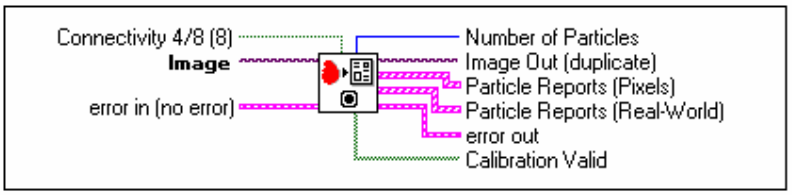


Figure 5-16: IMAQ Particle Analysis Report block

In Figure 5-16:

- **Connectivity 4/8 (8**) specifies the type of connectivity used by the algorithm for particle detection. The connectivity mode directly determines whether an adjacent pixel belongs to the same particle or a different particle. The default is 8. The following values are possible: 8 (TRUE): particle detection is performed in connectivity mode 8. 4 (FALSE): particle detection is performed in connectivity mode 4.
- **Image** refers to the source image.
- **Error in** (no error) describes error conditions that occur before this VI or function runs.
- **Number of Particles** indicates the number of particles detected in the image.
- **Image Out (duplicate)** refers to Image In.
- **Particle Reports (Pixels)** is an array that returns a set of uncalibrated pixel measurements from the detected particles. This cluster contains the following elements:
    - Area is the area of the particle.
    - Number of Holes is the number of holes in the particle. NI Vision can detect holes inside a particle as small as one pixel.
    - Bounding Rect is the smallest rectangle with sides parallel to the x axis and y axis that completely encloses the particle. This is then the part that is used further.

- Centre of Mass is the point representing the average position of the total mass of the particle assuming every point in the particle has a constant density. Centre of Mass may be located outside the particle if the particle is not convex.
- Orientation is the angle of the line passing through the particle centre of mass with the lowest moment of inertia.
- Dimensions indicate the width and height of Bounding Rect.

- **Particle Reports (Real-World)** is an array that returns a set of calibrated real-world measurements from the detected particles as above.
- **Error out** contains error information.
- **Calibration Valid** indicates whether the calibration information for a particle is valid. Calibration Valid has one Boolean for each report in Particle Reports (Real-World). If the calibration information is invalid for any pixel in the particle, the corresponding Calibration Valid Boolean is FALSE.

The particle report returns an array of clusters of each object containing a set of pixel measurements from the detected particles. At present the only data needed are the coordinates of the bounding oval. The relevant cluster is extracted from all the others using the index array block.

The Index Array returns the element or subarray of n-dimension array at index.

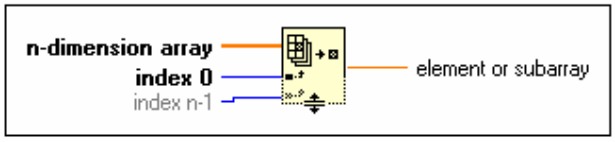


Figure 5-17: Index Array block extracting the relevant object cluster

In Figure 5-17:

- The **n-dimension array** can be of any type. If n-dimension array is an empty array, element or subarray returns the default value of the defined data type for the array.
- **index 0…**n-1 must be numeric. The number of index inputs matches the number of dimensions in n-dimension array.

If the index is out of range (<0 or N, where N is the size of n-dimension array), element or subarray returns the default value of the defined data type for the array.

The cluster containing the Particle Report for the object must now be unbundled to extract only the bonding coordinates.

The unbundle block splits a **cluster** into each of its individual elements.

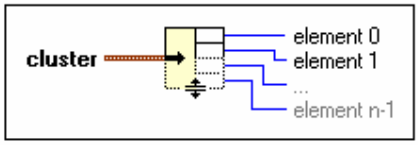


Figure 5-18: Unbundle cluster block

In Figure 5-18:

- **Cluster** is the cluster whose elements one wishes to access.
- **Element 0**...n-1 are the elements of cluster.

The next part of the programming shows how the necessary coordinates are unbundled and re-bundled in the required report form to generate a mask into a mask block using the region of interest.

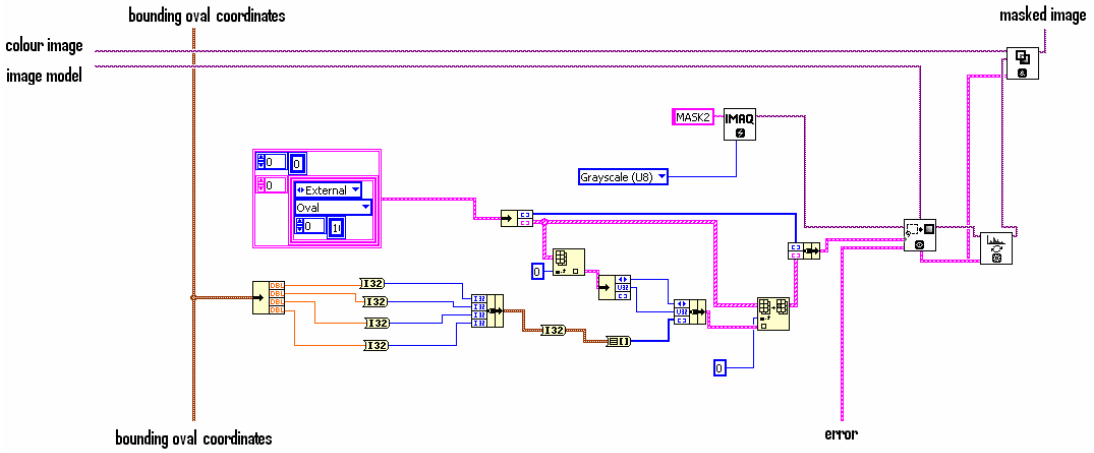


Figure 5-19: Unbundle the necessary information to generate Region of Interest (ROI) of the bounding oval for masking purposes

The IMAQ ROIToMask transforms a region of interest into a mask.

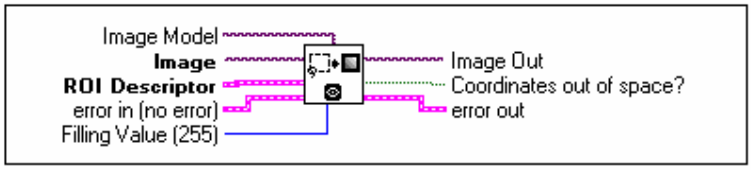


Figure 5-20: The IMAQ ROIToMask block

In Figure 15-20:

- **Image Model** serves as a template for the destination image where the mask is placed. This can be any image type that NI Vision supports. Image takes the characteristics of Image Model (size and location of ROI) when Image Model is connected. However, the connection of Image Model is optional. When no Image Model is connected, the size of the image mask generated is limited to the bounding rectangle of the ROI Descriptor, which reduces the amount of memory used. The offset of the image mask generated is set to reflect the real position of the ROI.
- **Image** refers to the destination image in which the mask is created.
- **ROI Descriptor** is the descriptor that defines the region of interest.
    - Global Rectangle is the minimum rectangle required to contain all the contours in the ROI.
    - Contours are each of the individual shapes that define the ROI, as follows: ID refers to whether the contour is the external or internal edge of an ROI. If the contour is external, the whole area inside it is considered part of the ROI. This VI draws the contours in the order that they are listed in the ROI Descriptor. One can use internal contours to create non-mask regions inside external contours.
    - Type is the shape type of the contour.
    - Coordinates are the coordinates that define the contour.

- **Error in** (no error) describes error conditions that occur before this VI or function runs.
- **Filling Value** (255) is the pixel value of the mask. All pixels inside the region of interest take this value. The default value is 255.
- **Image Out** refers to the destination image. If Image Dst is connected, Image Dst Out is the same as Image Dst. Otherwise Image Dst Out refers to the image referenced by Image Src.
- **Coordinates out of space?** returns TRUE if any ROI data are found outside the space associated with the Image Model.
- **Error out** contains error information.

A mask is now generated as a bounding oval, but this bounding oval must be inverted and removed so that only the parts of the object lying outside it are displayed for further calculations.

Inverting is done with IMAQ Inverse block as it inverts the pixel intensities of an image to compute the negative of an image.

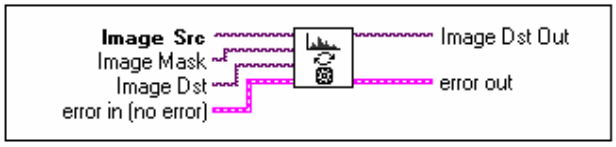


Figure 5-21: The IMAQ Inverse block

In Figure 5-21:

- **Image Src** refers to the source image.

- **Image Mask** is an 8-bit image that specifies the region in the image to modify. Only those pixels in the original image that correspond to an equivalent non-zero pixel in the mask image are processed. All other pixels keep their original values. The entire image is processed if Image Mask is not connected.
- **Image Dst** refers to the destination image. If Image Dst is connected, it must be the same type as Image Src.
- **Error in** (no error) describes error conditions that occur before this VI or function runs.
- **Image Dst Out** refers to the destination image. If Image Dst is connected, Image Dst Out is the same as Image Dst. Otherwise Image Dst Out refers to the image referenced by Image Src.
- **Error out** contains error information.

This inverted mask is now used as a mask for the thresholded image with IMAQ Mask

IMAQ Mask recopies Image Src into Image Dst. If a pixel value is 0 in the Image Mask, the corresponding pixel in Image Dst is set to 0.

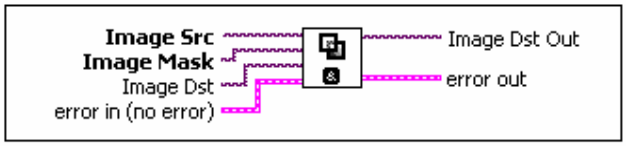


Figure 5-22: The IMAQ Mask block

In Figure 5-22:

- **Image Src** refers to the source image.
- **Image Mask** is an 8-bit image that specifies the region in the image to modify. The pixels in the original image whose equivalent pixels in the Image Mask have a value of 0 are set to 0. All other pixels keep their original value. For more information about image masks, refer to the Region of Interest VIs.
- **Image Dst** refers to the destination image. If Image Dst is connected, it must be the same type as Image Src.
- **Error in** (no error) describes error conditions that occur before this VI or function runs.
- **Image Dst** Out refers to the destination image. If Image Dst is connected, Image Dst Out is the same as Image Dst. Otherwise Image Dst Out refers to the image referenced by Image Src.
- **Error out** contains error information.

As with the previous area calculation, the area is calculated again, but only on the masked image, thus only the particles outside the bounding oval.

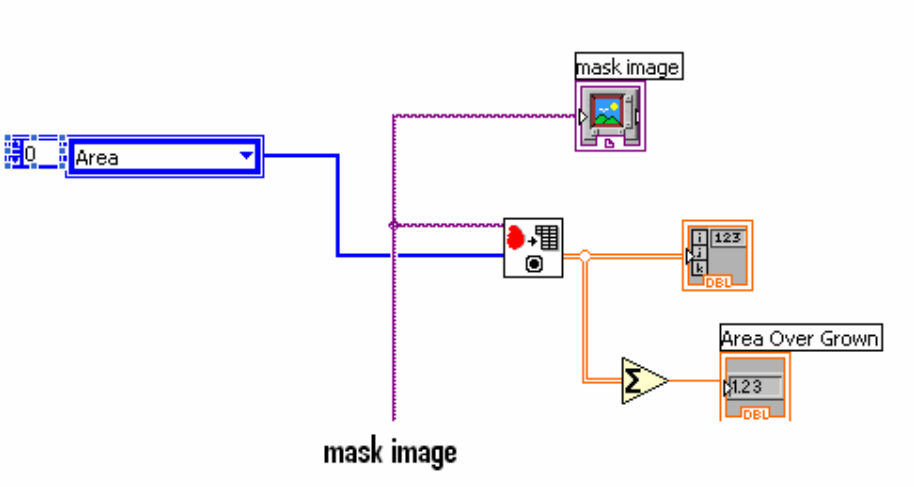


Figure 5-23: The code for calculating the area of the overgrown part

Figure 5-23 shows the code that calculates the area of the overgrown part by calculating the area outside the bounding oval. The result of the masking is as shown in Figure 5-24:

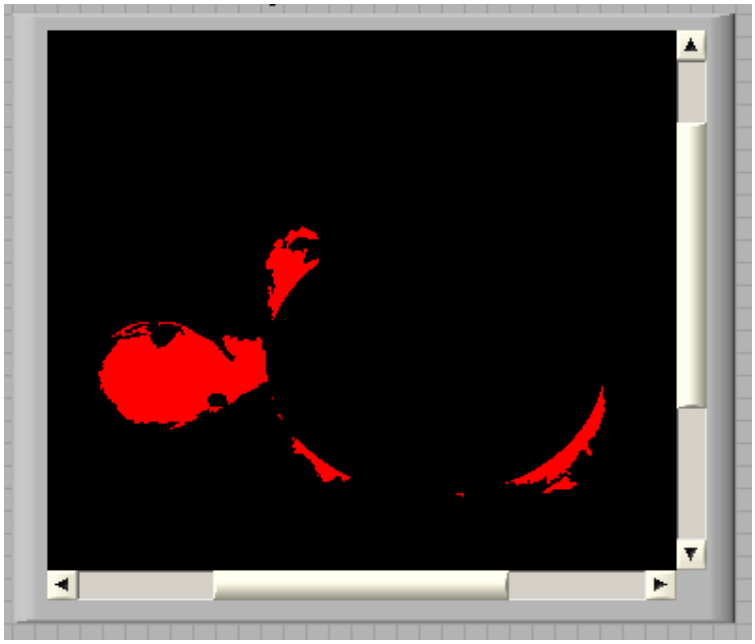


Figure 5-24: A section of the user interface of the implementation code showing the result of the masking function

Figure 5-24 shows the result of the masking function. Note how only the parts lying outside the bounding oval are displayed. The good visual correlation between this and Figure 4-17 in the simulation in section 4.1.5.4 indicates successful implementation.

5.1.6 Colour analyses

As previously stated and simulated, the colour of the potato was analysed with respect to its area of hue [38, p. 423] [28, p. 1591] [9, p. 12].

As stated in section 2.1.1.2 and simulated in section 4.1.6, hue is a colour processing technique where all the possible colours are plotted around a circle at certain radians. This implies that a certain colour is represented by a corresponding angle. Section 4.1.6 shows a simulation of how the decision-making process for this colour representation is broken up into areas of green (from yellowish-green to greenish-blue), blue (from blue-green to violet) and red (from violet to reddish-yellow). This is now simulated in LabVIEW with the same colour wheel to evaluate the implementation application. As with the simulation, colour processing is done on the colour image that was masked in the previous extraction steps.

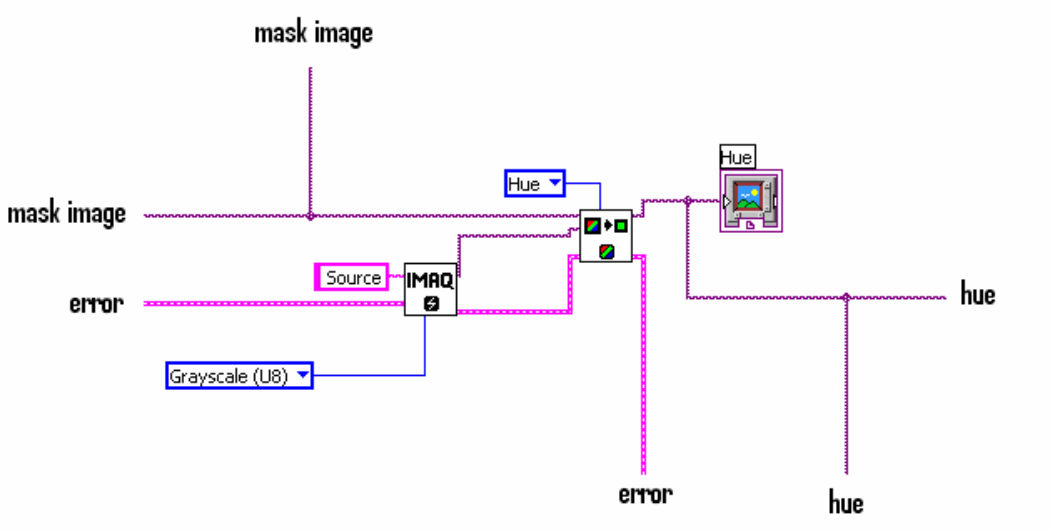


Figure 5-25: The code extracting the hue of the object

Figure 5-25 shows the code extracting the hue as simulated in section 4.1.6. The angle is converted so that an amplitude of 0 is 0 degrees and 255 is equal to 360 degrees. The reason for the range of 0 to 255 is more for illustration purposes than mathematical sense. The angle can be displayed in an image frame since 0 represents black and 255 represents white and the values in between would be all the shades of grey. Of course 255 can also be processed with 8 bits.

In Figure 5-25 IMAQ Create is the first block. It generates the basis of an image.

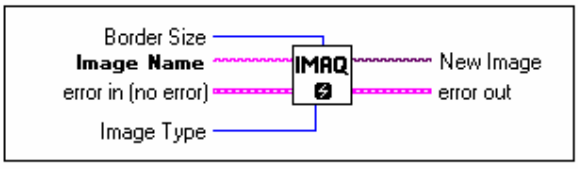


Figure 5-26: IMAQ Create

In Figure 5-26:**Border Size** determines the width, in pixels, of the border to create around an image.

- **Image Name** is the name associated with the created image. Each image created must have a unique name.
- **Error in (no error)** describes error conditions that occur before this VI or function runs. The default is no error.
- **Image Type** specifies the image type.
- **New Image** is the **Image** reference that is supplied as input to all subsequent (downstream) functions used by NI Vision. Multiple images can be created in a LabVIEW application.
- **Error out** contains error information.

This new source image is then filled with the masked colour image and the hue is extracted as shown in Figure 5-27:

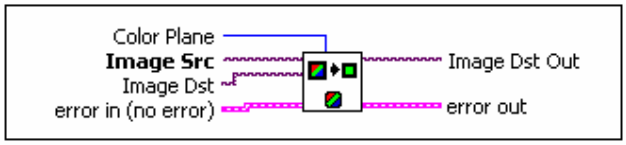


Figure 5-27: IMAQ ExtractSingleColourPlane

In Figure 5-27:

- **Colour Plane** defines the colour plane to extract. One can choose from the following values:
  - Red (default)
  - Green, Blue
  - Hue
  - Saturation

- Luminance
- Value
- Intensity.

- **Image Src** refers to a colour image that has one of its colour planes extracted. If Image Dst is not connected, the source image is converted to an image that contains the extracted plane.
- **Image Dst** refers to the destination image. If Image Dst is connected, it must have as many bits per pixel as the extracted colour plane.
- **Error in** (no error) describes error conditions that occur before this VI or function runs.
- **Image Dst Out** refers to the destination image. If Image Dst is connected, Image Dst Out is the same as Image Dst. Otherwise Image Dst Out refers to the image referenced by Image Src.
- **Error out** contains error information.

The result is shown in Figure 5-28.

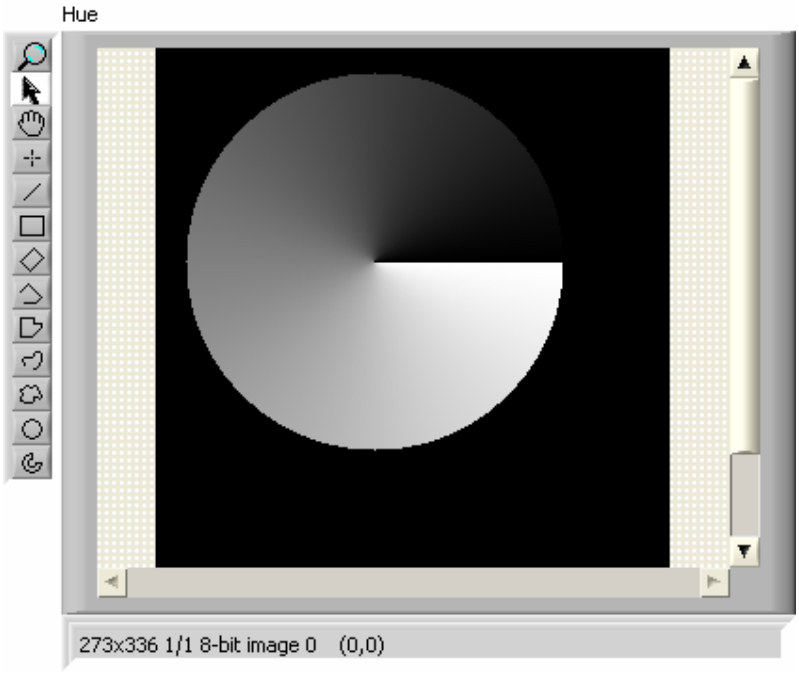


Figure 5-28: The angle of colour (hue) of the sample colour wheel

The good correlation between Figure 5-28 with the simulation Figure 4-19 shows a successful implementation thus far. The next step is to divide the colours into three subgroups for simplification of the decision-making part.

As in the simulation, the first area is green (from yellowish-green to greenish-blue), and the threshold function code is as given in Figure 5-29.

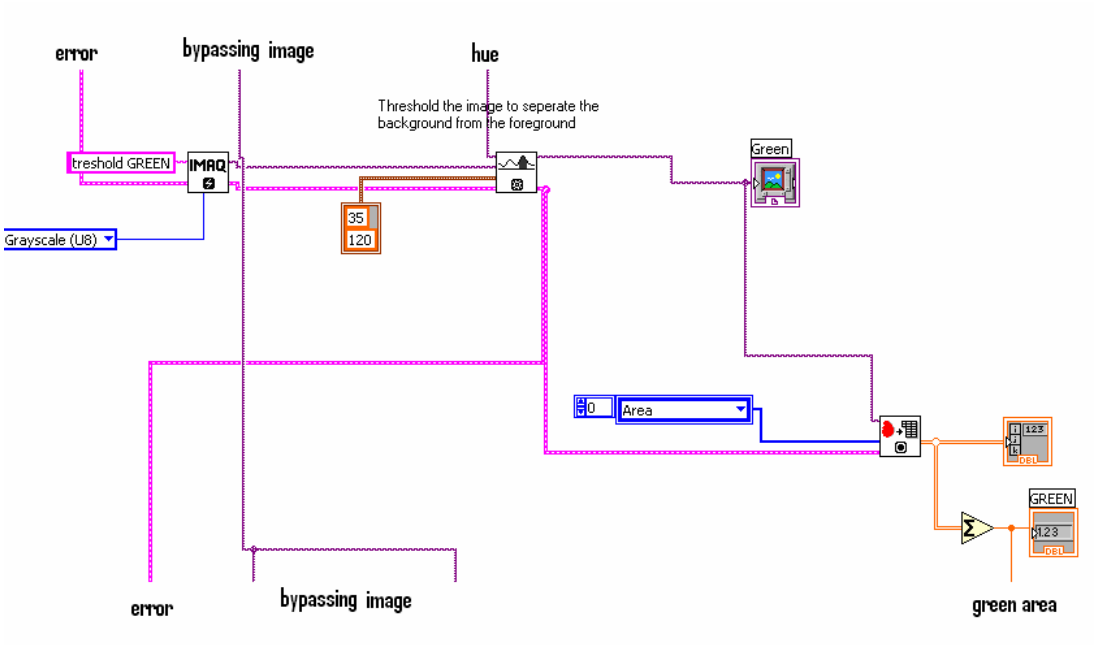


Figure 5-29: The code generating a new image with the threshold angles for yellowish-green to greenish-blue

Note that the threshold constants are the same as in the simulation, namely 35 to 120. In Figure 5-29 IMAQ Create is the first block. It generates the basis of an image.

