UNIVERSITY of LIMERICK

OLLSCOIL LUIMNIGH

Department of Electronic & Computer Engineering

Melanoma (Skin Cancer) Long-Term Monitoring

Name:	Ciaran Carroll
ID Number:	13113259
Supervisor(s):	Dr Sean McGrath/Dr Colin Flanagan
Course:	Electronic and Computer Engineering (LM118)
Academic Year:	2017/2018
Project Number:	SMcG/CF 1

Contents

Contents 2
Introduction & Product outline 5
Changes to the original plan 6
Product List 7
Software Used 7
Literature Survey 8
Background Theory & Principles of Operation 9
Melanoma (Skin Cancer) Background 9
Image Processing & Machine Vision Background 11
Image segmentation using the Watershed Algorithm 13
Feature Extraction 15
Image Moments 16
Assessing the shape of mole using the contour 17
Design Implementation 20
Hardware and Sensor Development 20
Image Processing Design 23
Image Segmentation design 24
Feature Extraction design 25
Design Testing & Evaluation 27
Hardware and Sensor Development 27
Image Processing Development & Evaluation 28
Image Segmentation – Watershed algorithm 28
Feature Extraction Development 31
Comparing the features of the mole 34
Implementation – Image Processing in Python 36
Image segmentation using the Watershed algorithm 37
Feature Extraction 42
Conclusion 46
Acknowledgements 47
References 48
Appendices 50

Table of figures

Figure 1 - Process of Otsu thresholding - Courtesy of 'Scipy lecture notes' [8] 11
Figure 2 - Erosion and dilation of a simple binary image [10] 12
Figure 3 – Illustration of the watershed technique [12] 13
Figure 4 – Basic 9 by 9-pixel grid 18
Figure 5 - Test example with the centre point and extreme points shown 19
Figure 6 – Hardware and sensor interaction diagram 20
Figure 7 - Raspberry Pi GPIO Pin Layout [14] 21
Figure 8 - Ultrasonic sensor circuit diagram [13] 22
Figure 9 - Testing initial functionality of the system 27
Figure 10 - Stages of the Image Segmentation process 29
Figure 11 - Watershed marker image 30
Figure 12 - Output from the program 33

Abstract

Melanoma is one of the fastest growing skin cancers in Ireland, with over 1,000 people diagnosed each year. It is a particularly dangerous form of skin cancer as it often spreads to other parts of the body beyond the skin where it can become hazardous and difficult to treat. Recently, huge improvements have been made in the diagnosis and treatment of the disease using Image Processing which is possible as the malignant moles associated are visible on the skin.

With the rise in the development of Machine Vision medical applications aimed to diagnosed Melanoma in its early stages, many medical firms have identified the potential that Machine Vision based products has on Melanoma detection and treatment. The overall aim of this project is to demonstrate the capabilities that Machine Vision applications has in a medical context.

Specifically, we will be designing a system to facilitate the monitoring process of Melanoma or other types of skin cancer. The usefulness of such a product could have huge benefits & implications for society and the medical sector at large, particularly for patients in measuring any changes in their malignant moles.

Due to the unpredictable nature of Melanoma, and the importance of monitoring its progress, the aiding of said monitoring process will lead to numerous benefits. These benefits include reducing the cost, skill level and knowledge required to monitor melanoma, and a dramatic improvement in the treatment and outcome of diagnosed patients.

Introduction & Product outline

The general purpose of this project is to develop a system to facilitate healthcare professionals and patient's in the monitoring process of Melanoma or other types of skin cancer. This system will involve integrating aspects of hardware and software engineering in its design to build a functional prototype, which can provide a viable method of measuring the progress of a malignant mole on a patient's arm. This project could prove to be an invaluable tool for healthcare professionals in the treatment of skin cancer and improve the outcome of patients.

To develop the system, it makes sense to build and test the software and hardware components separately before being combined during the final stages of the project. These components include:

- Hardware and Sensor Design & Integration
- Machine Vision Implementation for Image processing & Feature extraction
- Software Implementation for Monitoring Melanoma

The overall aim of this project is to design and develop a working prototype of a product to monitor Melanoma, while at the same time increasing my knowledge of the various concepts used to develop the product. This involves learning how to implement Image Processing techniques by using the Python library OpenCV and designing an adequate environment to apply these techniques effectively.

The hardware and sensor development aspect of the project will require combining a camera module with a Raspberry Pi and an ultrasonic sensor to measure the distance of the imaged mole from the camera. A condescend light and a white sheet of paper as a backdrop will also be added to deliver uniform lighting and an uncomplicated background which will ensure consistency in the images.

The software aspect of the project will require combining various Image Processing techniques such as Preprocessing, Segmentation and Feature Extraction. Preprocessing and Segmenting the image will remove the unwanted regions such as the background skin and hairs while highlighting the mole as the focus of the image. Feature Extraction methods will then be used to obtain the useful information from the segmented image. For this project, the feature extraction methods will be based on the ABCDE mnemonic which is commonly used

by dermatologists to classify a malignant mole. Combining these techniques to compare and analyse images of a malignant mole at various stages of its development will provide healthcare professionals and patients with a method to monitor any changes in the mole's appearance.