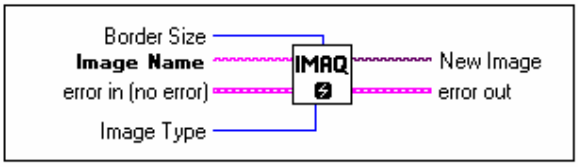


Figure 5-30: IMAQ Create

In Figure 5-30:

- **Border Size** determines the width, in pixels, of the border to create around an image.

- **Image Name** is the name associated with the created image. Each image created must have a unique name.
- **Error in (no error)** describes error conditions that occur before this VI or function runs. The default is no error.
- **Image Type** specifies the image type.
- **New Image** is the **Image** reference that is supplied as input to all subsequent (downstream) functions used by NI Vision. Multiple images can be created in a LabVIEW application.
- **Error out** contains error information.

This basis is now fed to a threshold block which extracts only the angles needed to represent the required colour scheme:

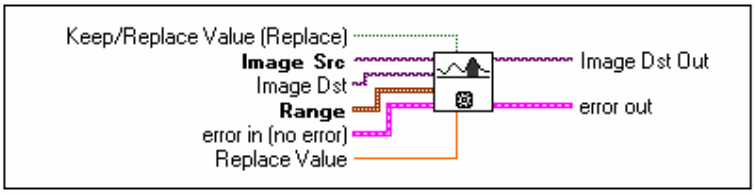


Figure 5-31: IMAQ Threshold

In Figure 5-31:

- **Keep/Replace Value (Replace)** determines whether to replace the value of the pixels existing in the range between **Lower value** and **Upper value**.
- **Image Src** refers to the source image.
- **Image Dst** refers to the destination image. If **Image Dst** is connected, it must be the same type as **Image Src**.

- **Range** is a cluster specifying the threshold range. In this case it will be user defined.
- **Error in (no error)** describes error conditions that occur before this VI or function runs.
- **Replace Value** is the value used to replace pixels between the **Lower value** and **Upper value**. As in the simulation, the default is 1.
- **Image Dst Out** refers to the destination image.
- **Error out** contains error information.

The result is given in Figure 5-32:

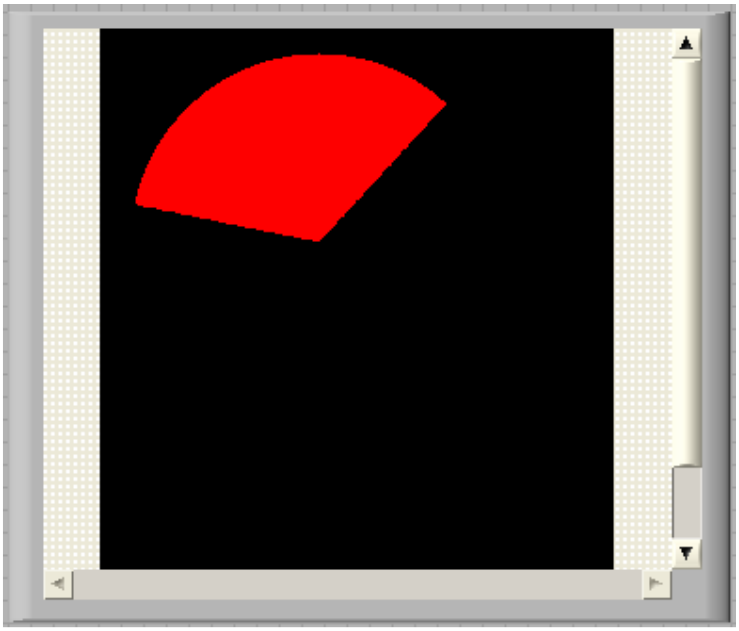


Figure 5-32: The area of green (from yellowish-green to greenish-blue)

The good correlation between Figure 5-32 with the simulation in Figure 4-20 indicates a successful implementation. Thereafter the area of the colour is calculated with the same code as that of the total area in section 5.1.4.

As in the simulation, the next colour to be extracted is from blue-green to violet. This is done with the same code as the extraction of yellowish-green to greenish-blue, where only the threshold values are different.

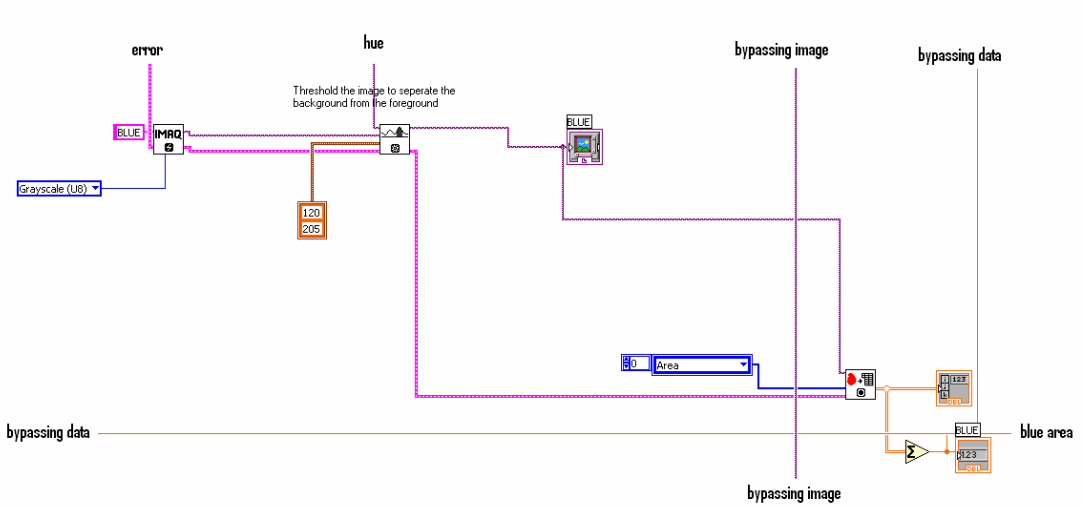


Figure 5-33: The code generating a new image with the threshold angles for blue-green to violet

Note that the threshold constants are the same as in the simulation, namely 120 to 205. The result is shown in Figure 5-34.

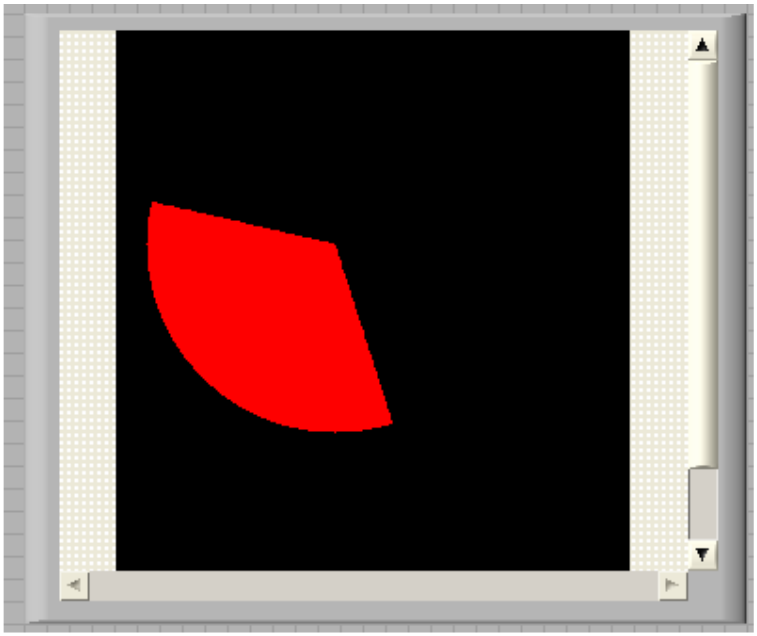


Figure 5-34: The area of blue (from blue-green to violet)

Figure 5-34 shows the added colour section and the good correlation between this and Figure 4-27,which indicates the success of the implementation.

The simulation in section 4.1.6 of violet to reddish-yellow is slightly different since it consists of a low and a high range of angles. As in the simulation it therefore consists of two threshold fictions that are summed together (Figure 5-35):

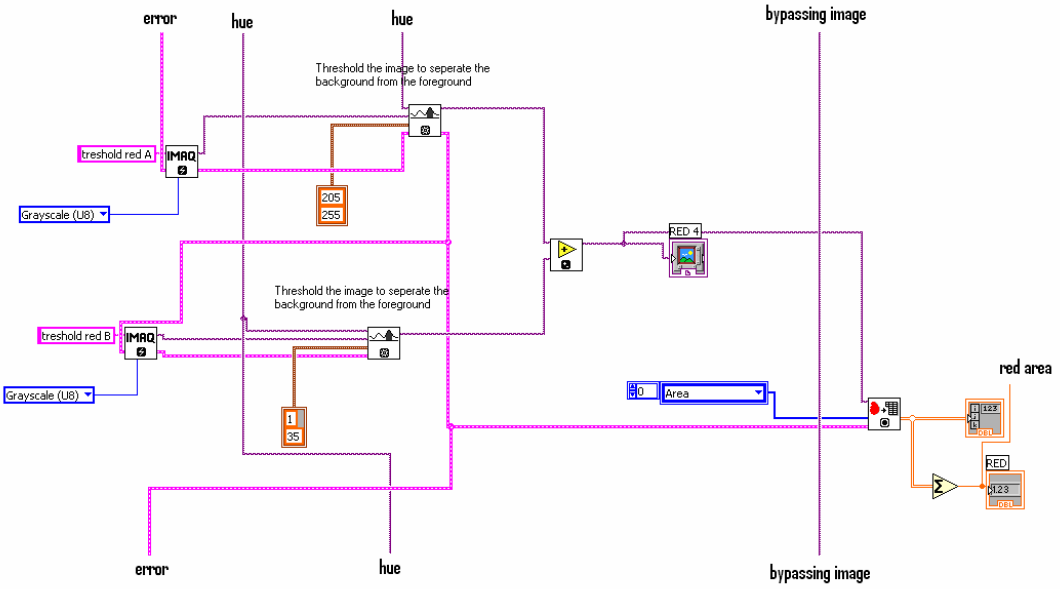


Figure 5-35: The code generating a new image with the threshold angles for violet to reddish-yellow

Again the thresholding code is done as with the previous colour section. The only difference is the threshold constants which are the same as those of the simulations, namely 1 to 35 and 205 to 255. This are added together as in Figure 5-36.

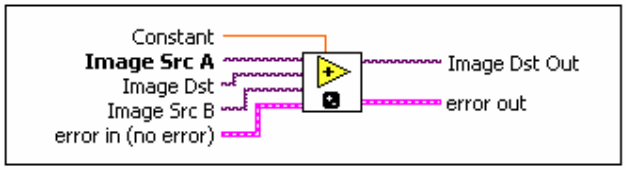


Figure 5-36: IMAQ Add

In Figure 5-36:

- **Constant** is the value added to Image Src A for image-constant operations. Constant is rounded down when the image is encoded as an integer. The default is 0.
- **Image Src A** refers to the first source image.
- **Image Dst** refers to the destination image.
- **Image Src B** refers to the second source image.
- **Error in** (no error) describes error conditions that occur before this VI or function runs.
- **Code** is the error or warning code. If the status is TRUE, the code is a non-zero error code. If the status is FALSE, the code is 0 or a warning code.
- **Source** describes the origin of the error or warning and is, in most cases, the name of the VI or function that produced the error or warning. The default is an empty string.
- **Image Dst Out** refers to the destination (output) image that receives the processing results of the VI. If Image Dst is connected, Image Dst Out is the same as Image Dst. Otherwise Image Dst Out refers to the image referenced by Image Src A.
- **Error out** contains error information.

The result is given in Figure 5-37.

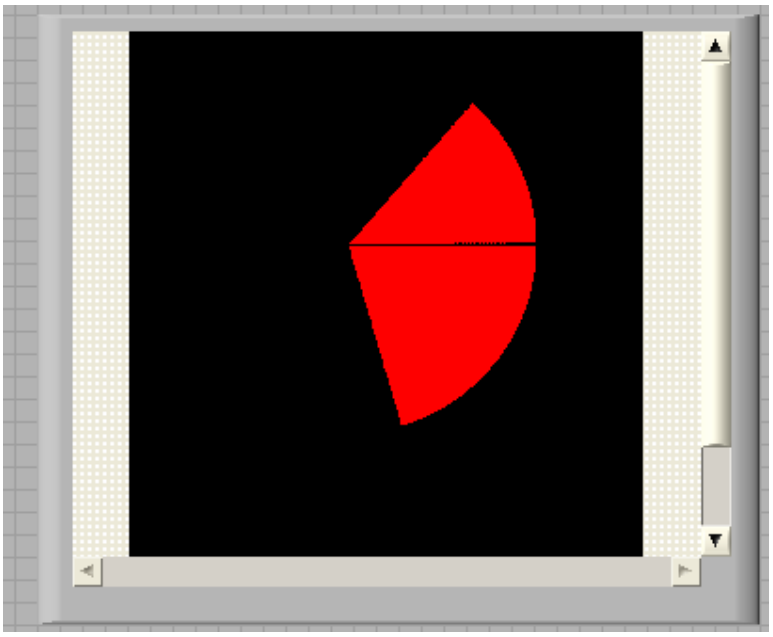
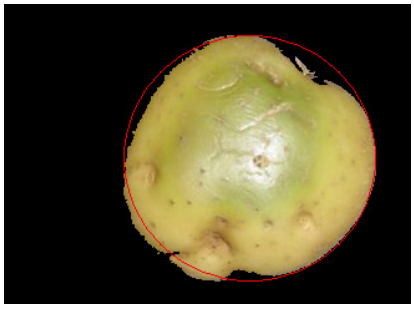


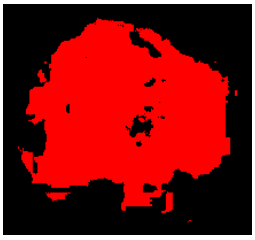
Figure 5-37: The area of red (from violet to reddish-yellow)

Figure 5-37 shows the added colour section, and the good correlation between this and Figure 4-26 indicates the success of the implementation.

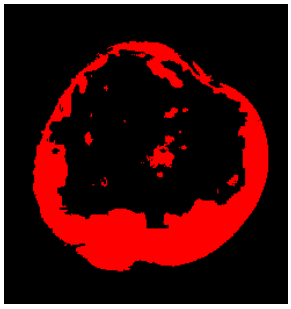
The result of this on a potato sample is as follows:



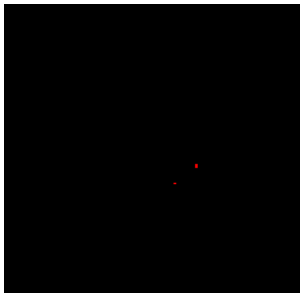
(a)



(b)



(c)



(d)

Figure 5-38: Hue areas showing the coloured areas of the potato image

Figure 5-38 shows the application of this function on the image in (a), where image (b) indicates the area of green, image (c) shows the area of red and image (d) shows the area of blue.

5.2 Real-time feature extraction

The bitmap read is part of the code that is changed with a code that will read in real-world samples in real-time. The part that would replace this with the camera input is given in Figure 5-39.

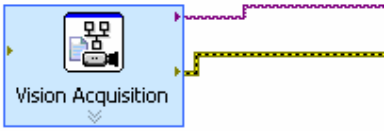


Figure 5-39: The Vision Acquisition code block

This block creates and edits acquisitioning using the NI Vision Acquisition Express VI. The NI Vision Acquisition Wizard is launched if this block is placed in the block window. There the edits can configure the camera variables.

The aperture of the camera must be as small as possible to gain depth of field. Depth of field is desirable in this image set-up, since an image of the sample in which every part is in focus. A problem that arises from this is that with a small aperture less light enters through the lens to the sensor, resulting in a darker image. To compensate for this, the shutter speed can be adjusted to expose the sensor longer to the object. The problem with this is that since the object is moving on the turn-table or conveyer, the exposure time cannot be too long, otherwise the image would be blurred. Another option would be to increase the intensity of the light source which is limited by the fiscal space. The gain of the camera can be adjusted, although this would aggravate the signal-to-noise level.

With all of these considerations was the setup as follow:

The focal length is set so that the focal point is 5 mm below the top of the object. With this aperture setting, the top and most of the rest of the object would be in focus. This is done as explained in section 2.1.1.3.

Experimentally the best combination was achieved with an aperture setting of *f*/2.5 as set on the camera. The rest is then edited in the software, or to be more specific, in the Vision Acquisition code block express window. These were as follows

- Brightness: 734 out of 1023
- Gain 255 out of 255
- Shutter speed 117 out of 4095

The images taken with these settings can be seen in the examples of the training set data in Figure 3-7, Figure 3-8, Figure 3-9 and Figure 3-10.

As explained in section 4.1 and coded in section 5.1, a greyscale image is needed for further coding. In Figure 5-1 this is done with a greyscale read code block, but in section 4.1.1 the colour image is converted to greyscale by using the red, green and blue data and applying the NTSC algorithm 4-4. The code to extract the red, green and blue data is shown in Figure 5-40:

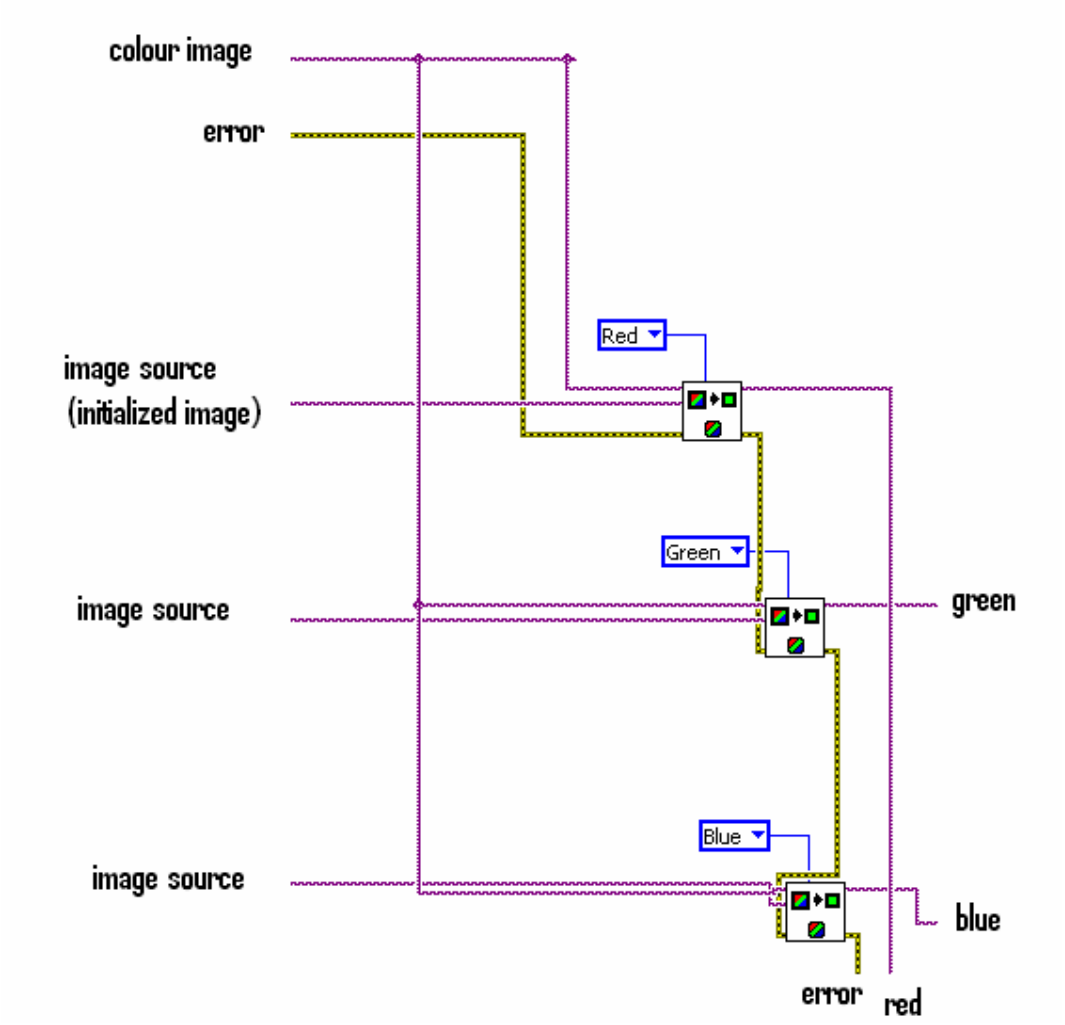


Figure 5-40: The colour extraction code blocks

The colour image as imported in Figure 5-39 is fed to these blocks where they are split up into red, green and blue based on the IMAQ Extract Single Colour Plane code block as in Figure 5-41:

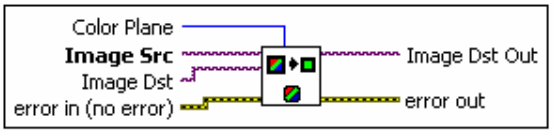


Figure 5-41: The IMAQ Extract Single Colour Plane code blocks

In Figure 5-41

- **Colour Plane** defines the colour plane to extract.

- **Image Src** is the reference to a colour image that has one of its colour planes extracted. If Image Dst is not connected, the source image is converted to an image that contains the extracted plane.
- **Image Dst** is a reference to the destination image. If Image Dst is connected, it must be the same size as Image Src.
- **Error in (no error)** describes the error status
- **Image Dst Out** refers to the destination image. If Image Dst is connected, Image Dst Out is the same as Image Dst. Otherwise, Image Dst Out refers to the image referenced by Image Src.
- **Error out** contains error information. These data are then applied to the

NTSC algorithm 4- 4 as in Figure 5-42:

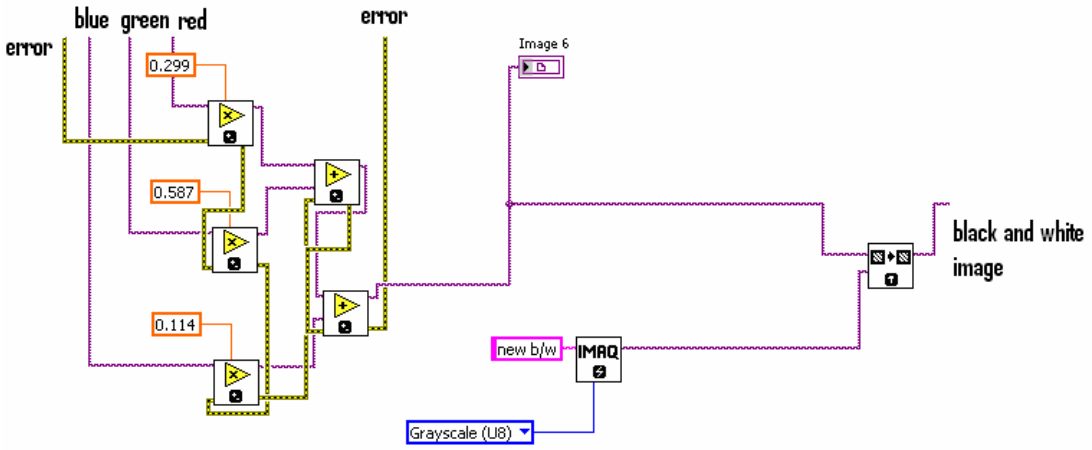


Figure 5-42: Implementation of the NTSC algorithm 4-4 to generate a greyscale image

The extracted colours are connected to the necessary multiplication block that will multiply the correct constant by its colour. For instance, red multiplied by 2.99 of the first block, then green will be multiplied by 0.587 of the second block and blue

will be multiplied by 0.114 of the bottom block on the left. These results are then added together and fed to the IMAQ copy block as in Figure 5-43:

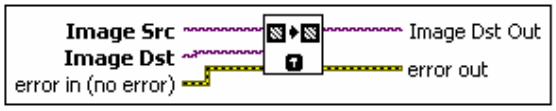


Figure 5-43: The IMAQ copy code block

In Figure 5-43:

- **Image Src** refers to the source image.
- **Image Dst** refers to the destination image.
- **error in (no error)** describes the error status before this VI or function runs.
- **Image Dst Out** refers to the destination image. If Image Dst is connected, Image Dst Out is the same as Image Dst. Otherwise, Image Dst Out refers to the image referenced by Image Src.
- **error out** contains error information.This function would now make this an 8-bit greyscale image as required for further processing.After these codes, one is left with a colour image and a greyscale image that connect directly to the rest of the code as seen in section 5.1.

5.3 Neural network

As described in section 3.5 and simulated in section 0, a neural network must be implemented with one hundred inputs consisting of percentages of size, shape, red, green and blue via the weighted connections to the four neurons in the centre. These are then connected via their weighted connections to another four neurons that have one output each that produce the result.

5.3.1 Transfer functions

The transfer functions for size and shape as well as the incrementing step function described in section 4.2.1 must first be implemented in LabVIEW for further connection to the neural network. This step function of equation 4-34 is implemented or coded in LabVIEW as shown in Figure 5-44:

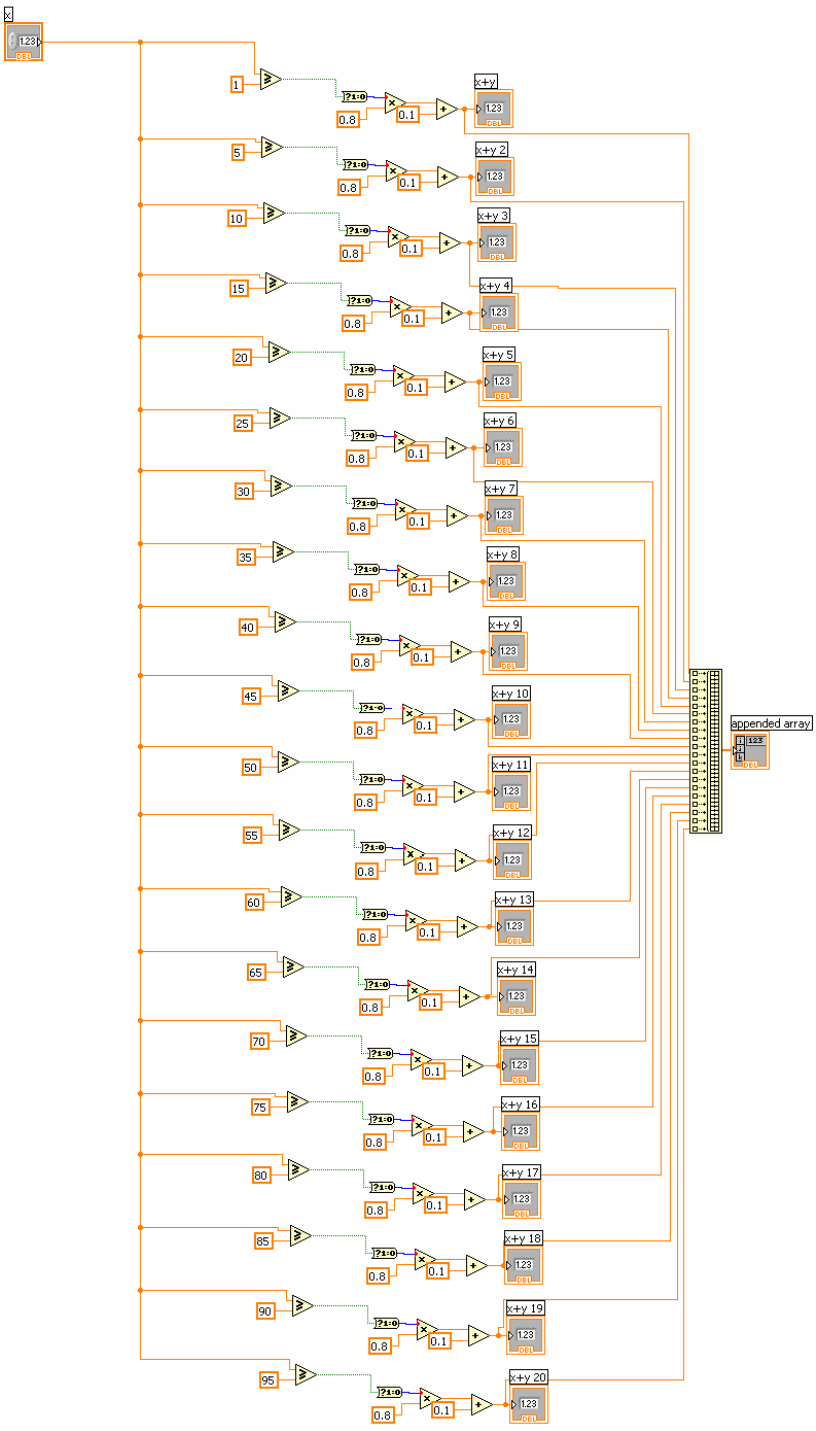


Figure 5-44: The incrementing step function of equation 4-34 coded in LabVIEW

Figure 5-44 shows the graphical programming of each step. Note that the first IF

statement in equation 4-34 is $\text{step1}(\text{size}) := \text{if}(1 \le \text{size\_out}(\text{size}), 0.9, 0.1)$ and is implemented by:

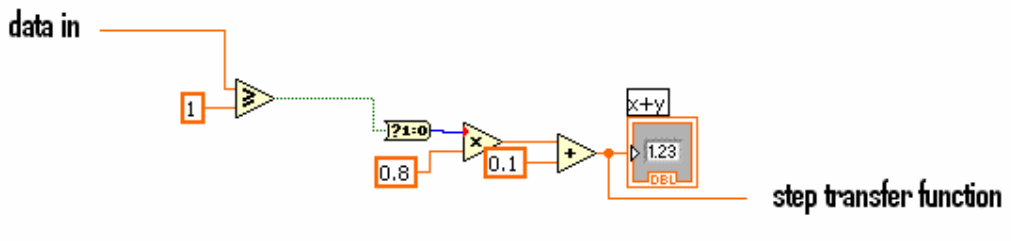


Figure 5-45: The incrementing step functions, first step IF statement equivalent

The percentage, or values between 0 to 100, is inputted from the left. The first code block it encounters is the Greater Or Equal block:

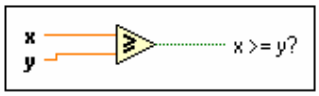


Figure 5-46: The Greater Or Equal block

In Figure 5-46:

- **x and y** must be of the same type.
- **x >= y?** returns the Boolean result of the operation. When one compares arrays, x >= y? is a scalar in Compare Aggregates mode and a Boolean array in Compare Elements mode (default).

This would give a Boolean 1 if the input is bigger than the defined constant, in this case 1, or otherwise 0. But $\text{step1}(\text{size}) := \text{if}(1 \le \text{size\_out}(\text{size}), 0.9, 0.1)$ states that the result should be 0.9 and 0.1.

The next code block is:

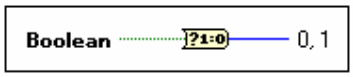


Figure 5-47: The Boolean To (0,1)

In Figure 5-47:

- **Boolean** can be a scalar, an array, a cluster of Boolean values, an array of clusters of Boolean values, and so on.
- **0,1** is decimal 0 if Boolean is FALSE and decimal 1 if Boolean is TRUE.

By taking this decimal of 0 and 1 and by multiplying it by 0.8 and adding 0.1 as shown in Figure 5-45, it can produce the required 0.9 and 0.1.

As shown in Figure 5-44, these IF statements are repeated and the only difference is the comparing constants which are the same as in equation 4-34, namely 1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90 and 95.

5.3.1.1 Adapted sigmoid transfer function for the size percentages as coded in LabVIEW

The transfer function for the size percentages must be sigmoid and, as stated in section 3.5 and simulated in section 4.2.1.1, must be adapted so that the best resolution is in the 50% range. The formula of 4- 35 in LabVIEW is as follows (Figure 5-48):

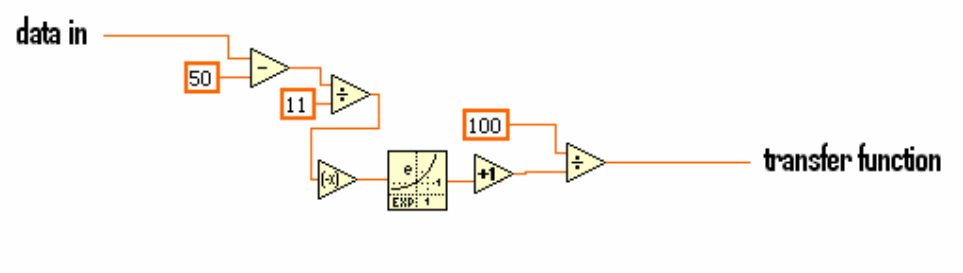


Figure 5-48: The sigmoid transfer function of 4- 35 as coded in LabVIEW

The percentages of size are fed in from the left where they are subtracted from 50. The next part of the formula states that it must be divided by 11. This is then also done by the divide block which divides the input by 11. The next step is to negate. This is done with the Negate block shown in Figure 5-49:

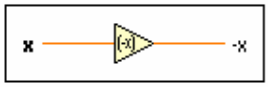


Figure 5-49: The Negate block

Where:

- **x** can be a scalar number, an array or cluster of numbers, an array of clusters of numbers, and so on. x cannot be an unsigned integer, because unsigned integers represent only non-negative integers.
- **-x** is the negative value of x.

After this it must be taken to the power of e. This is done with the Exponential block:

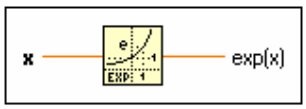


Figure 5-50: The Exponential block

In Figure 5-50:

- **x** can be a scalar number, array or cluster of numbers, array of clusters of

numbers, and so on.

- **Exp(x)** is of the same numeric representation as x. When x is of the form x = a + bi, that is, when x is complex, the following equation defines the exponential exp(x): exp(x) = exp(a)(cos(b)+i sin(b))

This answer must be added to 1 and this is done with the Increment block (Figure 5-51):

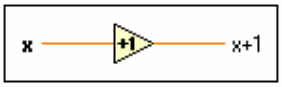


Figure 5-51: The Increment block

In Figure 5-51:

- **x** can be a scalar number, array or cluster of numbers, array of clusters of numbers, a time stamp, and so on.
- **x+1** is the result of x+1. If the input is a time stamp value, this function increments the time by one second. If the input is an enumerated type value, this function increments the last enumerated value to the first.

According to formula 4-35, the 100 must be divided by this answer, which is done as shown in Figure 5-48, in the last block producing the final answer on the right side. This is then fed to the incrementing step function.

5.3.1.2 Adapted log transfer function for the shape percentages as coded in LabVIEW

As explained in section 3.5 and simulated in section 4.2.1.2, the transfer function of the shape percentages must be a log function. It was also stated in section 3.5 and simulated in section 4.2.1.2 that it must be adapted so that the best

resolution is in the lower ranges.

The formula of 4-37 is coded in LabVIEW as follows:

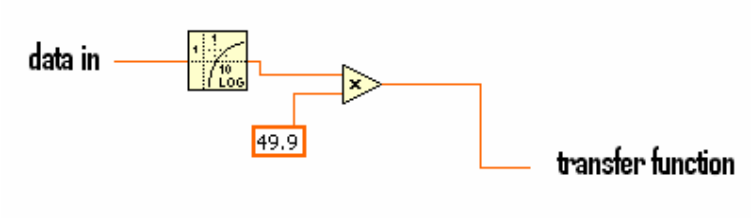


Figure 5-52: The log function of formula 4- 37 as coded in LabVIEW

The first part of formula 4-37 states that the input must be log. To do this, the input from the left is connected to a logarithm block:

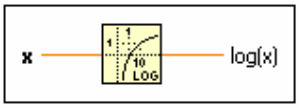


Figure 5-53: The Logarithm Base 10 code block

In Figure 5-53:

- **x** can be a scalar number, array or cluster of numbers, array of clusters of numbers, and so on.
- **Log(x)** is of the logarithm with base 10 of x.

This answer must be multiplied by 49.9 as in Figure 5-52, producing the answer on the right, which in turn is connected to the incrementing step function.

5.3.1.3 Linear function for the colour percentages as coded in LabVIEW

As stated in section 3.5, the inputs of red, green and blue cannot be transformed, since they have a linear relationship to each other. These are directly connected to their individual incrementing step functions.

5.3.2 A one hundred input to four neurons to another four neurons with a four-output neural network as coded in LabVIEW

As described in section 3.5 and simulated in section 4.2.2, a neural network will be implemented with one hundred inputs of percentages of size, shape, red, green and blue, of which the size and shape first go through a transfer function. The network is able to produce four outputs labelled A, B, C, and D which could activate actuators and thus sort the objects into options A, B, C or D.

5.3.2.1 Forward pass

The input data from the extraction part are compiled for an input matrix for the neural network as follows in Figure 5-54:

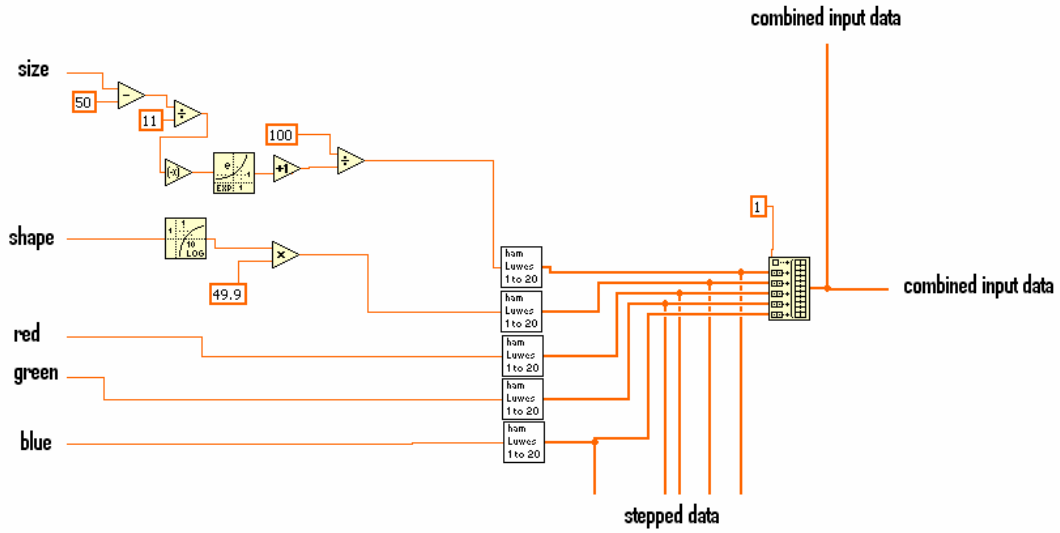


Figure 5-54: The input matrix for the neural network

The inputs of percentages are connected to the left where the top input (connected to the sigmoid) is size. The second input from the top (connected to the log function) is shape. The last three inputs on the left are of course red,

green and blue. These are then connected to the incrementing step function blocks which are user-coded:

Figure 5-55: The user-defined incrementing step function

These blocks are the compiled code as shown in Figure 5-44. They produce a series of twenty incrementing values of either 0.1 or 0.9 as explained in section 4.2.1.

These answers are connected to a Build Array block:

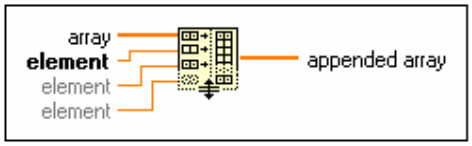


Figure 5-56: Build Array code block

Where:

- **Array** or **element** can be any n-dimensional array or scalar element. All inputs must be either elements or 1D arrays or n-dimensional and (n-1)-dimensional arrays. All inputs must have the same base type.
- **Appended array** is the resulting array.

Note that a constant of 1 is connected to the first input of this array. Thereafter all the input arrays of size, shape, red, green and blue are connected. The 1 is the activation input for each neuron.

After this, there is now an input matrix of one hundred and one values. This is connected to the first layer, as visualised in Figure 3-6 and simulated in section 4.2.2. The code of the first layer is as follows (Figure 5-57):

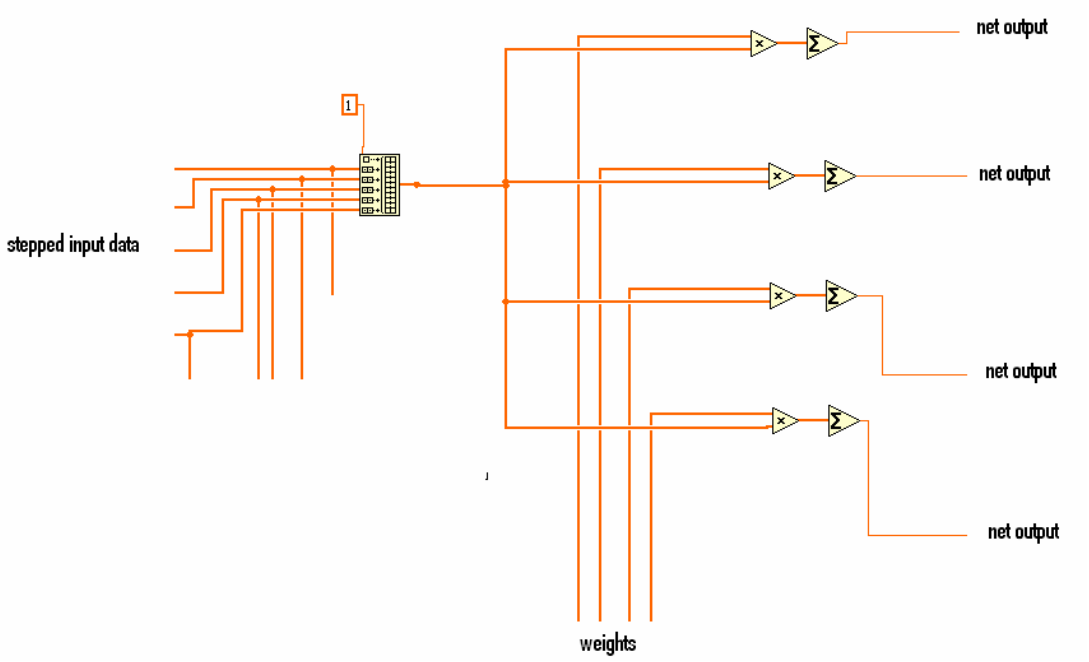


Figure 5-57: The first layer of four neurons as in equation 2-22

Figure 5-57 describes the implementation of equation 2-22. This formula is, as in the simulation in section 3.5, a basic sigma function that adds the multiplications of inputs and their weights together. This is seen as the inputs from the build array block (as filled with the data given in Figure 5-54) on the left connected to multiplication blocks. The other inputs are from the weights. This full matrix is then added together with the sigma block as in Figure 5-58:

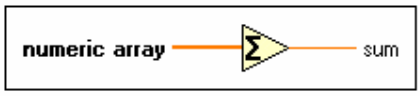


Figure 5-58: The Add Array Elements code block

In Figure 5-58:

- **Numeric array** can have any number of dimensions.
- **Sum** is of the same data type and structure as the elements in the numeric array.

As described in section 3.5 and simulated in section 4.2.2, initial random weights between 0 and 1 must be generated. These initial random weights will of course adapt and change as the system learns new operations. Generating these random weights is done as follows in Figure 5-59:

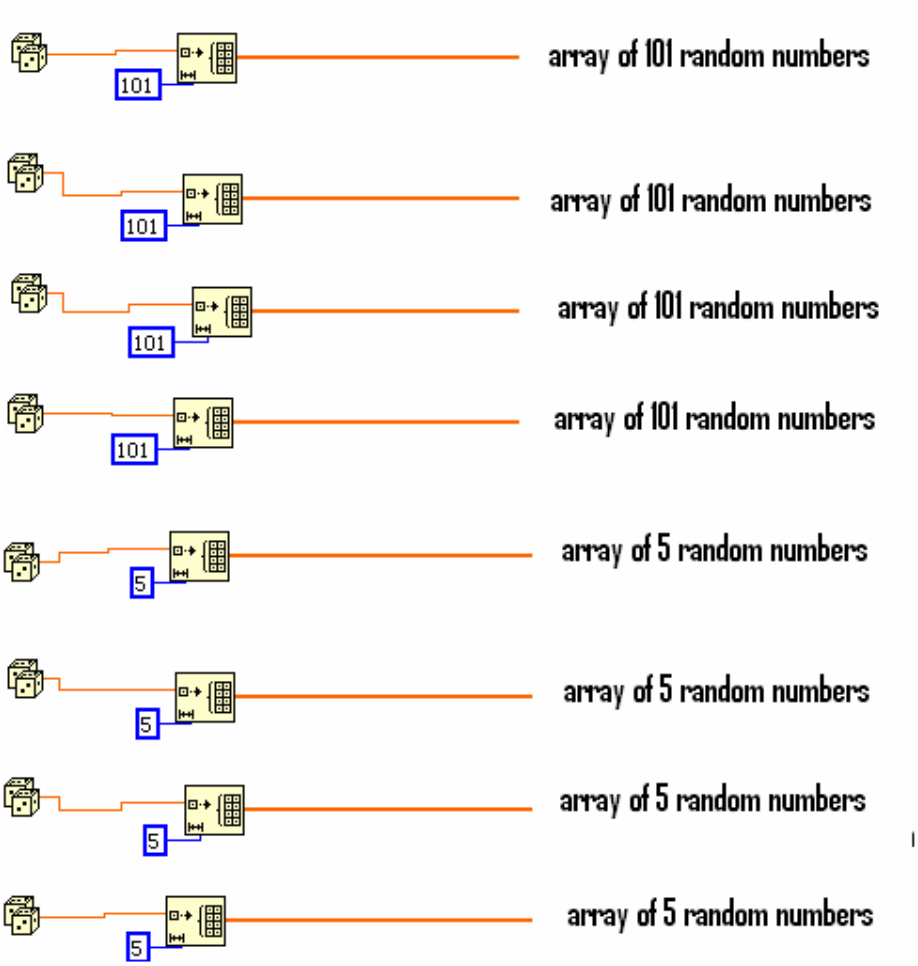


Figure 5-59: The code for generating random weights for the first and second layer

The first building block on the left is the random number (0-1) generator:

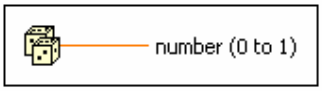


Figure 5-60: Random Number (0-1) code block

This block produces a double-precision, floating-point number between 0 and 1, exclusively. The distribution is uniform. This is connected to the Initialize Array block in Figure 5-61:

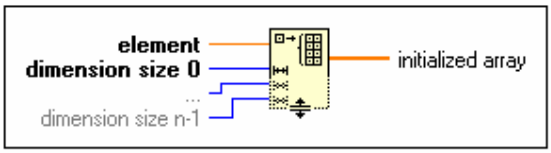


Figure 5-61: Initialised array code block

In Figure 5-61:

- **Element** is the value used to initialise all elements of the initialised array. This element can be any scalar type.
- **Dimension size 0**…n-1 must be a number. The function creates an empty array if any dimension size is 0. n dimension size terminals are required for n-dimensions.
- **Initialised array** is an array of the same type as the type wired to an element.