The Melanoma monitoring method will involve creating a Python script on a Raspberry Pi to take an image of a mole with a camera module, applying various Image Processing techniques on the image taken to separate the mole from the skin and background and saving the new image of the mole on the Raspberry Pi. A combination of Image Processing techniques and mathematical manipulations will then be used to extract information about the features of the mole. Another image is taken of the mole in the same environment and processed with the same Image Processing steps. The features of both images are compared forming the basis for the melanoma monitoring method. Any changes in the mole's appearance will be noticeable and measured.

Changes to the original plan

My initial plan for this project was to implement the Image processing and feature extraction components in a custom-built enclosure which was to provide the environment to capture the images of the suspicious moles on a patient's arm. The enclosure was to be fitted with multiple infrared-sensitive cameras to obtain a full representation of all the moles on a patient's arm by combining the images. A Raspberry Pi connected to the enclosure would provide a method for capturing and storing the images on the system as well as the implementation of the software required for monitoring the Melanoma.

However, I encountered many issues when attempting to build an enclosure with multiple infrared cameras in a feasible way. Mainly, I was having difficulty connecting multiple cameras to a Raspberry Pi and aligning those cameras to minimise the overlap in the images. So, I decided to alter the project to incorporate a normal camera module to capture the images of the moles and I would design a suitable environment to capture the images without using an enclosure. This will allow me to focus on a single suspicious mole and monitor its development as the cancer progresses.

Product List

- Raspberry Pi 3 – Model B
- Raspberry Pi Camera Module – V2
- SanDisk 16GB SD Card with NOOBS pre-installed
- Raspberry Pi Power Supply – 13W, 5.1V, 2.5A
- HC-SR04 – Ultrasonic distance sensor
- 3 – 1 kΩ Resisters
- 4 – Male to female jumper wires

Software Used

Python3.5.2 – Anaconda3.5.0.1

Python is a high level, object-oriented programming language which is used for a variety of purposes. [1] It has an elegant syntax that is easy to code in. Anaconda is an open-source distribution for the Python language which simplifies package management and deployment. It has a large number of open-source libraries built into the distribution making it easy to expand programs by importing new moduleS. Python also has many volunteers constantly improving the language and open-source libraries.

In this project, we will be using Python for implementing the image processing aspects of the project. There is a large amount of online support for Python in this specific application, and many of the standard Python libraries have built-in functions which will be used. Python also comes as standard with the Raspbian operating system for the Raspberry Pi.

OpenCV 3.4

OpenCV (Open Source Computer Vision Library) is a library for computational efficiency with a strong focus on real-time image processing applications. The library is cross-platform and was released under the open-source BSD License so it's free to use for both academic and commercial use. [2] It was initially developed by Intel before becoming maintained as open-source software by Itseez.

Most of the image processing techniques used in this project will be implemented through the OpenCV library.

Literature Survey

Automatic Detection of Malignant Melanoma using Macroscopic Images [3]

In this paper, three Biomedical Engineering students developed a method for detecting malignant melanoma from benign pigmented lesions using macroscopic images. The Image Processing steps necessary to achieve this were Preprocessing, Image Segmentation, Feature Extraction, and Classification. The students acquired their images for training their Machine Learning algorithm from the dermatology atlases Dermnet where they collected 282 RGB images of various sizes (149 benign and 133 malignant). Preprocessing of the images involved using a median filter and morphological operators to remove impact noise, skin lines, hairs, and reflections. The RGB channels (Red, Green and Blue Channels) of each image was also converted to HSV colour space (Hue, Saturation, and Value) to weaken the effect of non-uniform lighting. There were many methods used to segment the lesion from the skin such as K-mean clustering and Otsu Thresholding to remove unwanted parts of the image. The feature extraction method used for this project was based on the ABCD criteria which is similar to the traditional process of visually inspecting the mole. Finally, a Support Vector Machine classifier predicted whether a lesion was benign or malignant.

Smartphone Medical Toolkit [4]

In this project, an Electronic and Computer Engineering student developed an Android application that integrates with a Raspberry Pi connected with an infrared camera to map the veins on a person's hand to assist with the process of injection. The student created an LED circuit to emit infrared light at a certain wavelength to avail of the light-absorption properties of veins in the body. An optical filter with a narrow band-pass was also placed over the infrared camera to ensure that only light within the band of interest would be observed. A Python script was created on the Raspberry Pi to take an image with the infrared camera and save the image on the system. The script also applied various image processing to the image taken mainly a type of histogram equalisation known as 'CLACHE' performed. The student created an Android application within Android Studio, with the graphical UI and layout of the application designed in Sketch and the application communicated with the Raspberry Pi using the 'ssh' protocol. By combining the different aspects, the student proved that a viable product like this could be created to help the injection process using smartphones.

Although neither of these projects directly monitored Melanoma, I learned a lot from both of them and took inspiration when designing the system for my project.