Note that the blocks in Figure 5-59 initialise eight arrays in total, of which the first four have one hundred and one values, and the second four have five values each. These are the weights connected in Figure 5-57, but not before they are connected to a shift register that will replace them with the newly taught weights in the next cycle.

After the answers are generated at the left in Figure 5-57, they must be applied to

the sigmoid function of equation 2-23 as in Figure 5-62:

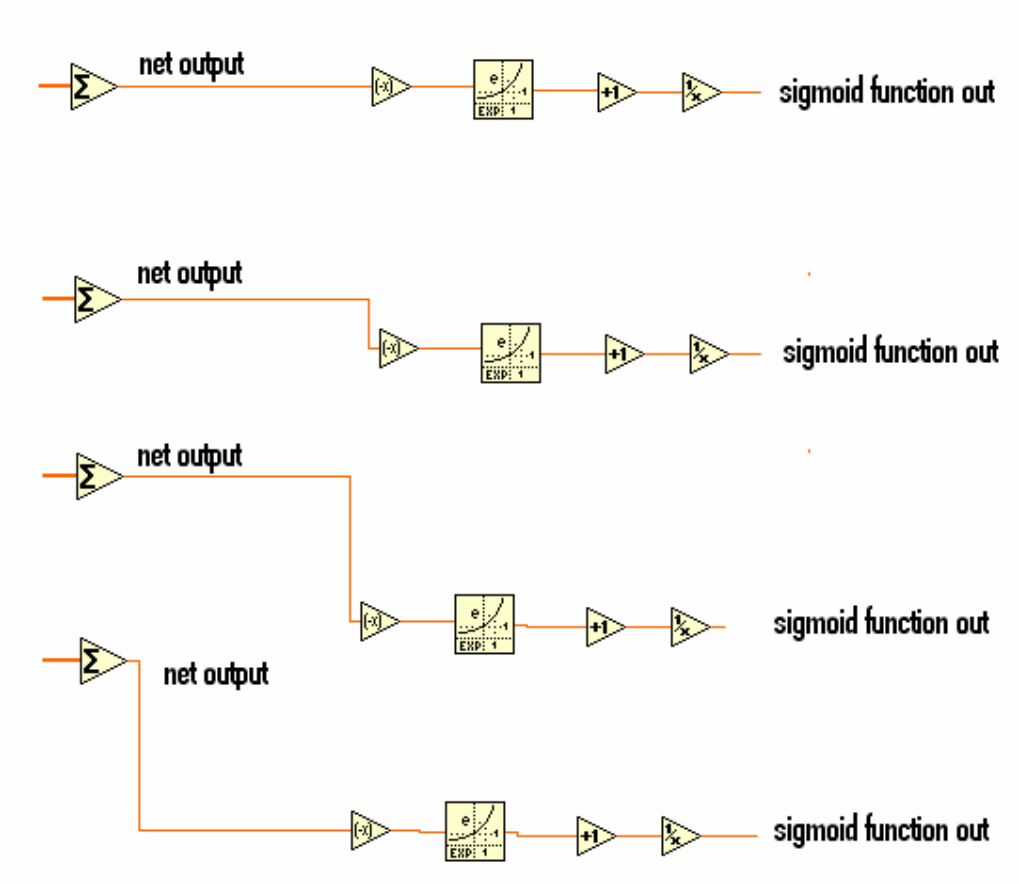


Figure 5-62: The sigmoid function of equation 2-23

The sigma blocks on the left are the outputs of equation 2-22 from Figure 5-57. This is applied to function 2-23 which states that this input must be negated, which is done with the individual negate blocks. This must be raised to the power of e with the exponential block. Next it must be incremented with 1 and this is done with the incremental block. Then according to the formula this must the reciprocal, hence the Reciprocal block:

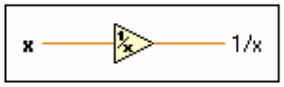


Figure 5-63: The Reciprocal code block

In Figure 5-63:

- **x** can be a scalar number, array or cluster of numbers, array of clusters of

numbers, and so on.

- **1/x** is infinity if x is 0. If x is an integer, 1/x is a double-precision, floating-point number.

These four outputs are connected to a build array block of which the first input is 1 for activation of the next layer. This is thus the five inputs for the next layer. The next layer is implemented as in Figure 5-57 and Figure 5-62 with the initial weights as generated in Figure 5-59 and later adapted with the help of a shift register. This layer has only five inputs, not one hundred and one like the first layer.

5.3.2.2 Actuator decision code

This neural network has four outputs and after training them it will give a prediction of which one should be active. The network produces any value but it is trained to give 0.1 for binary 0 and 0.9 for binary 1. It still is a prediction mathematical system and it can produce a value of, for instance, 80% chance of a high value or 0.72 instead of 0.9 (0.72/0.9 x 100 = 80%). The actuators work with full binary, which means that a 1 would activate it and a 0 would make it return. The four outputs are labelled A, B, C and D and they obviously need some code that would transform this prediction output to a Boolean output. This function is as follows in Figure 5-64:

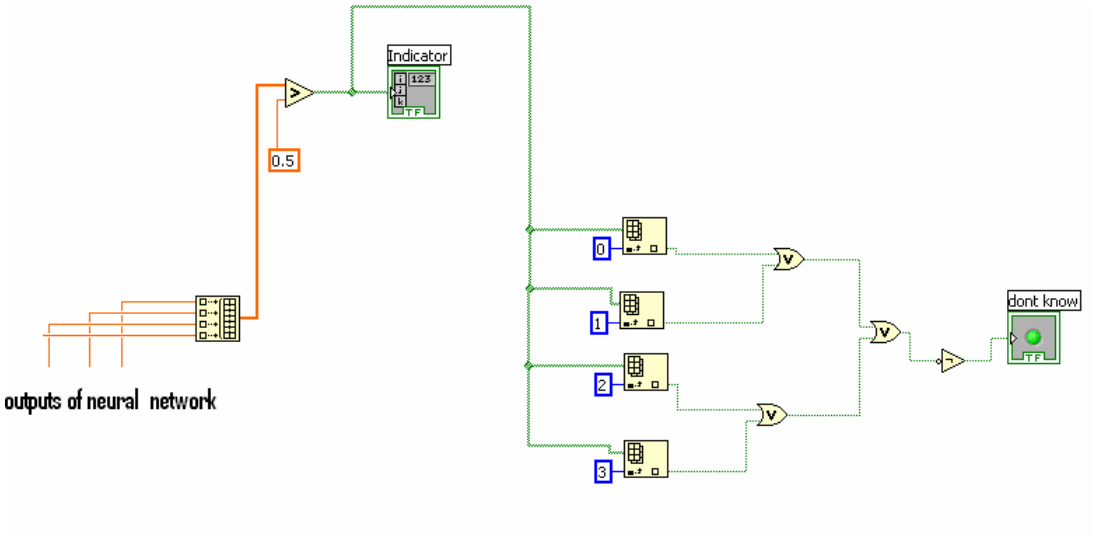


Figure 5-64: The actuator Boolean transform

The four inputs on the left that are connected to the build array block are the four outputs of the neural network. This array is taken to be greater than block, which compares it to a constant of 0.5, resulting in binary. If an output of the neural network is higher than 0.5, it would produce a 1, otherwise a 0. The block labelled Indicator is just an indicator that will display the results to the user. This array is now expanded again into its separate elements using the index array blocks. The direct output of this can be used to drive the actuator via the DSP card. These outputs are, however, connected to a logic circuit (or gates) that would indicate whether the network is not sure of an output. This system has five indirect outputs, namely the four actuator controllers, and if none of them is active an object can be dropped off the end of the conveyer, namely output five.

5.3.2.3 Backward pass

As stated earlier, the backward pass is the function of a neural network that gives it the ability to learn. The first part is calculating the error of each layer. The error

of the last layer is calculated first. This layer is calculated using equation 2-24 as simulated in section 4.2.2.

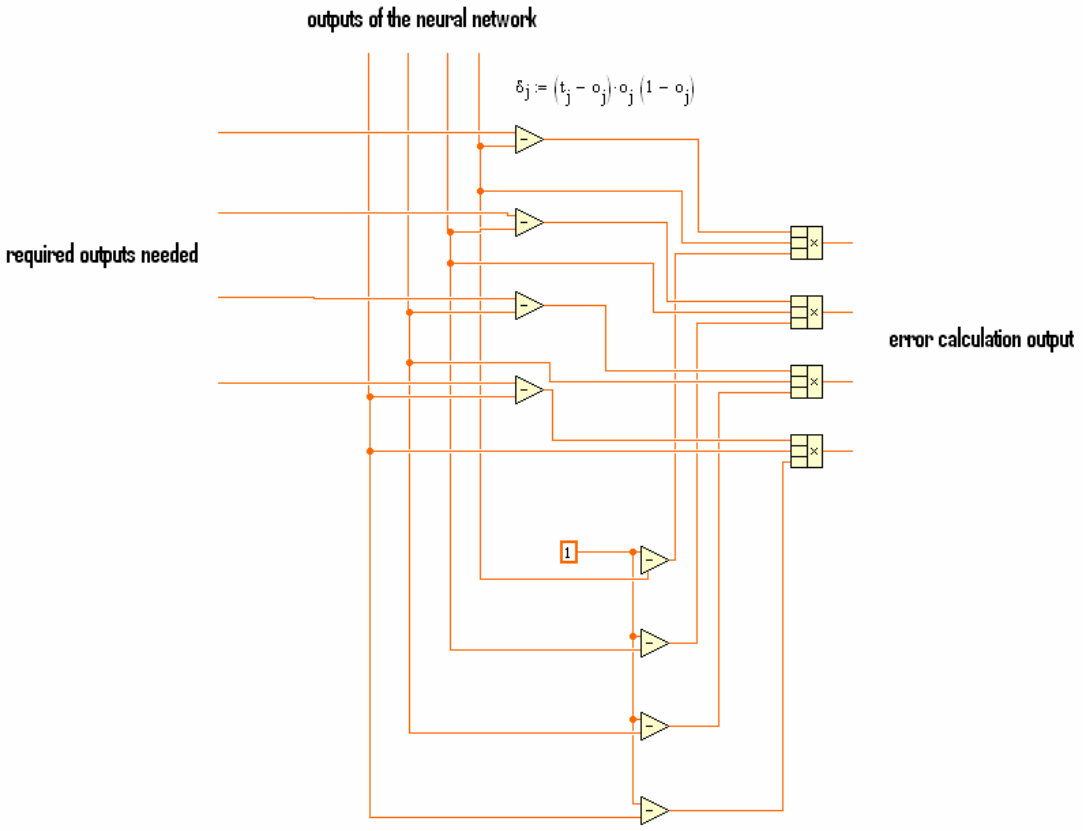


Figure 5-65: The error calculation for the last layer as in equation 2-24

The inputs at the top of Figure 5-65 are the inputs from the four separate outputs of the neural network. The inputs from the left are the required outputs needed as defined by the training data set. These are subtracted as in equation 2-24. Equation 2-24 also states that 1 must be subtracted from each network output and this plus the answers of the previous subtraction must be multiplied by the values of the outputs of the network. This is done with the Compound Arithmetic code block. This block performs arithmetic operations on one or more numeric, array, cluster or Boolean inputs. The designer can select the operation (Add,

Multiply, AND, OR, or XOR) which in this case was multiplication. The block in default setting is as follows (Figure 5-66):

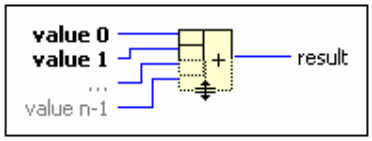


Figure 5-66: The Compound Arithmetic code block

In Figure 5-66:

- **value 0…n-1** can be a number or Boolean value, an array of numbers or Boolean values, a cluster, array of clusters, and so on. One can wire a waveform to only one value input. If an input is a waveform, one can have an unlimited number of scalar inputs of varying sizes.
- **Result** returns the result of the selected operation applied to the value 0...n-1. For AND, OR, or XOR, **result** returns the bit-wise operations on numeric inputs and logical operations on Boolean inputs.

The next calculation is of the error of the first layer using equation 2-24 and as simulated in section 4.2.2.

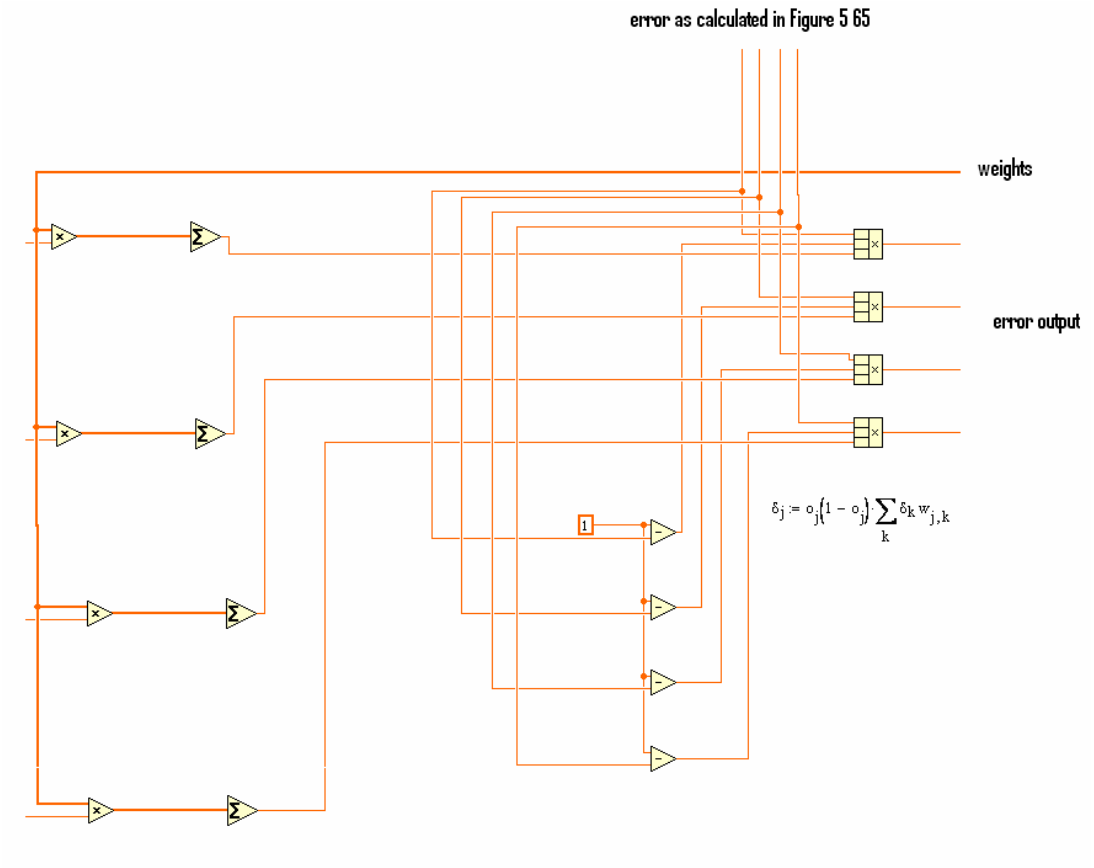


Figure 5-67: Error calculation for the first layer as in equation 2-25

Figure 5-67 shows how the error for the first layer is calculated by implementing equation 2-25. On the left is part of the sigma of the formula. It multiplies the error as calculated in Figure 5-65 by its corresponding weight. The inputs on the left are thus the corresponding weights, where the other input to the multiplier block coming from the right is the error as calculated in Figure 5-65. The middle part is as the formula states, that 1 must be subtracted from each output of this layer of neurons. This must be multiplied by each output of this layer of neurons and by the output of the sigma part of the equation. This is realised with the Compound Arithmetic code blocks (set to multiply) on the right.

The weight change is done next with equation 2-26 as in Figure 5-68:

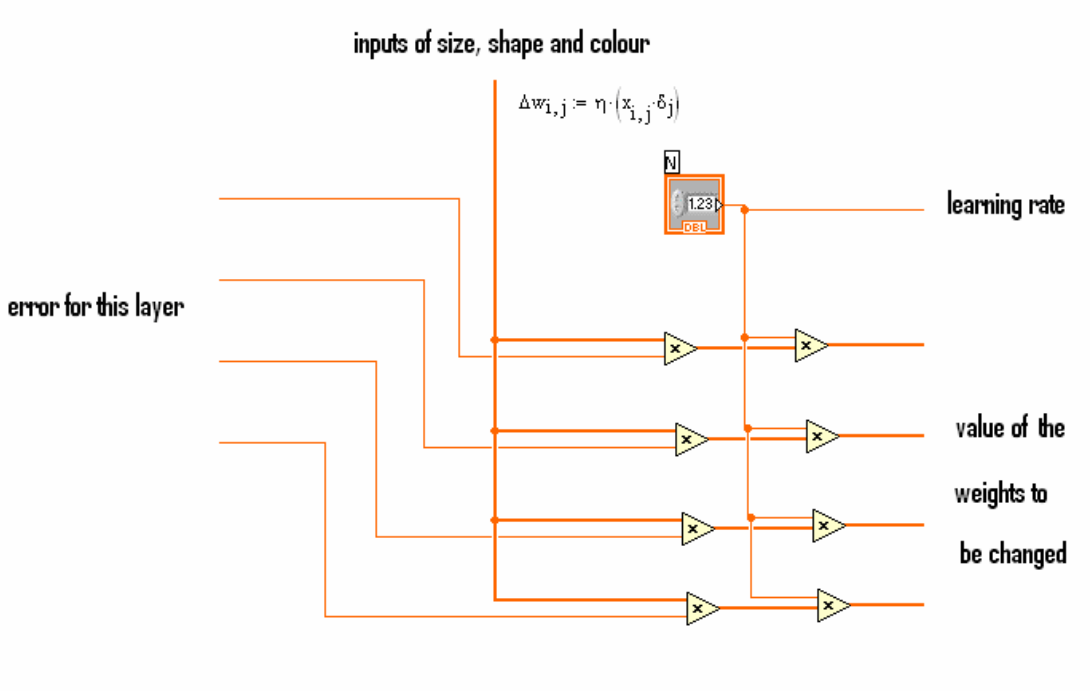


Figure 5-68: The weight change code for equation 2-26

Figure 5-68 shows how the weight change of equation 2-26 is implemented for the first layer. The calculation for the second layer is a duplication of this, using the same learning rate, but of course it uses its own error and its inputs would be the outputs of the first layer. The inputs on the top connected to the layer of multipliers in Figure 5-68 are, however, the inputs of size, shape and colour. These are multiplied by the error for this layer. This result is then multiplied by the learning rate.

All the weights are now adjusted using equation 2-27 and connected to individual shift registers, so that the new weights are used in the next cycle of operation. This is implemented as follows in Figure 5-69:

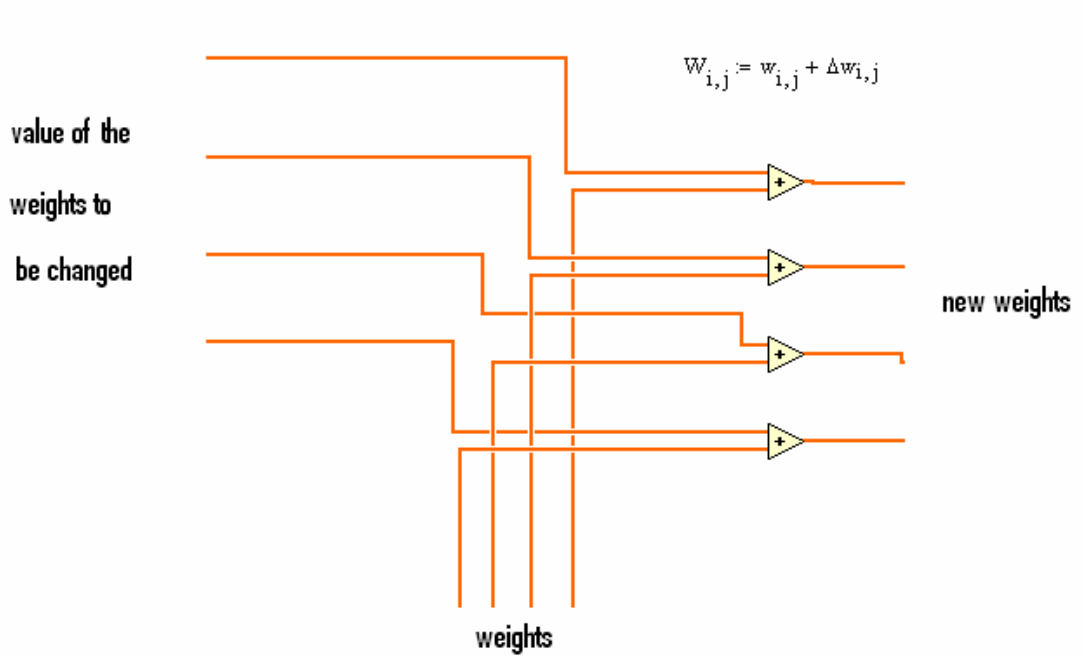


Figure 5-69: The weight adjusting code for equation 2-27

Figure 5-69 again only applies to the first layer, but it is duplicated for the second layer, using the necessary data. Equation 2-27 is quite simple in that it just adds the old weights (inputs on the left) to the weight change (inputs from the bottom) as calculated in Figure 5-68. These outputs are the new weights to use in the next operation loop.

5.3.2.4 Neural network coding evaluation

To evaluate the coded “one hundred inputs to four neurons to another four neurons with a four-output neural network”, it is fed the same inputs and constants as the mathematical simulation in section 4.2.2.

The weight generation part as shown in Figure 5-59, would normally generate random numbers, but to evaluate these, random generators are replaced with constant values the same as in section 4.2.2. This is done as follows:

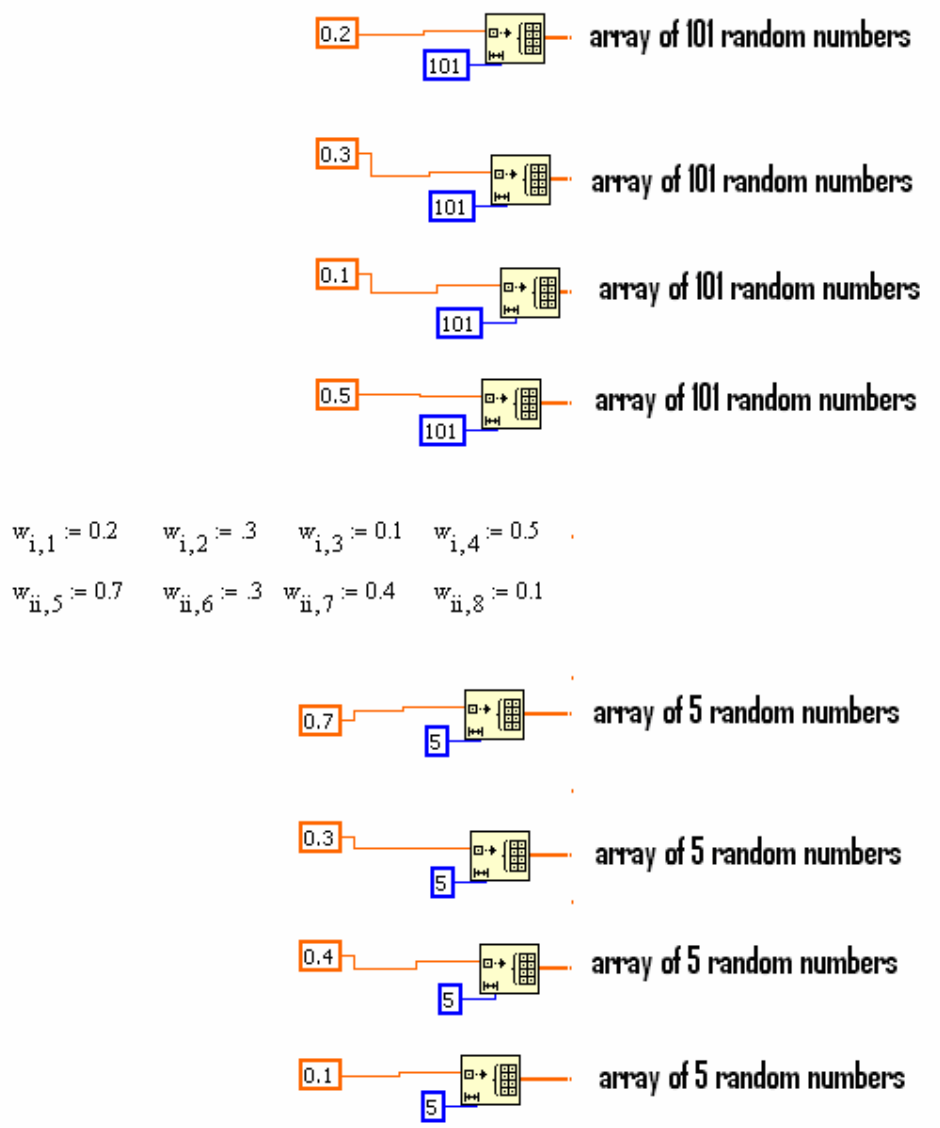


Figure 5-70: The code for generating constant weights for the first and second layer as per simulation

Note that the values of the weights are the same as defined for the simulation in equation 4-52 to 4-59.

The inputs must also be the same as defined for simulation in equation 4-48 to 4-51. This is done as follows:

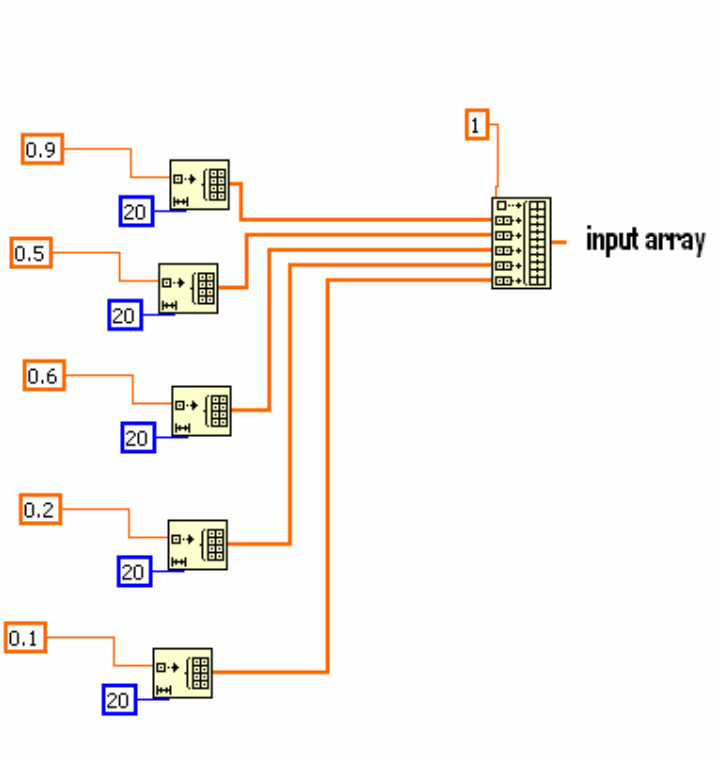


Figure 5-71: The code for generating similar inputs as those of the simulation

This figure shows how the input array is built up by generating five arrays of twenty inputs each with the constant as defined in the simulation in equations 4-48 to 4-51. These answers were 0.9 for area, 0.5 for shape, 0.6 for red, 0.2 for green and 0.1 for blue. This system is then, as in the simulation, stepped forward ones, of which the front panel is as follows:

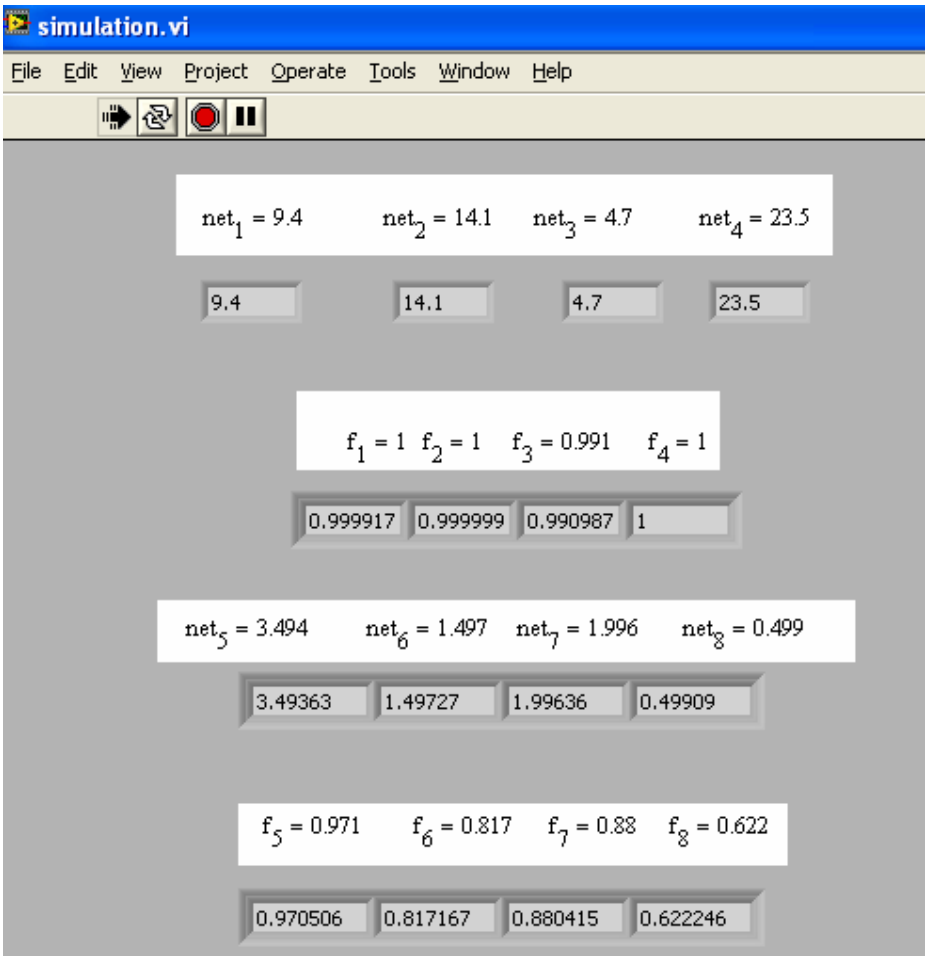


Figure 5-72: The front panel for the coded neural network to evaluate by comparing to the simulation

The values of the simulation are pasted into the coded software to compare the answers. Note how these simulated answers correlate with the implementation answers in the display windows under each of them. The only difference is that in the simulation the values are rounded off by the software to three decimal places. To evaluate the ability to learn, the outputs are redefined as in the simulation with equations 4-66 to 4-69. As in the simulation, the neural network is stepped through a backward sweep with a learning rate of 10. After this, the outputs are re-evaluated for correlation with the simulation:

Figure 5-73: The window on the front panel for the coded neural network showing the result of the final layer after one training cycle similar to that of the simulation

Note the good correlation between this and the equivalent values of the simulation in section 4.2.2, namely:

$f_5 = 0.905$

$f_6 = 0.021$

$f_7 = 0.109$

$f_8 = 0.977$

Similar to the simulation, the network is able to produce the required output after twenty-five cycles:

Figure 5-74: The window on the front panel for the coded neural network showing the result of the final layer after twenty-five training cycles similar to that of the simulation

Figure 5-74 shows a section of the front panel of the implemented code for the neural network. The results displayed are the answers produced, as in the simulation in section 4.2.2, after twenty-five cycles. All the variables are set to those of the simulation in section 4.2.2. The good correlation here indicates successful implementation of the simulation to the implementation code.

5.3.3 Training and data sets

To generate training data, an addition was made to the LabVIEW code as discussed thus far. It is so coded that an object, for instance potato (a) in Figure 3-8, is placed under the camera, and then one of the buttons labelled A, B, C or D on-screen is pressed. In this case for potato (a) it would be A since all the small not-green potatoes must go with actuator A.

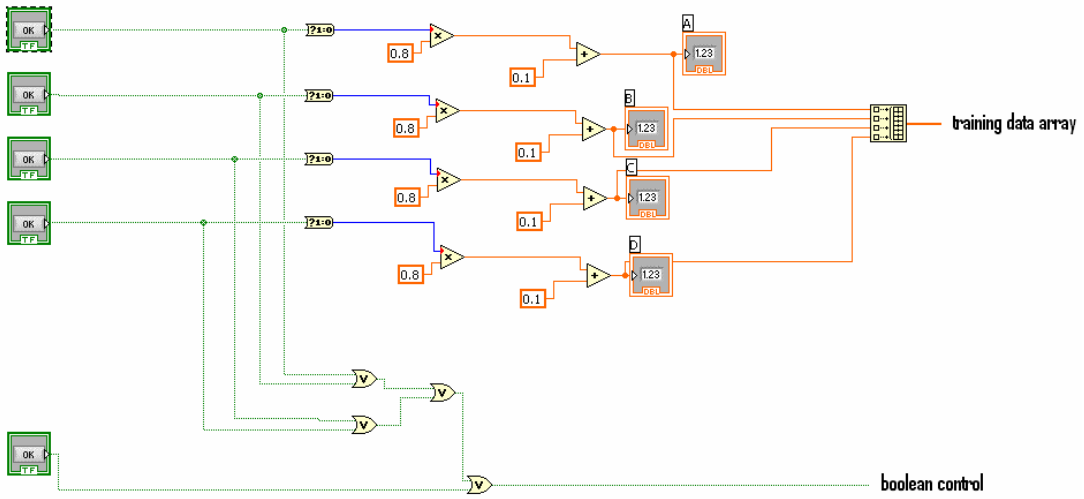


Figure 5-75: The training data set generation code

Figure 5-75 shows how these Boolean switches on the left are converted to decimal 0 or decimal 1 and then multiplied by and added to the constants to produce the required 0.9 and 0.1. The build array block after the image extraction part (not shown here) then gather the data of percentage of size, shape, red, green and blue and save it to a spreadsheet with the data from the build array block in Figure 5-75. This generates a file with percentages of size, shape, red, green and blue and the trailing 0.9, 0.1, 0.1, 0.1. In this example it would mean that A is 0.9, B is 0.1, C is 0.1 and D is 0.1. The logic circuit at the bottom generates the activation for the write to file block. The switch at the bottom is just

to write a value where A is 0.1, B is 0.1, C is 0.1 and D is 0.1. This write to file block is as follows (Figure 5-76):

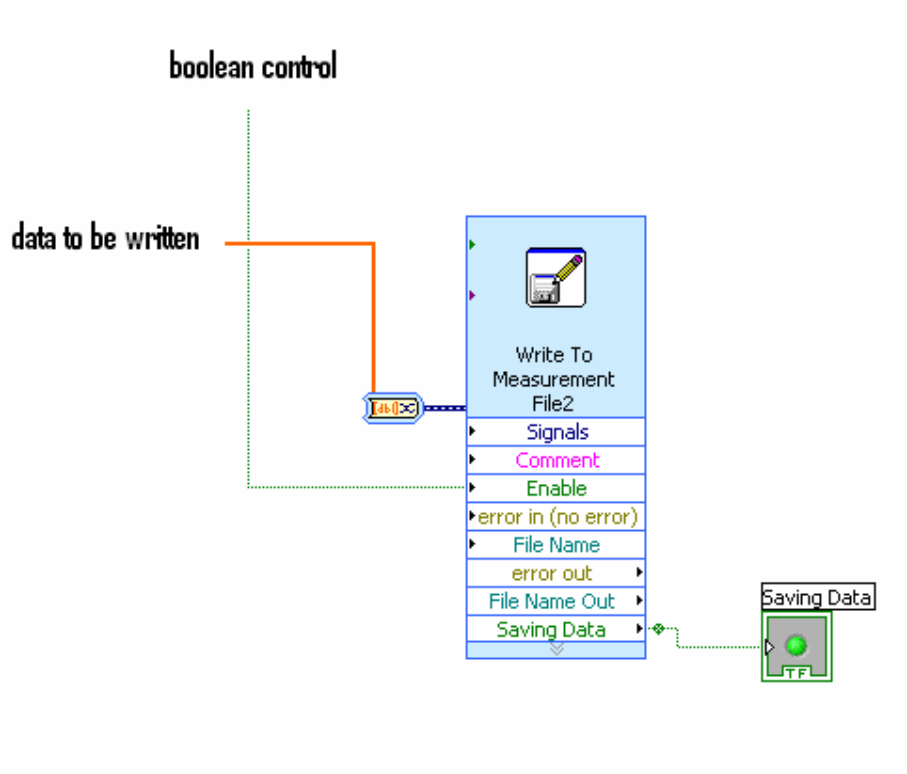


Figure 5-76: The training data set write to file code

In Figure 5-76:

- **Signals** contains the input signal or signals. If two or more signals with the same name are wired to the Signals input, LabVIEW appends an integer to the end of the names written in the file, which enforces unique channel names. For example, if two signals named Sine are wired to the Signals input, LabVIEW writes the names as Sine and Sine 1.
- **Enable** Enables or disables the Express VI. The default is ON or TRUE.

The path and file name were defined in the initialised placement of this block. The enable input on the left is connected to the logic diagram output.

After this the next sample can be placed under the camera, and by pressing the

correct group button the data and selection be will appended to the spreadsheet file. A sample of such a file is given in Table 5.

Table 5: Training data spreadsheet file

38.60677
1.753794
31.15514
72.27656
0
0.1
0.1
0.9
0.1
39.05273
1.116946
42.21055
60.12336
0
0.1
0.1
0.9
0.1
34.63542
6.917293
33.82519
69.56767
0
0.1
0.1
0.9
0.1
72.62695
7.077226
44.44444
58.43754
0
0.1
0.1
0.1
0.9
65.41016
1.139644
30.60117
73.13128
0
0.1
0.1
0.1
0.9

This training data sequence is randomised and then fed to the neural network

implementation in LabVIEW. It adapts the weights to produce these results. These weights are saved and read into a LabVIEW executable that consists of the image extraction and forward pass of the network.

The system can now be evaluated by placing the test data directly on the turn-table and testing the operation.

6 RESULTS

The objective is to design an artificial intelligence (using neural networks) machine vision system that can sort produce and products on a conveyer belt system. It must be able to be taught and retrained like a human. It must also be able to make predictions, thus giving it the ability to grade unique agricultural and biological products.

The system design is based on two aspects, namely the hardware set-up and the intelligent software component.

6.1 The hardware set-up

The hardware set-up was successfully implemented if one compare the conceptualised design in the CAD as seen in Figure 3-1 to Figure 3-3, compared with the actual equipment shown in Figure 3-4 or as seen below in Figure 6-1 which show Figure 3-1 and Figure 3-4.

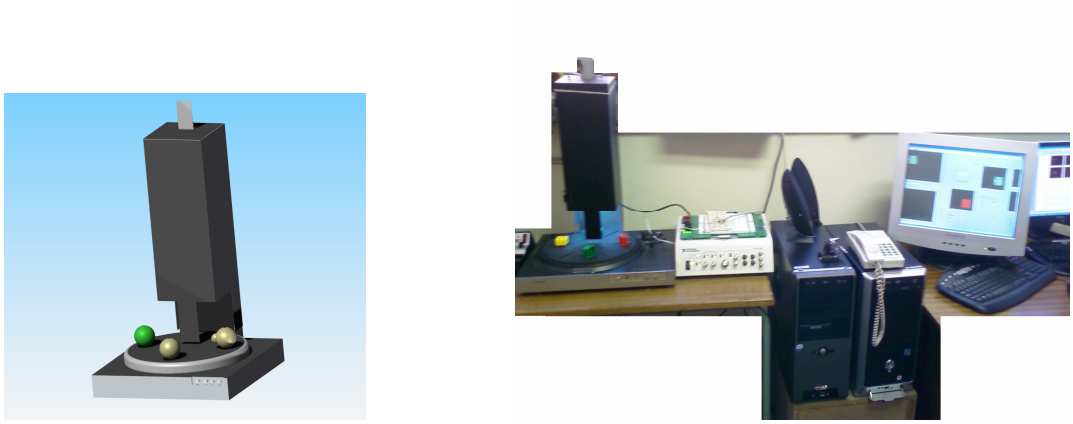
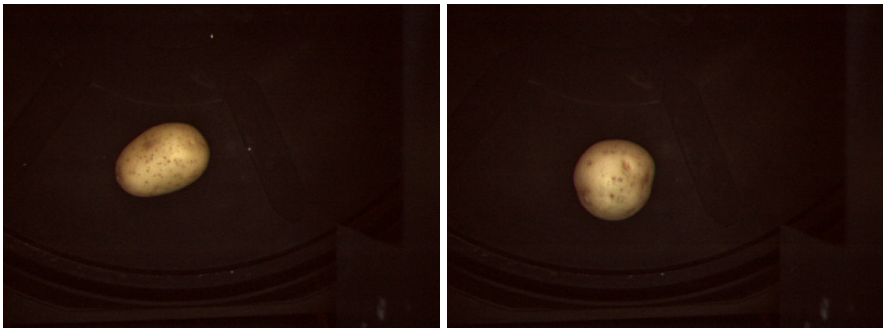


Figure 6-1: This figure show how the CAD design was realised

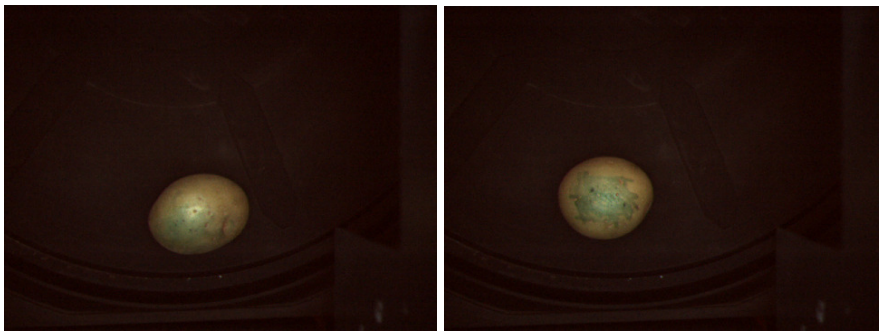
The mechanical functioning alone does not necessary mean successful operation, so the image the equipment produces must be evaluated.

Good-quality workable images were taken by the system as observed, in the products in Figure 3-7 as well as the produce in Figure 3-8 and Figure 3-10 or as seen in Figure 6-2 below, which indicate a successful hardware set-up.

(a) **(b)** **(c)**



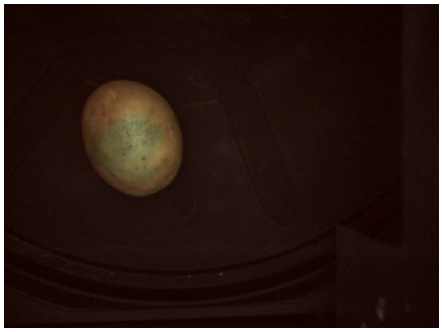
(d) **(e)**



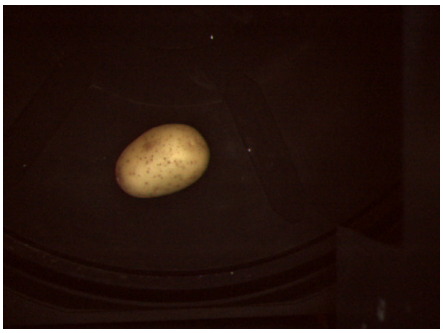
(f) **(g)**

(h) **(i)**

(j)

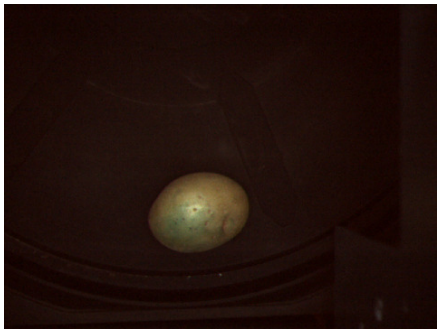


(k)

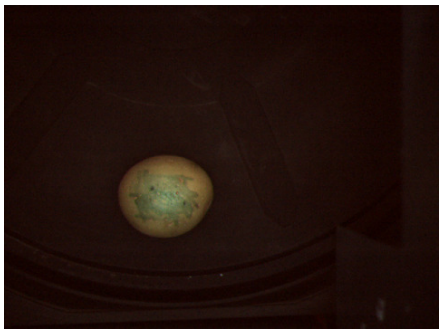


(l)

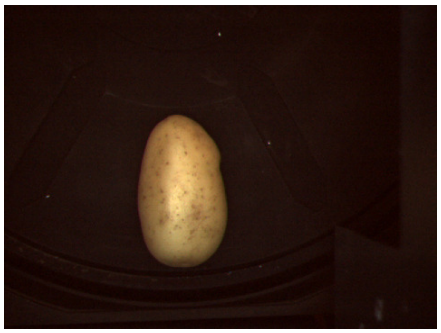
(m)



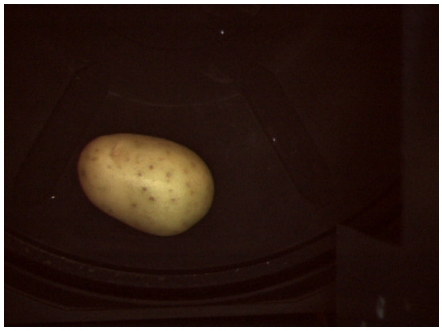
(n)



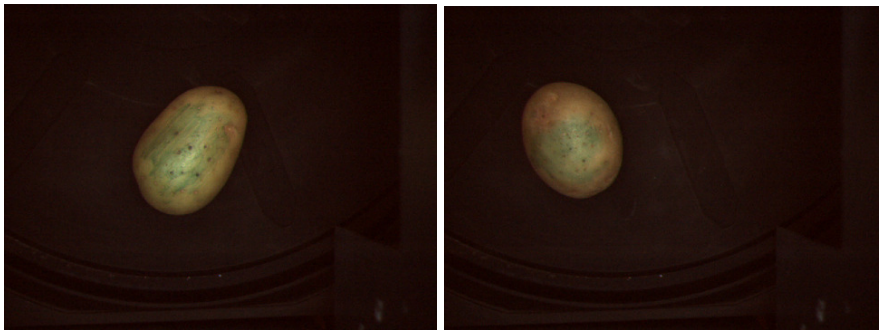
(o)



(p)



(q)



(r) **(s)**

Figure 6-2: Good-quality workable images as taken by the system

6.2 Intelligent software

The intelligent software section can be divided into the image extraction and the neural network decision-making parts.