Background Theory & Principles of Operation

Throughout this section, the impact of Melanoma in Ireland will be outlined. This includes an overview of Melanoma, potential risk factors for being diagnosed, preventative measures available in reducing its likelihood and the treatment options for the disease.

Also, the principles for which the Image processing and Feature extraction techniques used in this project during the Machine Vision Implementation will be explained. While it may not be possible to explain all the intricacies of each technique used in the project, a brief introduction into these concepts should provide enough of an understanding into how they operate and highlight the importance for each technique in the project. The Image segmentation and feature extraction techniques will be described separately to help with the understanding of each section and its relevance in the project.

Melanoma (Skin Cancer) Background

Skin cancer is the most common cancer in Ireland, with 10,304 cases of non-melanoma skin cancer and 1,041 cases of melanoma skin cancer diagnosed in 2014. [5] It is a mostly preventable cancer, and early detection of it will drastically improve the outcome of the patient as the treatment will be relatively easy and cheap. Skin cancer is the abnormal, uncontrollable growth of skin cells which is often developed when the genetic material (DNA) inside the skin cells is damaged which causes the cells to multiply rapidly usually because of overexposure to ultraviolet (UV) radiation either from sunlight or sunbeds. There are two main types of skin cancer: Non-melanoma and Melanoma. Non-melanoma skin cancer is a group of skin cancers that affect the upper layers of skin which mostly comprise of basal cell carcinoma and squamous whereas Melanoma skin cancer is a group of cancers that start in pigment cells (melanocytes) in the skin.

Melanoma is the deadliest form of skin cancer. Although it only accounts for 3-4% of all skin cancers, it is responsible for 75% of all skin cancer deaths and 1 in 8 people who are diagnosed with it result in a fatality. It is particularly dangerous as it can spread to other parts of the body beyond the skin usually through the lymphatic system and when it does, it can become hazardous and difficult to treat.

A person is more at risk for Melanoma if they have pale skin that burns easily, many moles, uses tanning beds regularly or has a family history of melanoma. If you have a higher risk of developing skin cancer, it is best to take extra precautions. The best way to prevent

melanoma is to avoid overexposure to the sun, use SPF 30 sunscreen when necessary, avoid sunbeds and have regular skin checks.

In Ireland, suspicious moles are first examined by a GP who usually decides if the moles require further testing by a dermatologist who is a doctor that specialises in skin conditions. [6] A dermatologist will use the ABCD method alongside a questionnaire to determine the severity of the mole. The ABCD method for visually inspecting a mole will assess if the mole is asymmetrical, has an unusual border, multiple colours and if the mole is large. Often if the mole looks unlike any other mole on the patient's body further inspection might be needed. The questionnaire will involve asking the patient if the mole has recently appeared, is changing over time or if the mole has caused any irritability problems such as itchiness or bleeding. A dermatologist may decide from this information to do a biopsy if they think it is necessary which is where a suspect mole is removed from your skin so that it can be studied under a microscope to check if the mole is cancerous. If the mole is confirmed to be cancerous, a further operation may be done to remove a wider margin of skin around the mole which may remove the melanoma.

Treatment of melanoma will depend on how far the melanoma has grown into the skin and whether it has spread to other regions of the body. In the early stages if the cancer is detected treatment will involve surgically removing the cancerous mole with a small area of the skin surrounding it which is known as surgical excision. If the melanoma has been removed, there is little chance of it returning, and no further treatment should be needed. In the case that the melanoma has spread to another part of the body it may not be possible to cure it so measures may be taken to slow the growth, reduce the symptoms and possibly extend the life expectancy of the patient. Some of these include Radiotherapy and drug treatments such as Chemotherapy to name a few which can have serious side effects on the patient, so it's better to treat it early to avoid these treatments. It is important that melanoma is detected and treated before it spreads as there are better treatments and higher survival rates.

Skin cancer is well suited for image processing as the abnormal moles are visible on the skin. There are many advances in the field of Machine Vision which could dramatically change how we detect and treat Melanoma in the area of detection and treatment. Computer programs could potentially be developed to provide a non-invasive method for determining if a mole is benign or malignant.

Image Processing & Machine Vision Background

Otsu Thresholding

Otsu's method is a global thresholding technique proposed by Nobuyuki Otsu in 1979, which has become a popular algorithm for converting a grayscale image to a binary image. Otsu's method assumes that the image contains two classes of pixels (foreground and background) following bi-modal histogram. This method aims to find the optimum threshold value separating the two classes to minimise their combined spread (intra-class variance) hence separating the greyscale image into a binary image. This is best seen graphically below.

For instance, assume an image has L grey levels and the total pixels with grey level I: n_i.

$$Total\ pixels\ N = n_0 + n_1 + n_2 + \dots n_i + \dots n_{L-1}$$

$$Probability\ that\ a\ pixel\ has\ grey\ level\ i: p_i = \frac{n_i}{N}$$

$$where: p_i \geqslant 0\ and \sum_{i=0}^{L-1} p_i = 1$$

$$Intra-Class\ Variance\ between\ two\ classes\ is\ given\ as: {\sigma_w}^2 = P_1 {\sigma_1}^2 + P_2 {\sigma_2}^2$$

$$where\ Variances: \sigma_1^2 = \frac{1}{P_1} \sum_{i=0}^{k} p_i (i - \mu_1)^2\ and\ \sigma_2^2 = \frac{1}{P_2} \sum_{i=k+1}^{L-1} p_i (i - \mu_2)^2$$

$$Class\ Means: \mu_1 = \frac{1}{P_1} \sum_{i=0}^{k} i\, p_i\ and\ \mu_2 = \frac{1}{P_2} \sum_{i=k+1}^{L-1} i\, p_i$$

$$Class\ Probability: P_1 = \sum_{i=0}^{k} p_i\ and\ P_2 = \sum_{i=k+1}^{L-1} p_i$$

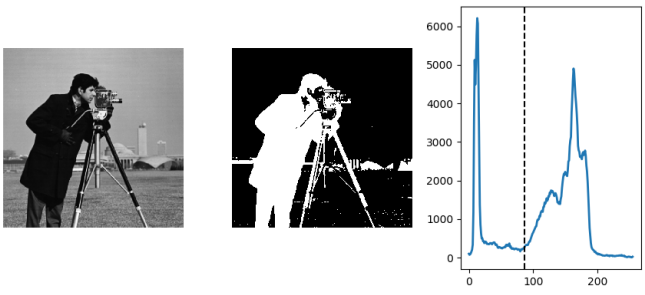


Figure 1 - Process of Otsu thresholding - Courtesy of 'Scipy lecture notes' [8]

Morphological Operations

Morphological operations are a useful set of operations that are commonly applied to binary images to process them based on shapes. [9] The main aim of morphological operations is to apply a structuring element (or kernel) which decide the nature of the operation to an input image to generate a unique output image. Various morphological operations are used throughout this project such as Erosion, Dilation and Closing.

Erosion is a basic operation on binary (or grayscale) images that erodes away the boundaries of foreground pixels (i.e. white pixels) based on the structuring element applied. Any holes within the background become larger. Dilation is the opposite to erosion, it increases the size of the foreground pixels (i.e. white pixels). Any holes in the image shrink whereas the boundaries of the image expand. This can be seen in the below diagram.

Opening and Closing operators are derived from erosion and dilation. Opening is erosion followed by dilation using the same structuring element. It is used to remove noise from an image. Closing in contrast is dilation followed by erosion using the same structuring element. It is used to close small black points in an image.

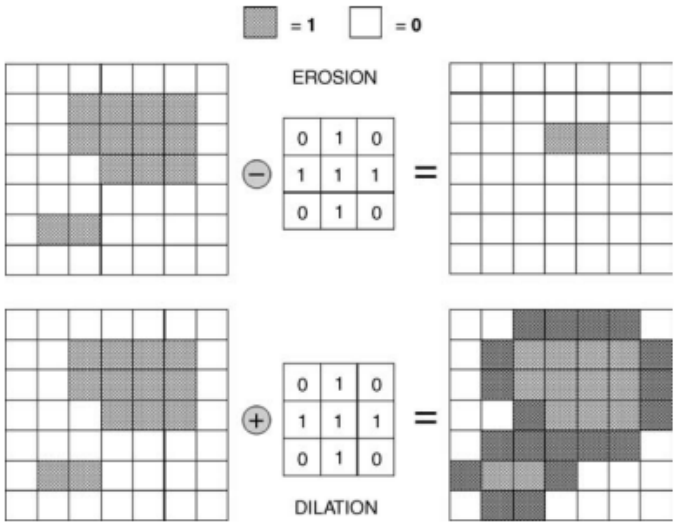


Figure 2 - Erosion and dilation of a simple binary image [10]

Image segmentation using the Watershed Algorithm

Image Segmentation is used to partition an image into meaningful regions to improve its analysis. It is done by separating the objects within the image by finding the boundary regions around the objects and removing the unwanted regions. The main aim for image segmentation in this project is to remove the skin and background from the image to create a new image containing only the mole of interest.

The Watershed algorithm is a specific type of marker-based image segmentation introduced by Serge Beucher and Christian Lantuéj in 1979, which has become a classic method for segmenting an image. It is based on an analogy of rain filling low-lying regions in a landscape.

Any grayscale image can be viewed as a topographic surface where high intensity denotes peaks and hills while low intensities denotes valleys. [11] The topographic intensities can be calculated using Distance transform as a measure of depth/altitude. When the image has been converted to a 'height' image, the landscape can be filled with water. Every isolated valley (local minima) detected by the algorithm will be labelled with a different coloured water. As the water rises, depending on the peaks (gradients) nearby, the different coloured waters will start to merge. To avoid overlapping, barriers are built in the locations where the water merges. This process of filling water and building barriers will be continued until all the peaks are underwater. The barriers created from this process should give us the boundaries of the regions in the image. The regions of the image are labelled during the watershed, with the background of the image labelled 1, and the other objects that appear labelled with integers starting with 2. This allows us to alter the image using the labels to create a new image containing only the region of interest which is the mole under inspection in our case.