6.2.1 Image extraction

To evaluate the image extraction component, the simulated results in section 4.1 can be compared with the results in section 5.1 as shown in the following:

6.2.1.1 Binary image (Thresholding)

Observe Figure 5-6 or Figure 6-2 below:

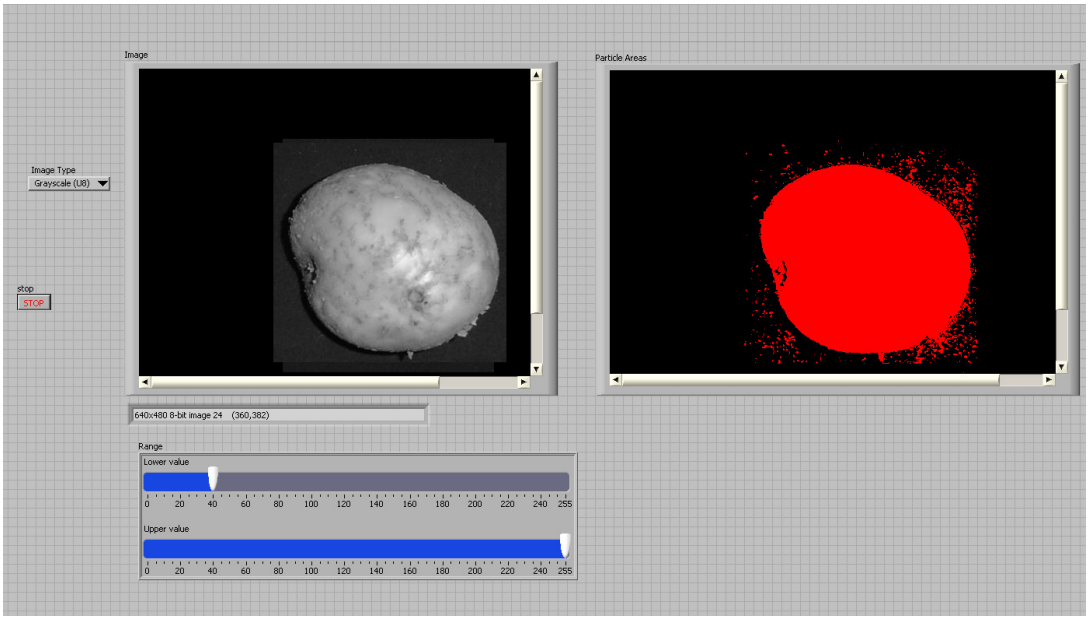


Figure 6-3: The view of user interface to the implementation code

The two images represent the read-in greyscale image (on the left) and the threshold image (on the right). In this example, the values were set to the same as those of the simulation.

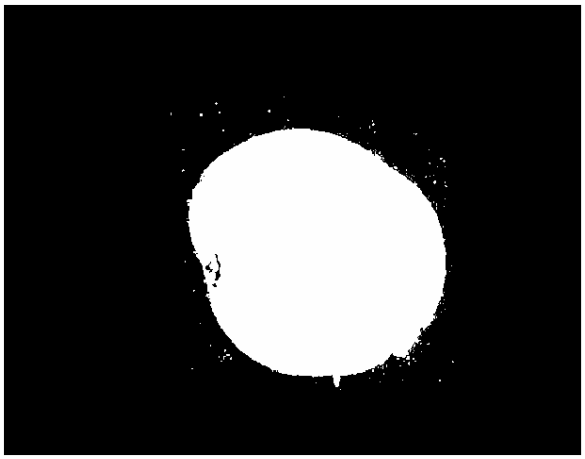


Figure 6-4: The binary image

Note the good correlation between the simulation image in Figure 4-3 or Figure 6-4 above, and the result of the implemented image in Figure 5-6 or Figure 6-3. The simulation was thus successfully implemented.

6.2.1.2 Erosion

Observe Figure 5-9 or Figure 6-5 below.

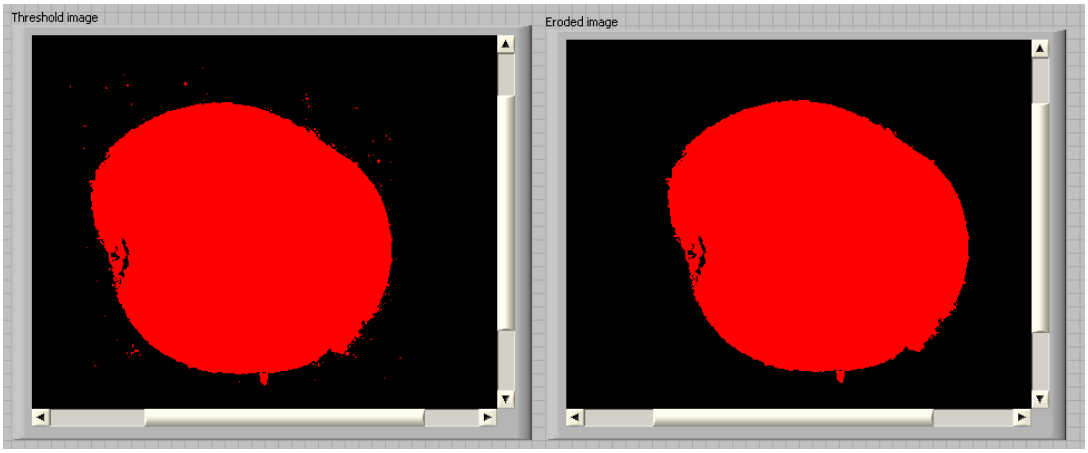
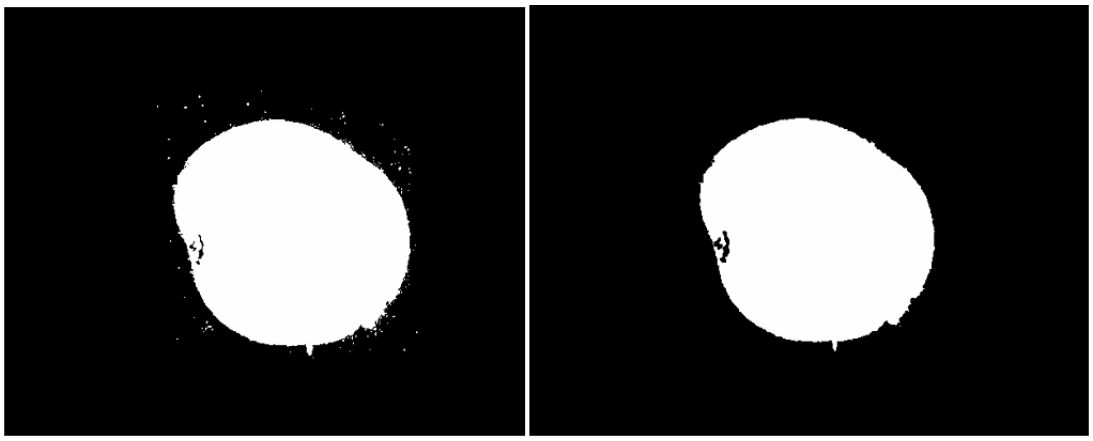


Figure 6-5: The user interface to the implementation code showing the result of the erode function

Note how the small spectral dots have been removed when the threshold image is compared with the eroded image. Note the good visual correlation between this and the simulation in Figure 4-4 or Figure 6-6 below.



(b) **(b)**

Figure 6-6: The erosion result (b) of the binary image of (a)

The simulation was thus successfully implemented.

6.2.1.3 Fill

Observe Figure 5-12 or Figure 6-7 below.

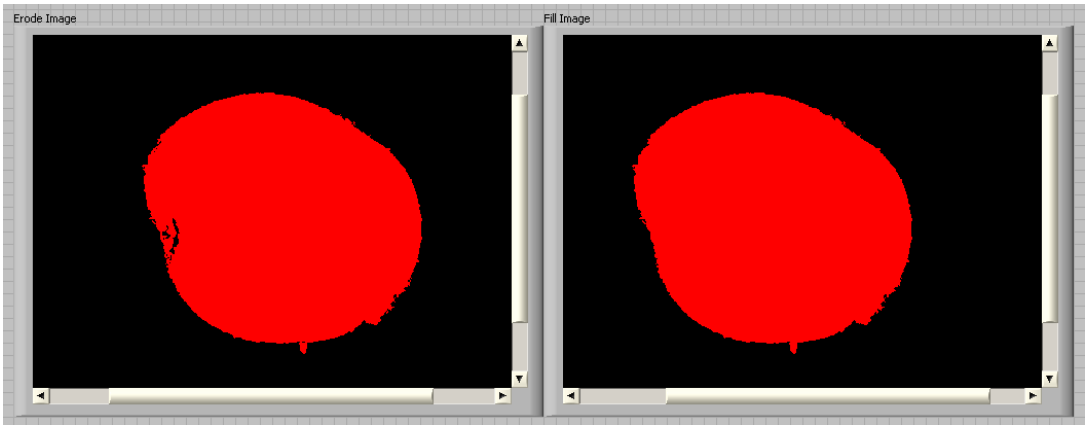


Figure 6-7: A section of the user interface of the implementation code showing the result of the fill function

Note how the holes in the object have been filled when the eroded image is compared with the filled image. The good visual correlation between this and Figure 4-5 in the simulation in section 4.1.3 or Figure 6-8..

(a) **(b)**

Figure 6-8: The filled result (b) of the binary image of (a) as in simulation

This indicates successful implementation

6.2.1.4 Area

The results of this function implemented were the same as those produced by the simulation for the same objects. This indicates successful implementation.

6.2.1.5 Shape

Observe the implementation result in Figure 5-24 or in Figure 6-9 below.

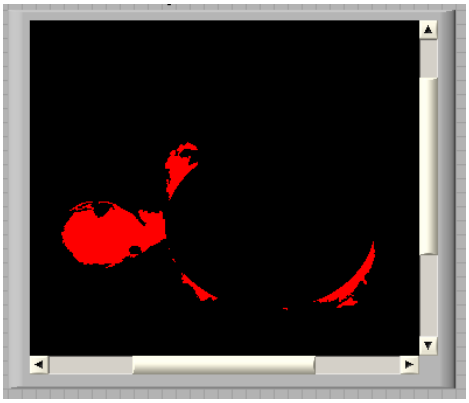


Figure 6-9: A section of the user interface of the implementation code showing the result of the masking function

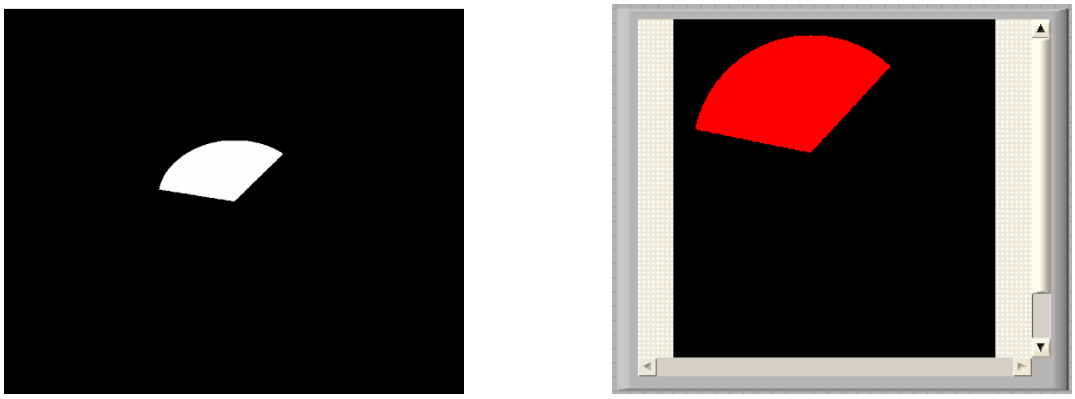
Note how only the parts lying outside the bounding oval are displayed. The good visual correlation between this and Figure 4-17 in the simulation in section 4.1.5.4 or Figure 6-10 indicates success.

Figure 6-10: The oval mask applied to the object image in simulation

6.2.1.6 Colour analyses

The colour is divided into three sections with a linear relation to each other. The green section consists of the hues between yellowish-green and greenish-blue.

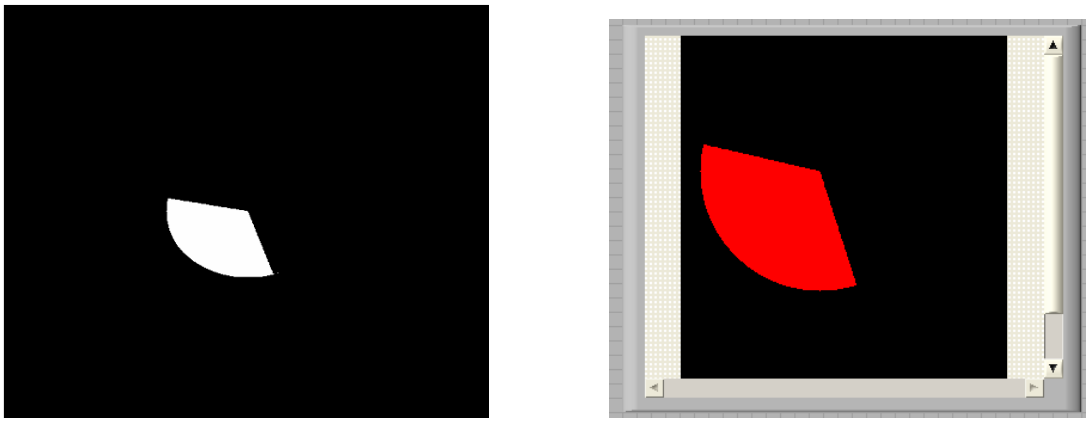
Note the good correlation of this function in Figure 5-32 and the simulation in Figure 4-20 or combined as below in Figure 6-11.



(a) (b)

Figure 6-11: The area of green (from yellowish-green to greenish-blue) for simulation (a) and implementation (b)

The blue section consists of the hues between blue-green and violet. Note this extraction in Figure 5-34 and the good correlation between this and the simulation in Figure 4-22 or combined in Figure 6-12 below.



(a) **(b)**

Figure 6-12: The area of blue (from blue-green to violet) for simulation (a) and implementation in (b)

The last part is red, which consists of the hues between violet and reddish-yellow. The result of this is given in Figure 5-37, and the good correlation between this

result and the result in Figure 4-26, or combined in Figure 6-13 indicates implementation success.

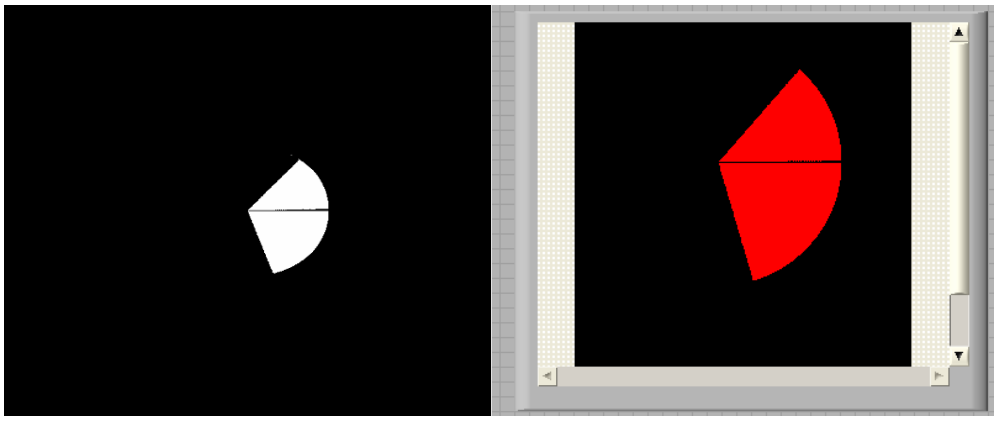


Figure 6-13: The area of red (from violet to reddish-yellow)

6.2.2 Neural networks

The neural networks implementation code is evaluated in section 5.3.2.4 by feeding it the same inputs and constants as the mathematical simulation in 4.2.2. The results produced are the same. It can therefore be stated that the implementation of the simulation was successful.

6.3 Artificial intelligence machine vision grading system

The complete system consists of three parts. The first part generates the training data. It consists of the feature extraction modules and produces the training data set spreadsheet. The second part is the one hundred input to four neurons to another four neurons with a four output neural network that learn this data set by adapting its weights. After a successful training stage, these weights are saved for the final stage. The last stage or operational stage consists of the feature extraction part with the forward sweep part of the neural network, of which the outputs are connected to the sorting hardware (pneumatic actuators). The neural network reads its weights in, as generated by the previous step. It also has a training set generator part that can add to the training set data spreadsheet if required. Separately all the individual modules produced good correlations between expected, simulated and real-time coded results. It is, however, still important to evaluate the complete system with real products and produce in real-time. This is done by three experiments. First, products in the form of plastic building blocks are sorted. The system is also evaluated to see if it can be retrained from sorting manufactured products to sorting natural produce. Secondly, potatoes must be sorted as an example of fresh produce. This demonstrates the ability to sort difficult natural objects. Lastly, an experiment with oranges demonstrates that the system is capable of sorting and grading several types of natural objects. It is also for the speed at which it can learn a new function. For example, how long does it take to adapt from sorting and grading potatoes to sorting and grading oranges when the season changes?

6.3.1 Building blocks

Figure 3-7 shows an example of the data from a training set of building blocks. The data were fed to the part that generates the training spreadsheet. The spreadsheet was randomised and fed to the training part. The training part was able to learn the data repeatedly under 5 minutes. This was re-evaluated due to the fact that each time this section started, the weight constants were newly generated random numbers that produced different learning times. The newly adapted weights were read into the operational part. Some of the results are given in Figure 6-1.

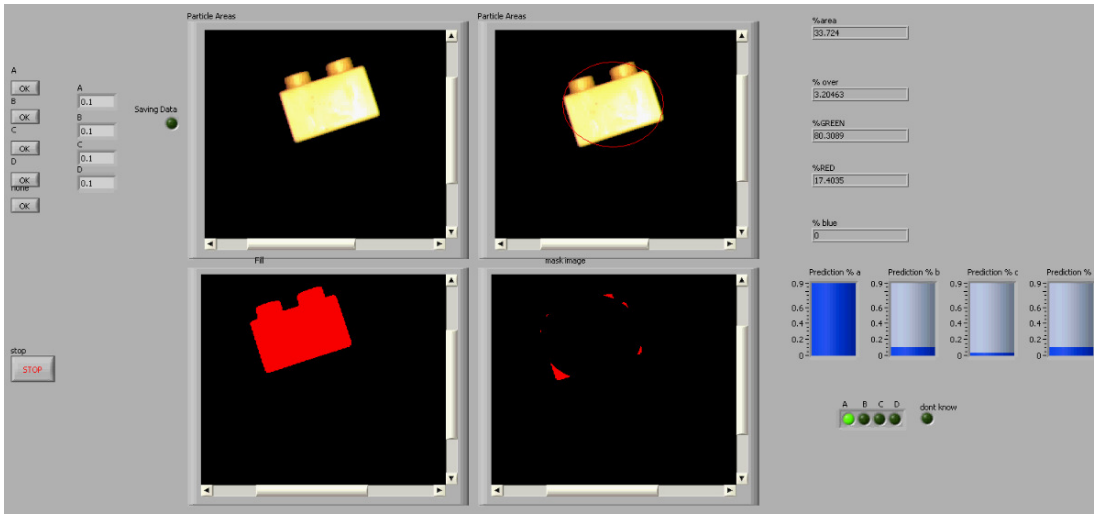


Figure 6-14: The artificial intelligence machine vision grading system front panel with a yellow block under the camera sorting it to option A

Figure 6-1 shows the final operation code. On the left are the buttons with their displays and the saving data indicator next to them. This is for the user to add to the training data if there is a sample that cannot be decided upon. Under this is the stop button that stops the software. In the centre are the four images. The one

at the top left shows the acquired sample. The one at the top right is the image sample with the bounding oval and mask. The one at the bottom left is the binary image used for masking an area. The image to the right of this shows the area of the object that is outside the bounding oval. The data on the right show, from the top, the percentage area, the percentage shape, the percentage green, the percentage red and lastly the percentage blue. The four tanks below give the output of the neural network and show the probability for options A, B, C or D. The Boolean on the bottom right-hand corner indicates the option to be taken or what actuator would be activated namely A, B, C, D or “don’t know”, which will let the object through to the manual sorters. At this point the sample can be made part of the training cycle with the switches on the left. Observe that the trained system sorts the yellow to option A.

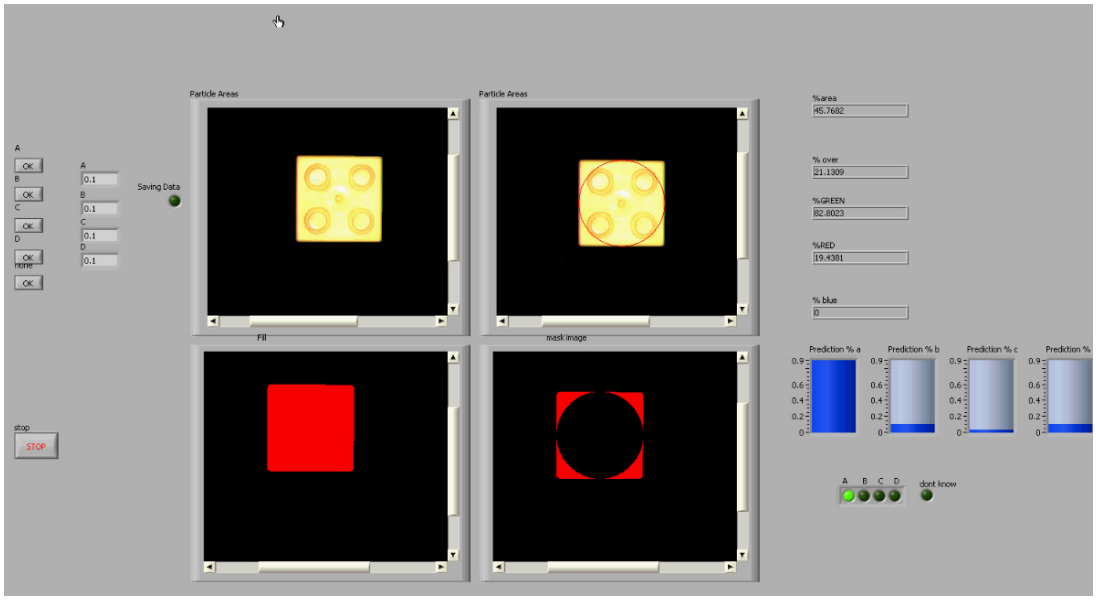


Figure 6-15: The artificial intelligence machine vision grading system front panel with a yellow block under the camera sorting it to option A

Observe that even though the orientation differs from that in Figure 6-14 the system still opts for option A as it is trained to sort yellow to option A.