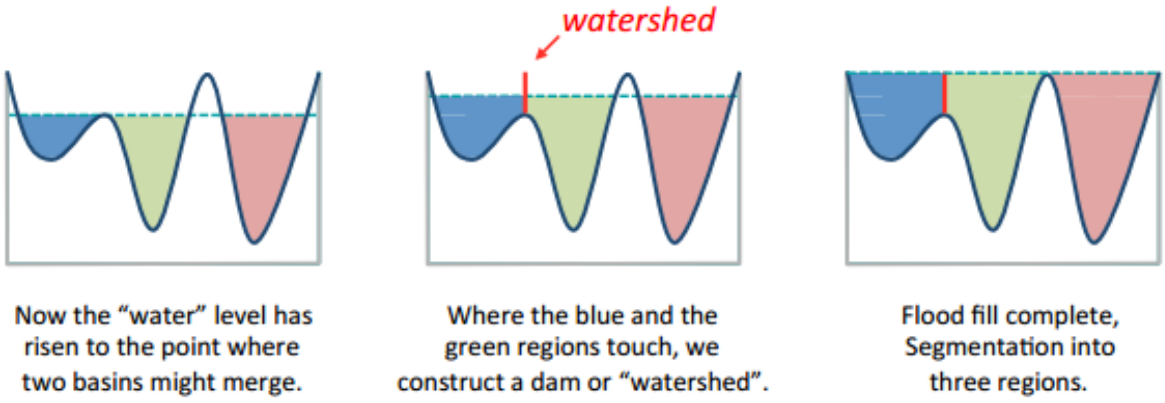


Figure 3 – Illustration of the watershed technique [12]

The Watershed algorithm can be comprised of nine steps, which are executed in the following order:

(1) Convert the image to grayscale so that it can be viewed as a topographic image.
(2) Threshold the grayscale image using Otsu Thresholding to output a binary image.
(3) Use Morphological closing to remove noise and reduce the effects of hairs on the image.
(4) Use Morphological dilation to increase the boundary by 1 pixel to increase the regions threshold to find areas that are surely background.
(5) Apply the Euclidean Distance Transform to the binary image to output a 'height' image to find areas that are surely foreground.
(6) Find the unknown region by subtracting the sure foreground from the sure background.
(7) Find local maxima of the height image and assign unique labels to each maxima, this is the seed label image.
(8) Build the barriers (dams) to stop the regions from merging.
(9) Now apply watershed routine to the depth (= -height) image, using the seed labels to start the regions.

It can be difficult to imagine how an image can be separated into different regions using this method so the analogy of rain filling a landscape is useful for explaining the algorithm in a human-friendly way.

In this method, tiny imperfections in the binary image could have a huge effect on the watershed technique and cause over-segmentation. Morphological closing is used to prevent this by 'filling in' small holes in the foreground of the image, without changing the structure of the rest of the image too much.

The marker-based watershed algorithm used in this project was implemented using OpenCV, with an interactive segmentation as it is possible to specify which valley points are to be merged and which aren't. OpenCV also handles all the operations needed to achieve a desirable segmentation.

Feature Extraction

Feature Extraction is the process of transforming an input image into features of distinct properties for analysis. Its main purpose for this project is to extract useful information from the segmented image gathered from the watershed algorithm to improve human interpretation of said images. The features gathered from the images of the mole of interest will be compared with different images of the same mole under the same conditions at a later stage of the cancers development forming the basis of our method for monitoring Melanoma. By comparing the features of the moles in each image we can identify if the mole is changing or evolving which is important as Melanoma can become difficult to treat when it spreads.

The feature extraction methods used in this project will be based on the ABCD method (Asymmetry, Border Irregularity, Colour Variation and Diameter) which is an approach commonly used by GPs (General Practitioner) and Dermatologists to assess the severity of a suspicious mole. This method involves inspecting if a mole has an irregular border or is asymmetrical, has variation in the colour across the mole and a diameter greater than 6mm to determine the likely the mole is cancerous or identify the risk associated with a malignant mole. There are many useful Image processing techniques available that can be applied to the segmented image of the mole to obtain information about its features.

To apply the feature extraction methods required to analyse the mole, we must convert the segmented image gathered from the watershed algorithm into a binary image using Otsu Thresholding. The threshold image should show the mole in white pixels and the rest of the image in black pixels. From here, we can use the OpenCV function ‘cv2.findContours’ to search for contours in the image, which refers to a closed curve. There should only be one contour found in the image which should contain the mole we are interested in analysing. We can analyse the contour with Image moments to calculates its pixel area, centre point, diameter in pixels, eccentricity and orientation. There is also a function in OpenCV capable of finding the four most extreme points in the contour. Using these extreme points and the centre point we can evaluate the general shape of the mole by calculating the distance, slope and angle between each of the points.