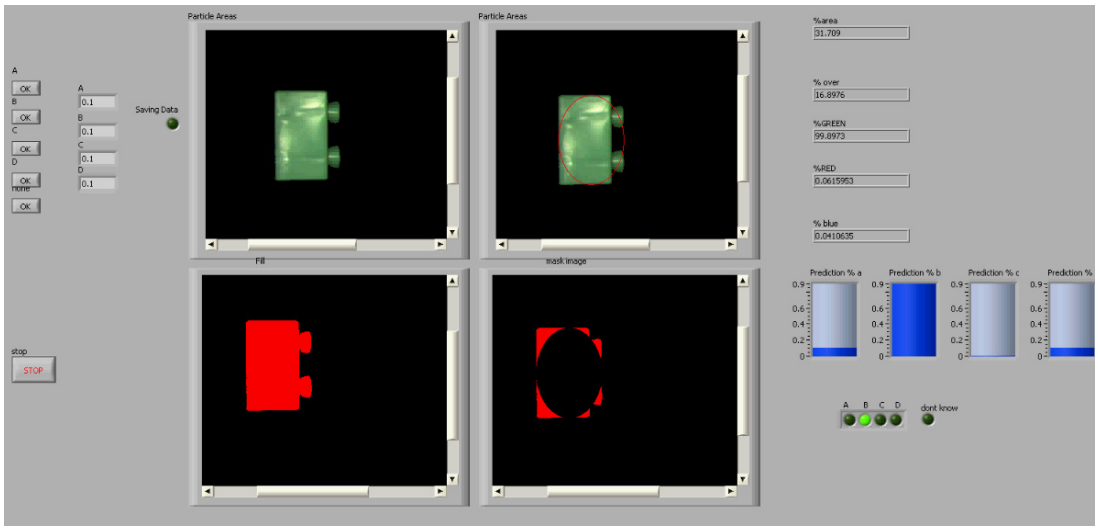


Figure 6-16: The artificial intelligence machine vision grading system front panel with a green block under the camera sorting it to option B

Observe that although the shape is the same as that in Figure 6-14. the colour is different (green, not yellow). Therefore the system opts for option B as it has been trained to do. Note that orientation is of no importance.

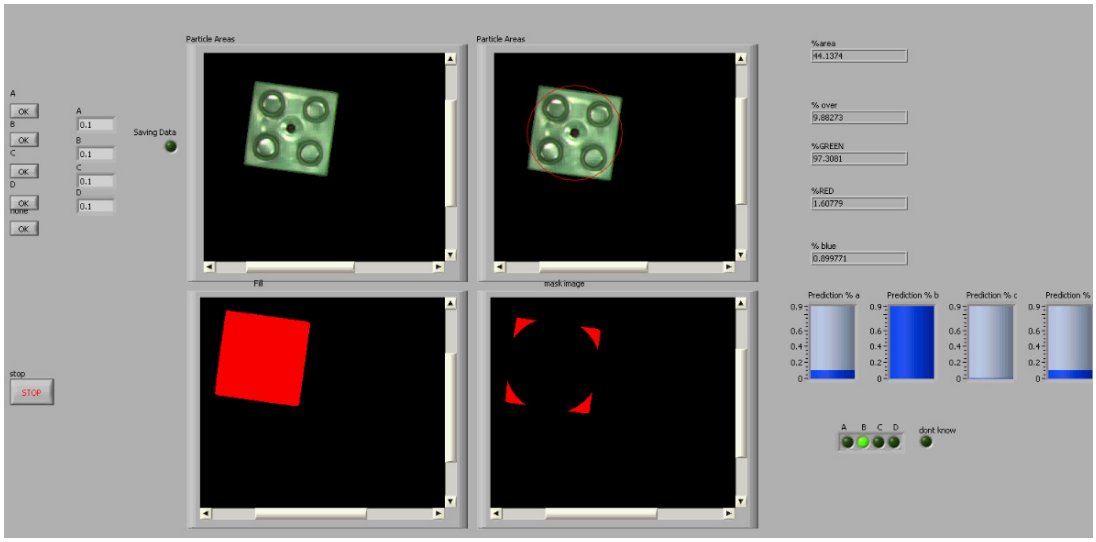


Figure 6-17: The artificial intelligence machine vision grading system front panel with a green block under the camera sorting it to option B

Figure 6-4 shows how another green block with a different orientation is sorted to B.

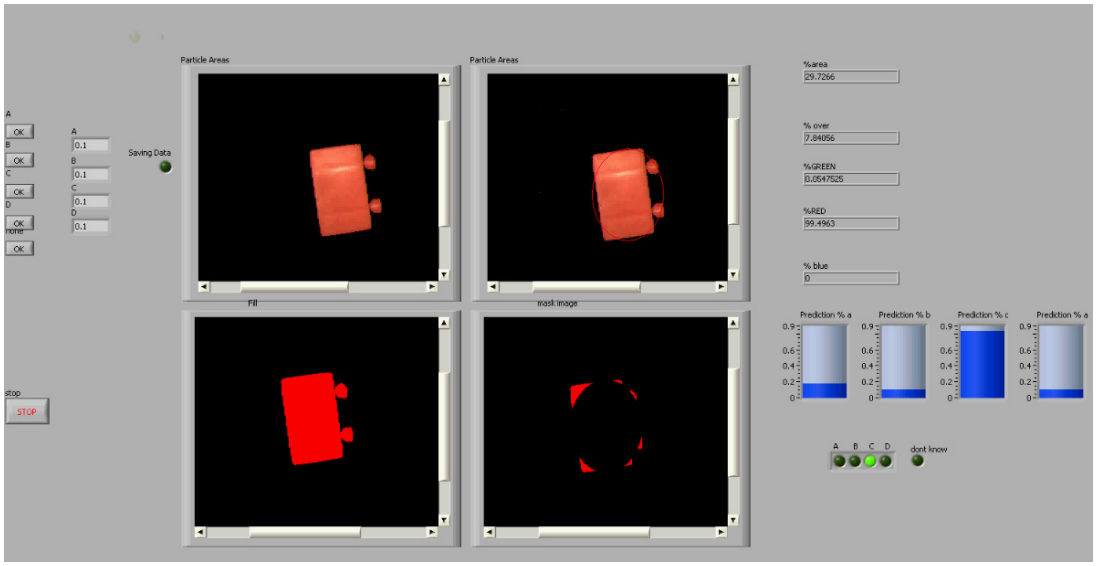


Figure 6-18: The artificial intelligence machine vision grading system front panel with a red block under the camera sorting it to option C

Figure 6-5 shows how the neural network predicts option C for a red sample as it has been trained to do.

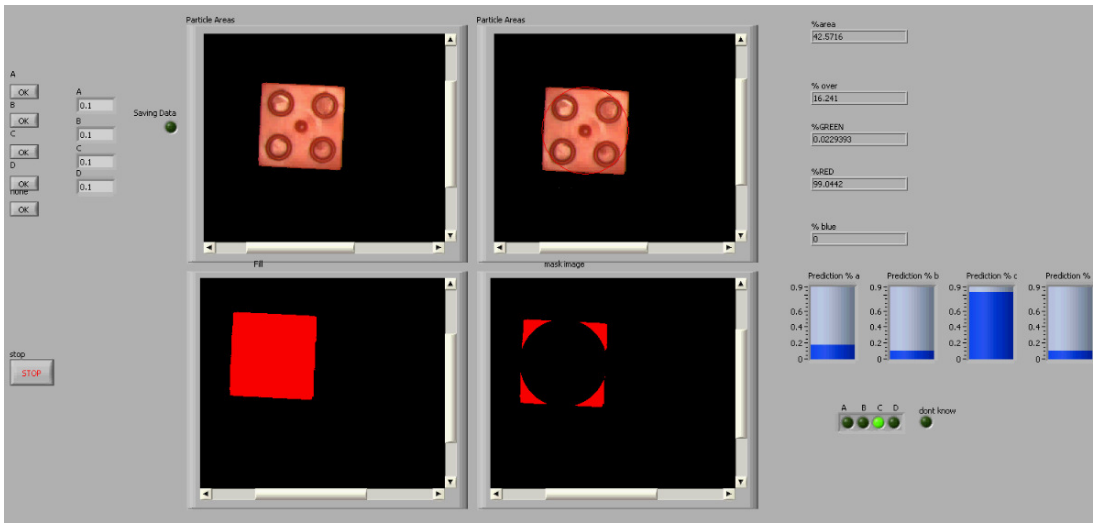


Figure 6-19: The artificial intelligence machine vision grading system front panel with a red block under the camera sorting it to option C

Figure 6-6 shows how even with a different orientation this red sample is also sorted to C.

These results show that the system can successfully sort these products, and if there is a sample that is not recognisable, it can easily be added to the training data for better use in future.

6.3.2 Potatoes

Figure 3-8 shows an example of the training data set of potatoes. As before, the data are fed to the part that generates the training spreadsheet. This spreadsheet was randomised and fed to the training part. As before, the training part was able to learn the data repeatedly under 5 minutes. The newly adapted weights were read into the operational part. Some of the results are given below.

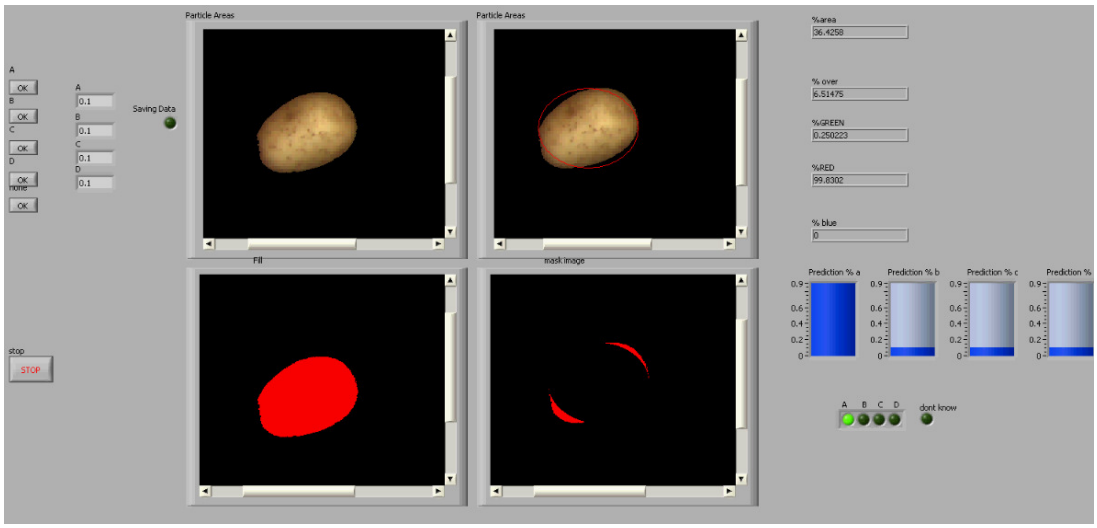


Figure 6-20: The artificial intelligence machine vision grading system front panel with a small potato under the camera sorting it to option A

This of course is exactly the same software used in the plastic block sorter. The only difference is new weights for the neural network. As before, the buttons are on the left with their displays and the saving data indicators are next to them.

Under this is the stop button that stops the software. In the centre are the four images. The one at the top left shows the acquired sample. The one next to it at the top is the image sample with the bounding oval and mask. The one at the bottom left is the binary image used for masking an area. The image to the right shows the area of the object that is outside the bounding oval. The data on the right show, from the top, the percentage area, the percentage shape, the percentage green, the percentage red and lastly the percentage blue. The four tanks below are the output of the neural network and show the probability of options A, B, C or D. The Boolean at the bottom right-hand corner indicates what option to take or what actuator would be activated, namely A, B, C, D or “don’t know”, which will let the object through to the manual sorters. Observe that the trained system grades small potatoes to option A.

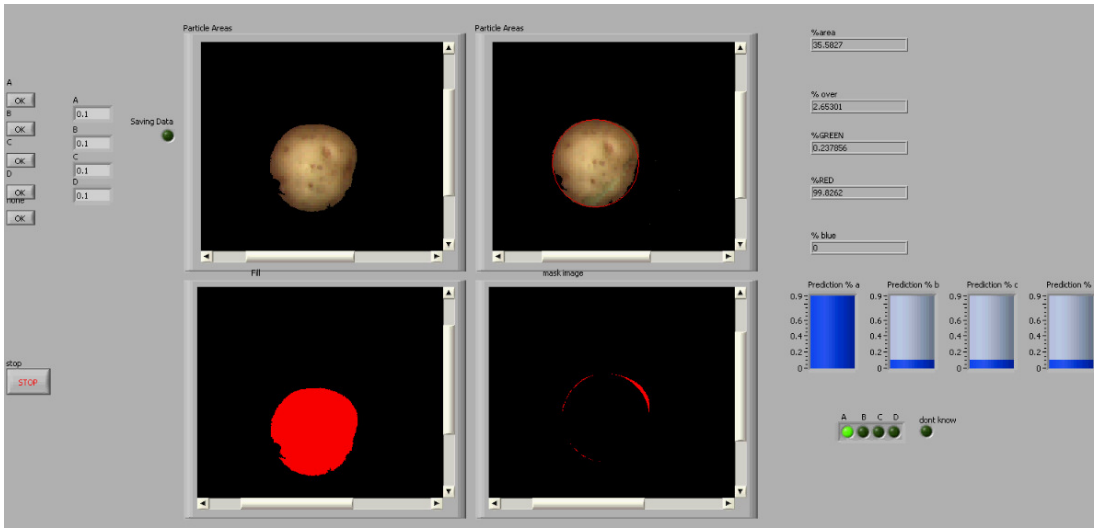


Figure 6-21: The artificial intelligence machine vision grading system front panel with a small potato under the camera sorting it to option A

As stated, fresh produce is difficult to sort using machine vision. Potatoes of the same class may not look the same. Figure 6-8 shows that even though the potato

looks different to the one in Figure 6-20, will still be sorted to A as the system has been trained.

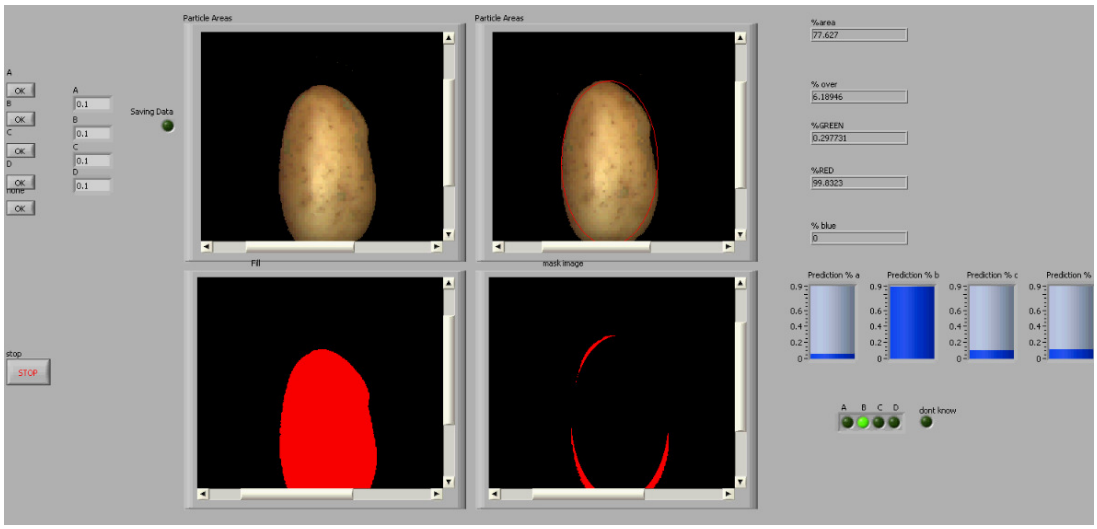


Figure 6-22: The artificial intelligence machine vision grading system front panel with a large potato under the camera sorting it to option B

Figure 6-9 shows how a larger potato would be sorted to option B.

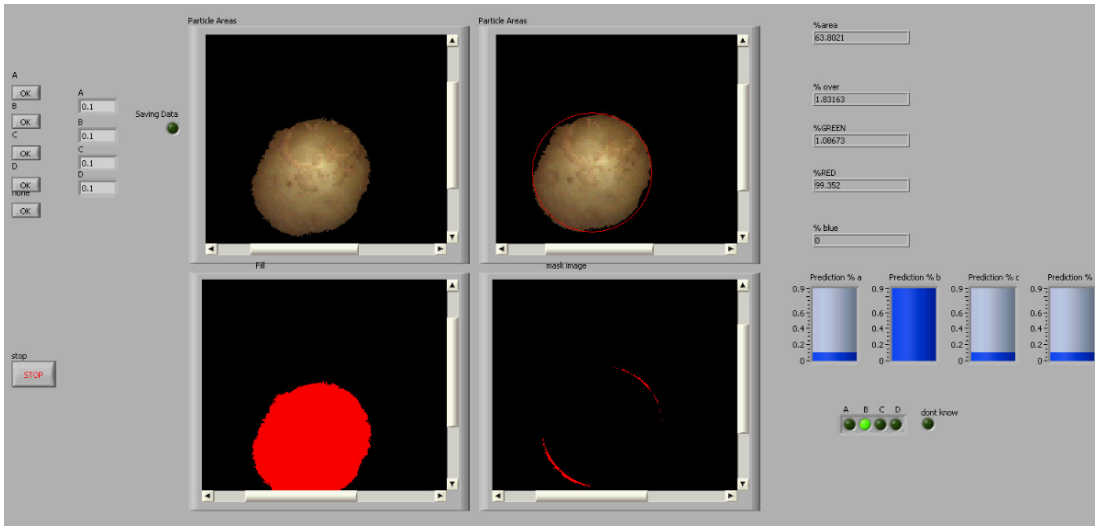


Figure 6-23: The artificial intelligence machine vision grading system front panel with a large potato under the camera sorting it to option B

Figure 6-10 shows how a large potato with a different shape will also be sorted as required in option B.

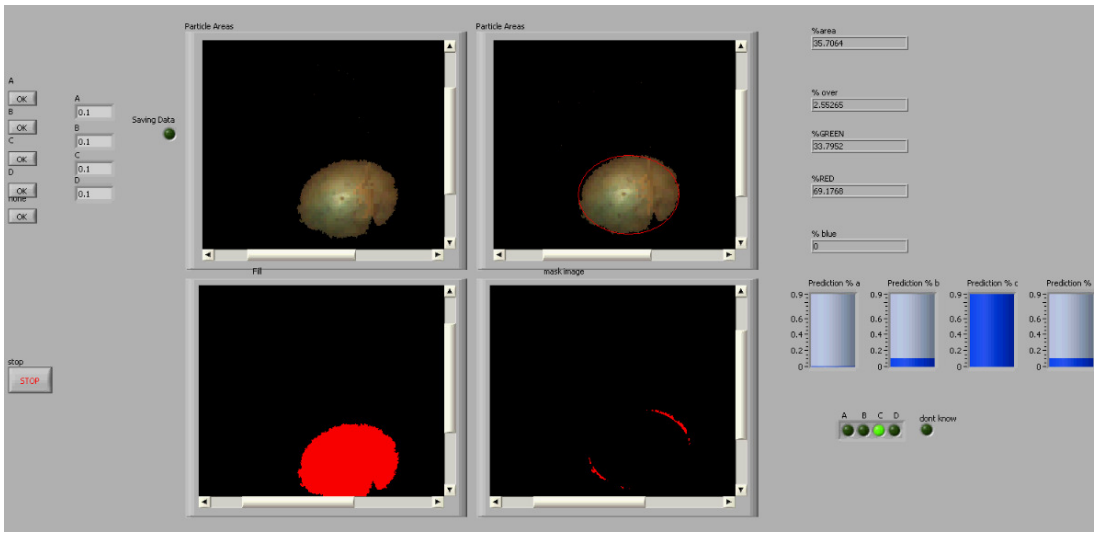


Figure 6-24: The artificial intelligence machine vision grading system front panel with a small green potato under the camera sorting it to option C

Figure 6-11 shows how a small potato that would have been sorted to A is now selected for C since it is too green.

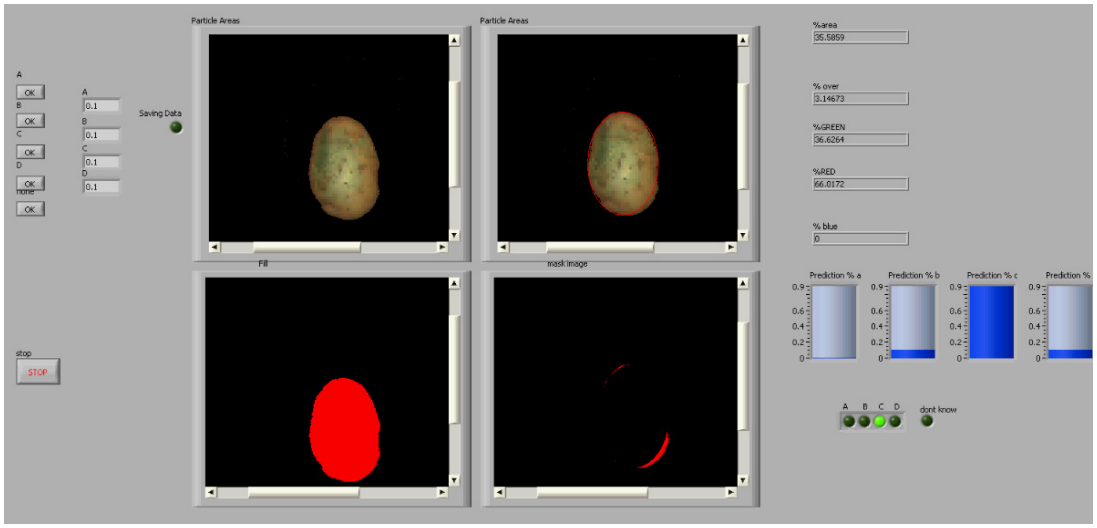


Figure 6-25: The artificial intelligence machine vision grading system front panel with a small green potato under the camera sorting it to option C

Figure 6-12 shows how a different shape small green potato is sent to C as required.