The main concepts behind the Image processing techniques mentioned will be explained below:

Image Moments

Image moments are the weighted average of the pixels intensities that can be used to describe the features of an image our in this case the contour which should represent the mole we are analysing. The moments of a contour represent its mass in pixels (area), the centre of mass (centroid) and rotational inertia (orientation, major and minor axis).

The raw image moment M_{ij} for the binary image with pixel intensities I(x, y) can be represented as:

$$M_{ij}=\sum_{x}\sum_{y} x^{i} y^{j} I(x,y)$$

The zeroth moment represent the area of the contour (mass). Its calculated by counting all the pixels in the contour.

$$M_{00}=\sum_{0}\sum_{0} x^{0} y^{0} I(x,y)$$

The first order moments represent the centre of mass or centroid of the contour and can be defined as:

$$\mu_{pq}=\sum_{x}\sum_{y} (x-\bar{x})^{p}(y-\bar{y})^{q} I(x,y)$$

where the centroid of the contour $(\bar{x}, \bar{y})$ is:

$$\bar{x}=\frac{M_{10}}{M_{00}} and \bar{y}=\frac{M_{01}}{M_{00}}$$

M_{10} and M_{01} are the sums over x and y respectively and M_{00} is the area.

The first and second order moments which are relevant to this project are calculated from:

$$\mu_{pq}=\sum_{m}^{p}\sum_{n}^{q}\binom{p}{m}\binom{q}{n}(-\bar{x})^{(p-m)}(-\bar{y})^{(q-n)} M_{mn}$$

or

$$\mu_{00}=M_{oo} \qquad \mu_{11}=M_{11}-\bar{x}M_{01}=M_{11}-\bar{y}M_{10}$$

$$\mu_{01}=0 \qquad \mu_{20}=M_{20}-\bar{x}M_{10}$$

$$\mu_{10}=0 \qquad \mu_{02}=M_{02}-\bar{y}M_{01}$$

The second moments represent the orientation of the contour, which can be derived from the second order central moments as shown below. These second moments also give us the covariance matrix of the contour, which can be used to find its eigenvalues and eigenvectors.

$$\mu'_{20}=\frac{\mu_{20}}{\mu_{00}}=\frac{M_{20}}{M_{00}}-\bar{x}^2$$

$$\mu'_{02}=\frac{\mu_{02}}{\mu_{00}}=\frac{M_{02}}{M_{00}}-\bar{y}^2$$

$$\mu'_{11}=\frac{\mu_{11}}{\mu_{00}}=\frac{M_{11}}{M_{00}}-\bar{x}\,\bar{y}$$

The covariance matrix of the contour in the image I(x, y) is:

$$cov[I(x,y)]=\begin{vmatrix}\mu'_{20} & \mu'_{11}\\ \mu'_{11} & \mu'_{02}\end{vmatrix}$$

The eigenvectors of this matrix correspond to the major and minor axis of the contour, so the orientation of the contour found from the angle of the eigenvector with the largest eigenvalue towards the axis closest to this eigenvector. This is found using the formula:

$$\theta=\frac{1}{2}\arctan\left(\frac{2\mu'_{11}}{\mu'_{20}-\mu'_{02}}\right)$$

The larger axis is the diameter of the contour in pixels, which is usually the major axis. The eccentricity of the contour, which refers to the curvature of the object is some ratio of the eigenvalues.

Assessing the shape of the mole using a contour

The shape of a mole is an important feature when assessing the risk associated with Melanoma. As moles that are asymmetrical or shaped irregularly are more likely to be cancerous and the degree of the irregularity could show the severity of the melanoma. It can also show if the melanoma is in its early stages. The information from the shape of the mole will be a crucial aspect of our monitoring method as any changes in its shape could indicate that the mole is evolving which would be problematic. To assess the shape of a mole in the image we will need to use the same contour as the previous section.

If we imagine the image as being coordinates in a graph and the contour or mole existing somewhere in those coordinates, by using the centre point we calculated previously and specific points on the boundary of the contour we can get an idea of its shape which would translate to the mole we are analysing.

For this project, there is an OpenCV function that finds the four most extreme points in the contour. With these points, we can calculate the distance from each point to the centre point, draw a line from each point to the centre and calculate the slope of each line. Finally with the slopes of each of the lines, we can calculate the angle between each line. These values will give us a general idea of the shape of the moles, any outliers should be noticeable.

When an image is viewed as coordinates the y-axis will increase in the opposite direction so when we apply formulas to points in the contour the results will be different than expected. Some mathematical manipulation can be applied to the results from the equations to solve this issue. This can be easily seen in the diagram below.