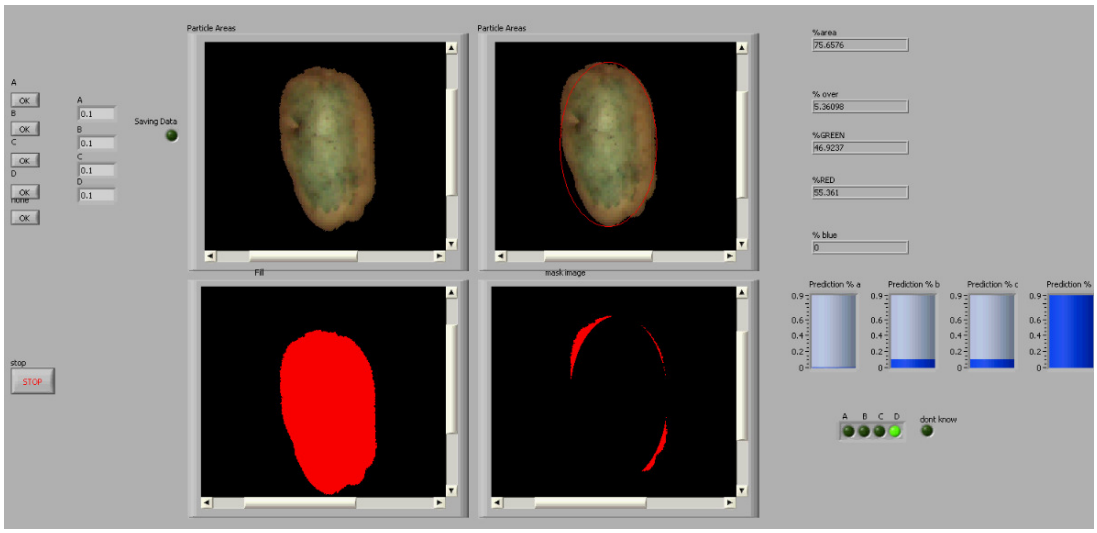


Figure 6-26: The artificial intelligence machine vision grading system front panel with a large green potato under the camera sorting it to option D

Figure 6-13 shows how a large potato that would have been sorted to B is now selected for D since it is too green.

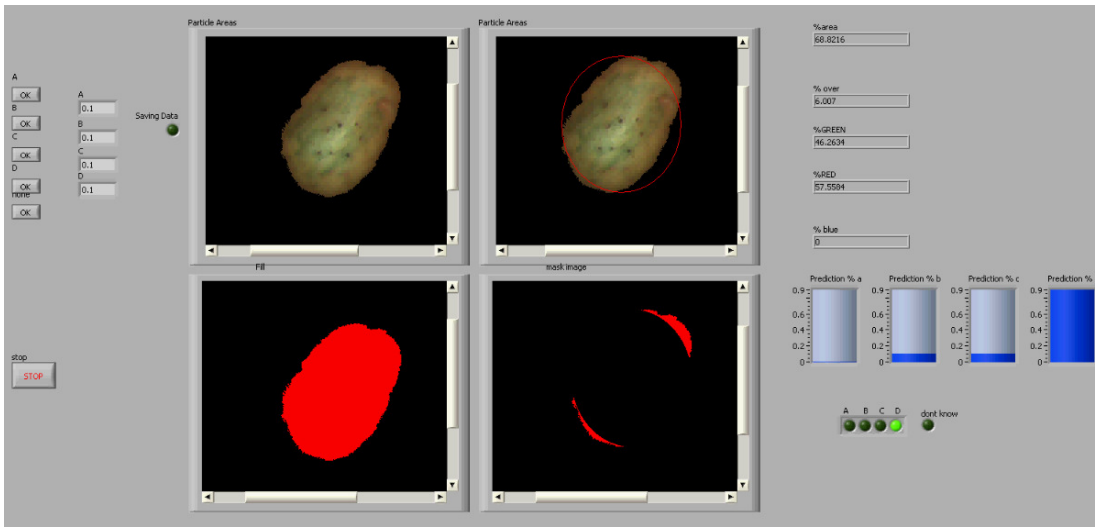


Figure 6-27: The artificial intelligence machine vision grading system front panel with a large green potato under the camera sorting it to option D

Figure 6-14 shows how a different shape large green potato is sent to D as required.

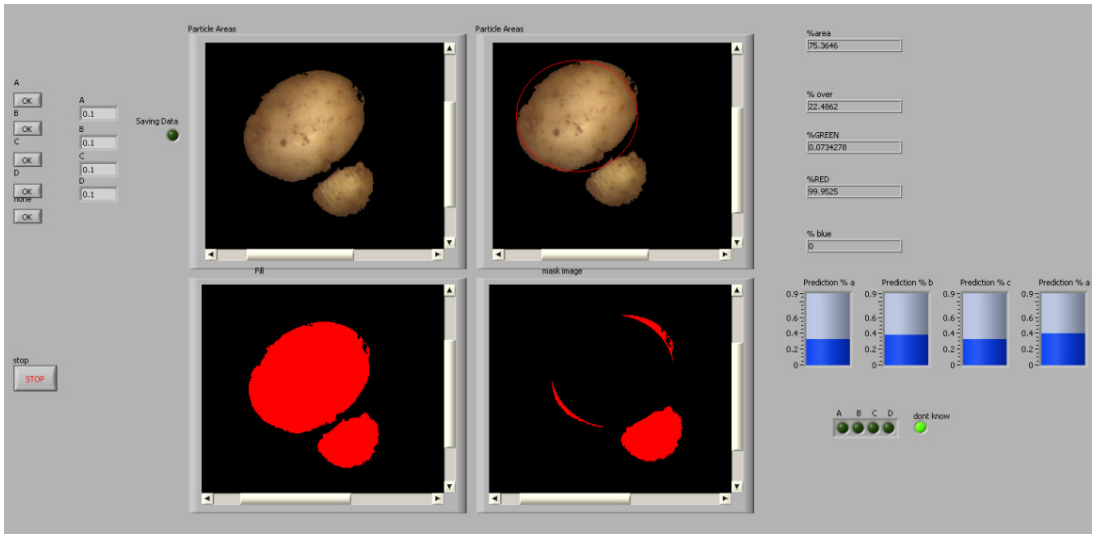


Figure 6-28: The artificial intelligence machine vision grading system front panel with a malformed potato under the camera sorting it to none of the options

Figure 6-15 shows how an irregularly shaped potato would be let through to the end of the conveyer by not removing it with actuator A, B, C or D as the system has been trained.

These good results indicate that this system is able to grade traditionally difficult fresh produce with ease. Again, if there is a sample that is undecided, it could be added to the training data, resulting in a system that is always improving and getting more “experience”.

6.3.3 Oranges

As stated earlier, it would be an advantage if this system could adapt to new produce in the new season. A new training set is given for example for oranges. See the example in Figure 3-10. As before, these are fed to the part that generates the training spreadsheet. This spreadsheet was randomised and fed to the training part. The training part was also able to learn the data under 5

minutes. The newly adapted weights were read into the operational part. Some of the results are given below.

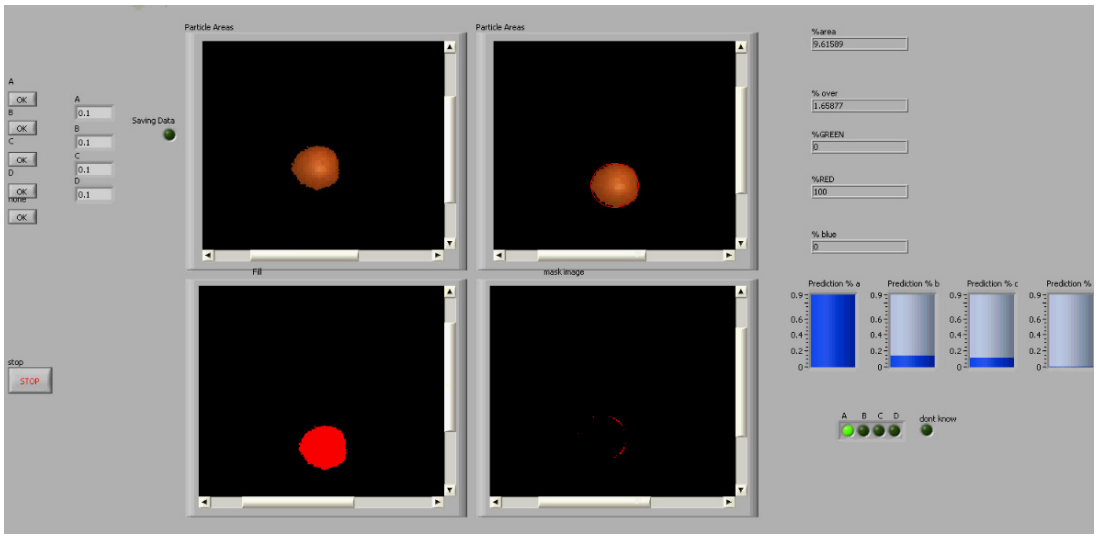


Figure 6-29: The artificial intelligence machine vision grading system front panel with a small orange under the camera sorting it to option A

Again this is the same software as used previously, but this time with weights that will sort, as seen in Figure 6-16, small oranges to A.

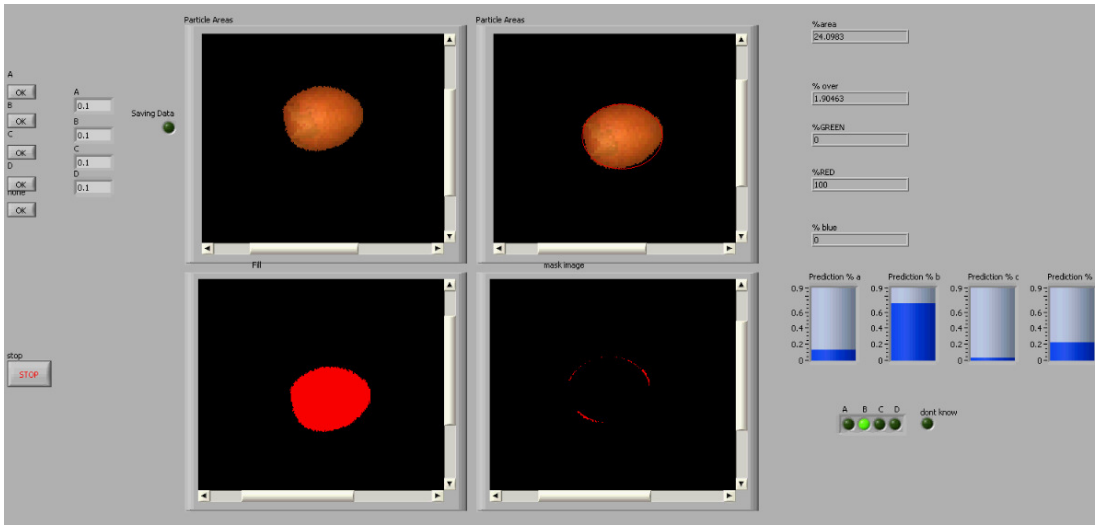


Figure 6-30: The artificial intelligence machine vision grading system front panel with a large orange under the camera sorting it to option B

Figure 6-16 shows how the neural network will sort larger oranges to B.

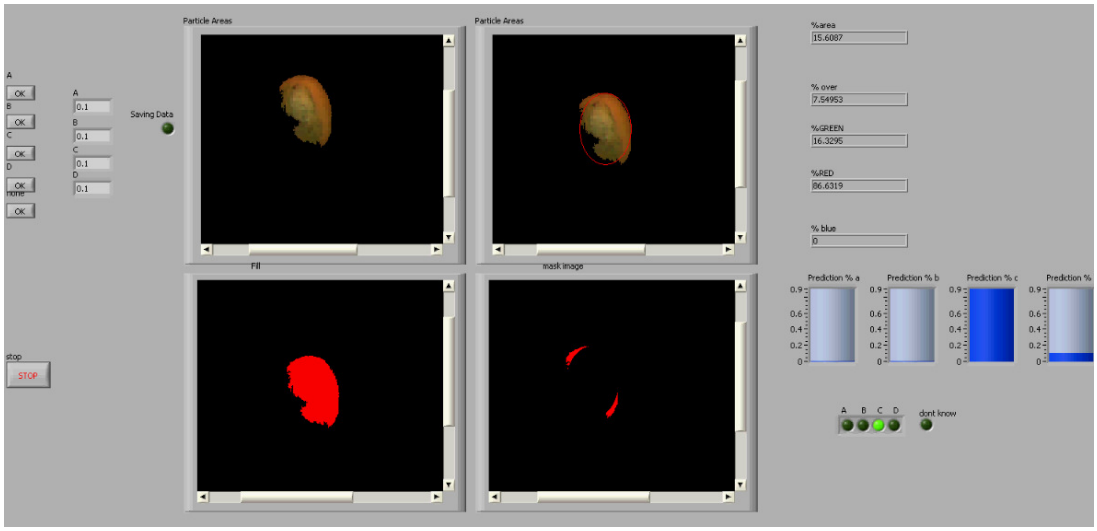


Figure 6-31: The artificial intelligence machine vision grading system front panel with a small green orange under the camera sorting it to option C

Figure 6-18 shows a small green orange being sorted to C

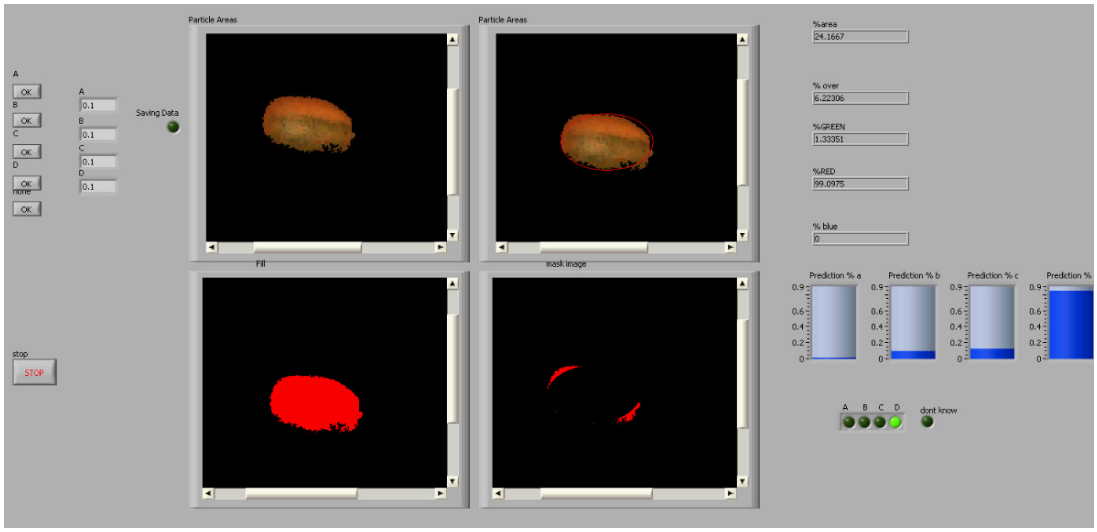


Figure 6-32: The artificial intelligence machine vision grading system front panel with a large green orange under the camera sorting it to option D

Figure 6-19 shows a large green orange being sorted to D.

These good results show that this system can be retrained to sort products or different types of produce in a short space of time.

7 CONCLUSION

The aim of this study was to design an artificial intelligence machine vision system that can be taught as a human is taught. This system should also be able to make predictions (like a human), with the ability to grade unique agricultural and biological products. To accomplish this, the neural network is based on the algorithms of a human brain, but not all is done with the network. By not using the network for all the functions, this system outperforms similar previous attempts. As advanced as computers have become, they are still based on single operations, whereas a biological brain has the ability to do true parallel calculations. Observe a neural network. In a biological brain all the neurons produce their results continually and simultaneously, whereas the simulation of this in a computer functions quite differently. In a computer, the first neuron is calculated and the answer is saved to be used at another time. Then the next neuron is calculated, and the result is saved, and the next until the first layer is completed. After this, the saved values are read in from memory one at a time, and one function is calculated at one neuron at a time. This makes it much slower than a human. A computer, on the other hand, is better than a human at executing raw mathematical calculations. When designing an artificial intelligence machine vision system, one should bear all this in mind. In some of the previous inefficient attempts, the designers connected every single camera pixel to an input of a neural network. This means that not only does the network have a lot of weights that need to be adapted (and saved), but a larger network is also needed. This is due to the fact that a more intelligent system must learn “how to see” first before it can learn how to evaluate. Seeing is a complex function which we

humans take for granted. Consider all the variables such as: what is the background and what is the object, what is the colour, is the object small and close or large and distant? There are also some basic data that one does not even bother to conceptualise until one tries to teach a computer to do so.What sets this system apart is that all the mathematical processing is done as far as possible, making the neural network only applicable to the decision part, thus limiting its inputs. But because the system has a neural network, it can carry out visual evaluations like a human does. It is able to do this efficiently since computer architecture was taken into consideration in the design. This system is thus able to grade manufactured products - note its ability to sort the plastic block regardless of the orientation. After this it was successfully retrained to sort and grade produce. The produce samples, potatoes, were graded according to size and colour. The system also did predictive sorting and grading on unknown potato samples, which is required for such a system due to variations in the same class. After this it was retrained again to sort and grade oranges, where it was also able to do predictive control. This artificial intelligence machine vision system is thus able to sort and grade products and produce of any kind or variant just by showing it some samples and what to do with them, just as a human can.

REFERENCES

1. Bishop, C. M. Neural Networks for Pattern Recognition. Oxford: Oxford University Press, 2005.
2. Bloomberg, D. S. Implementation Efficiency of Binary Morphology. Proceedings of the International Symposium for Mathematical Morphology VI, Sydney, Australia, 3-5 April 2002, p. 8.
3. Bryce, D. M.. Plastic Injection Molding: Material Selection and Product Design Fundamentals. Kansas City: SME, .1997.
4. Callan, R. Artificial Intelligence. New York: Palgrave Macmillan, 2003.
5. Chang, J., Han, G., Valverde, J., Griswold, C. N., Duque-Carrillo, J. F. and Sáchez-Sinencio, E. Cork Quality Classification System using a Unified Image Processing and Fuzzy-Neural Network Methodology. IEEE Transactions on Neural Networks, vol. 8, no. 4, July 1997, p. 965.
6. Chauvin, Y. and Rumelhart, D. E. Backpropagation: Theory, Architectures, and Applications. Philadelphia,: Lawrence Erlbaum Associates, 1995.
7. Department of Agriculture. United States Standards for Grades of Peaches (Effective 21 May 2004). United States Department of Agriculture, Agricultural Marketing Service Fruit and Vegetable Programs, Fresh Products Branch, 2004.
8. Department of Agriculture. United States Standards for Grades of Strawberries (Effective 1 July 1965. Reprinted - January 1997). United States

Department of Agriculture, Agricultural Marketing Service Fruit and Vegetable Programs, Fresh Products Branch, 2004.

9. Deshmukh, K.S. and Shinde, G. N. 2005. An Adaptive Colour Image Segmentation. Electronic Letters on Computer Vision and Image Analysis, vol. 5, no. 4, 2005, pp. 12-23.

10. Driggers, R. G. *Encyclopaedia of Optical* Engineering. Boca Raton: CRC Press, 2003.

11. Farid, H. Detecting Hidden Messages Using Higher-Order Statistical Models. Proceedings of the 2002 International Conference on Image Processing (ICIP 2002). Rochester, New York, USA, 22-25 September 2002, vol. 2, IEEE, pp. 905-908.

12. Frey, H. Machine Vision Seminar. University of Applied Sciences, Ulm, Department of Computer Science. 2005.

13. Gurney, K. An Introduction to Neural Networks. Boca Raton: CRC Press, 2003.

14. Haralick, R. M. and Shapiro, L.G. 1992. Computer and Robot Vision Volume 1. Redwood City: Addison-Wesley Publishing Company Inc, 1992.

15. Hornberg, A. Handbook of Machine Vision. Wiley-VCH, 2006.

16. http://nina-photo.com/images/wood_in_hdr.jpg [1 June 2009]

17. http://vistasem.files.wordpress.com/2008/06/fly-eye-250x.jpg [1 June 2009]

18. http://www.dnr.sc.gov/ael/personals/pjpb/lecture/spectrum.gif [21 May 2009]

19. http://www.efunda.com/DesignStandards/plastic_design/warpage.cfm [21 May 2009]

20. http://www.ni.com/swf/presentation/us/imaq/ [21 May 2009]

21. http://www.pinholeresource.com/shop/components/com_ virtuemart/shop_image/product/Leonardo_Pinhole_47ca066317936.jpg [1 June 2009]

22. http://www.protomold.com/DesignGuidelines_UniformWallThickness.aspx [21 May 2009]

23. https://admin.acrobat.com/_a56821929/labview [21 May 2009]

24. Jacobson, R. E. and Ray, S. The Manual of Photography: Photographic and Digital Imaging. St. Louis: Elsevier, Focal Press, 2000.

25. Jähne, B. Digital Image Processing: Concepts, Algorithms, and Scientific Applications. New York: Springer, 2005.

26. Lee, T. and Lewicki, M. S. Unsupervised Image Classification, Segmentation and Enhancement Using ICA Mixture Models. IEEE Transactions On Image Processing, vol. 11, no. 3, March 2002, p. 270.

27. Mammone, R. J. Artificial Neural Networks for Speech and Vision. London: Chapman and Hall, 1994.

28. Naik, S. K. and Murthy C. A. Hue-Preserving Colour Image Enhancement without Gamut Problem. IEEE Transactions on Image Processing, vol. 12, no. 12, December 2003, p. 1591.

29. Navulur, K. Multispectral Image Analysis Using the Object-oriented Paradigm. Boca Raton: CRC Press, 2006.

30. Rao, P. S., Gopal, A. S., Revathy, R. and Meenakshi, K. Classification of Fruits Based on Shape Using Image Processing Techniques. J. Instrum. Soc. India, vol. 34, no. 4, 2002, pp. 227-239.

31. Rao, P. S., Gopal, A. S., Revathy, R. and Meenakshi, K. Colour Analysis of Fruits Using Machine Vision System for Automatic Sorting and Grading. J. Instrum. Soc. India, vol. 34, no. 4, 2002, pp. 284-291.

32. Russ, J. C. The Image Processing Handbook. Boca Raton: CRC Press, 2007.

33. Saxby, G. The Science of Imaging: An Introduction. Boca Raton: CRC Press, 2002.

34. Snyder, W. E. and Hairong, Q. I. Machine Vision. Cambridge: Cambridge University Press, 2004.

35. Spitzer, M. The Mind within the Net: Models of Learning, Thinking, and Acting. Cambridge: MIT, 2000.

36. Steger, C. Ulrich, M. and Wiedemann, C. Machine Vision Algorithms and Applications. Weinheim: Wiley-VCH, 2007.

37. Wilkinson, M. E. Doan, A. P. Essential Optics Review for the Boards. Iowa: MedRounds Publications, 2006.

38. Zhoi, L., Chalana, V. and Kim, Y. PC-Based Machine Vision System for Real-Time Computer-Aided Potato Inspection. International Journal of Imaging System Technology, vol. 9, 1998.

39. Zuech, N. Current Machine Vision Activities in the Food Industry. Machine Vision online. 2007 Available at :

http://www.machinevisiononline.org/public/articles/
archivedetails.cfm?id=3144 [21 May 2009]

40. Zuech, N. Machine Vision in the Food Industry. Machine Vision Online. 2005 Available at:
http://www.machinevisiononline.org/public/articles/archivedetails.cfm?id=790 [21 May 2009]

41. Zuech, N. Understanding and Applying Machine Vision. Boca Raton: CRC Press, 2000.