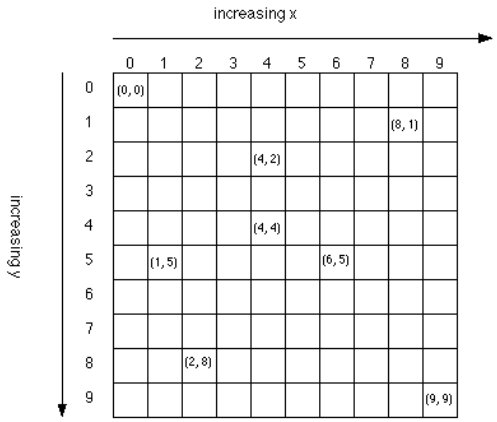


Figure 4 – Basic 9 by 9-pixel grid

The steps for assessing the shape were as following:

Using OpenCV to find the extreme points of the contour. This comprises of the leftmost, bottommost, rightmost and topmost point in the contour. When we have these points, we can draw a line from each extreme point to the centre and calculate each lines distance.

Points: $Extreme\ point = (x_1, y_1)$ $Center\ point = (x_2, y_2)$

$$Distance\ between\ two\ points = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

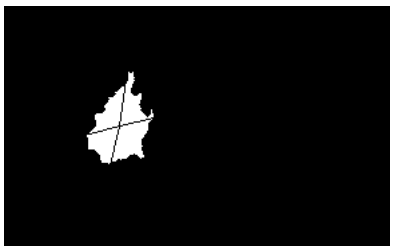


Figure 5 - Test example with the centre point and extreme points shown

Next, the slope of each of the lines is calculated from the same points as before. The order in which these points appear in the formula won't affect our results as long as the order is maintained for each coordinate. The result from each of the slopes will then need to be multiplied by -1 to get the actual slope of each of the lines.

Points: $Center\ point = (x_1, y_1)$ $Extrem\ point = (x_2, y_2)$

$$Slope\ between\ two\ points = \frac{y_2 - y_1}{x_2 - x_1}$$

Finally, the angle between each of the lines can be calculated by using the slope of each line. The first angle calculated will be between the leftmost point to bottommost point, and this process was continued in an anticlockwise direction until all the angles were calculated. The slope values needed to be adjusted for each angle. Multiply the slope to the left of the angle by -1 for each angle calculated except for the last angle between the topmost point to leftmost point where the slope to the right of the angle is multiplied by -1 instead.

$$Angle\ between\ two\ lines: \theta = \arctan\left(\left|\frac{m_1 - m_2}{1 + m_1 m_2}\right|\right.$$

Design Implementation

Hardware and Sensor Development

In the below diagram, we see the basic components of the ‘hardware and sensor’ aspect of the melanoma monitoring system and how they interact with each other. The main goal of combining these components is to create a complete system that can take an image of a malignant mole on an arm, process that image on the system and ultimately gather information about the mole to monitor its progress.

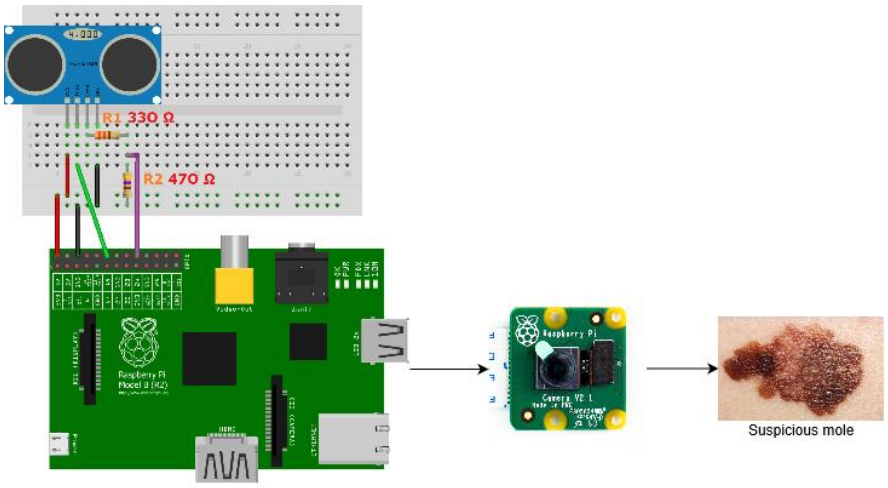


Figure 6 – Hardware and sensor interaction diagram

In order to create a suitable environment for the software implementation the combination of the camera module connected to a Raspberry Pi will be required to allow the system to take an image of a suspicious mole, process it and save the image on the Raspberry Pi’s microSD card. A condescend light source and white backdrop will be added to the environment to help achieve clean images from the camera module for the image processing. Finally, an ultrasonic sensor will be integrated with the system to measure the distance of the mole from the camera module.

One of the main motivations behind this project was to create a cheap Melanoma monitoring system that would allow patients to measure any changes in the appearance of their moles. This entire system can be produced for less than €100 which is affordable for most of the world's population.

HC-SR04 – Ultrasonic Range Sensor

The ultrasonic sensor is a crucial addition to the system as it will allow the user to ensure that the mole is relatively the same distance from the camera module in each image which ensures consistency across all the images taken by the system.

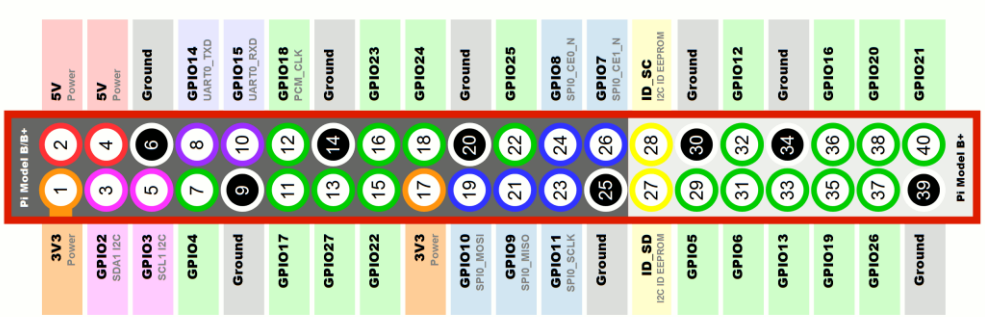


Figure 7 - Raspberry Pi GPIO Pin Layout [14]

The ultrasonic sensor can measure the distance from the camera module to the mole effectively from a range from 2-500 cm (5 meters) which makes it perfect for our system. It works by sending a 10µs high pulse from the Raspberry Pi to the sensor which causes the sensor to produce an 8-cycle sonic burst at 40 kHz. When the sensor hears the echo, it can determine the time between producing the burst and receiving its echo. To communicate this time back to the Raspberry Pi, the sensor sends one high pulse for a length equal to how long it took to hear the echo. So, the length of the object is proportional to how far the object is. All we need to do is listen to the echo pin and time how long its high for.

The Raspberry Pi will need to calculate the distance between the sensor and an object (i.e. the inspected mole). To do this the pulses will travel 2d as it will travel a distance d to the object and the same distance d back to the object. If the 'Time Elapsed' is the time the Echo pin is high then the distance travelled can be calculated as the following:

$$Speed\ of\ sound = 343\,m/s = 34300\,cm/s$$

$$Distance\ travelled\ d = d$$

$$Speed = \frac{Distance}{Time}$$

$$34300 = \frac{2d}{Time\ Elapsed}$$

$$d = (17150)(Time\ Elapsed)$$

The wiring for the Ultrasonic sensor circuit can be seen in the diagram below which shows a voltage divider between the Echo and Gnd pins. The voltage dividers purpose is to reduce the output voltage (Vout). In our circuit, Vin will be Echo, which needs to be decreased from 5V to a Vout of 3.3V. For this project the resistor R1 will be chosen to be 1 kΩ which will allow us to calculate a suitable resistor R2 value. A resistor R2 value of 2 kΩ will sufficiently meet our requirements as seen in our calculations. Both the Ultrasonic sensor circuit diagram and resistor 2 calculations can be seen below.

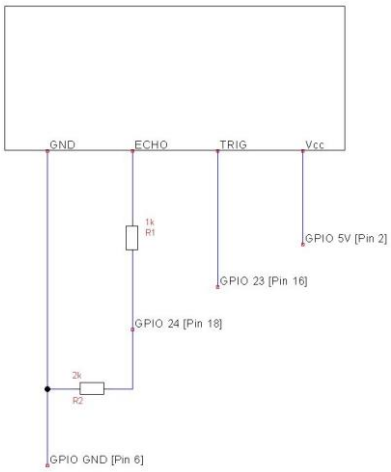


Figure 8 - Ultrasonic sensor circuit diagram [13]

Source Voltage from Raspberry Pi (Vin): 5V

Target output Voltage (Vout): 3.3V

$$Vout = Vin\left(\frac{R2}{R1+R2}\right)$$

$$R1 = 1k\Omega$$

$$3.3 = 5 \times \left(\frac{R2}{1000+R2}\right)$$

$$\frac{3.3}{5} = \frac{R2}{1000+R2}$$

$$0.66\left(1000+R2\right) = R2$$

$$660 + 0.66\,R2 = R2$$

$$660 = 0.34\,R2$$

$$R2 = \frac{660}{0.34} = 1.94k\Omega \sim 2k\Omega$$

Image Processing Design

For the computer vision and image processing task of the project, various algorithms will be applied to images of suspicious moles captured by a camera module attached to a Raspberry Pi. From these images, we will only be interested in the mole that we are monitoring, the rest of the image will be ignored. Regarding the moles that we are investigating with our monitoring system, we will be analysing the physical features of the mole at different stages to determine any changes in the size, diameter and shape of the mole.

From there, we will begin to implement the various image processing algorithms to separate the mole from its background to further extract information from the mole. While one aspect of the project aims to create a Melanoma monitoring system in the most effective and time efficient way as possible, this project also provides me with an opportunity to delve into new topics and concepts. From an image processing point of view, the aim is to learn many different image processing techniques to investigate which algorithms produce the most desirable results. While not being particularly time efficient, this method provides the greatest likelihood for success in the project as there is a backup plan in place if the main image processing technique does not work.

While researching similar Computer vision projects which were completed in the past, it became apparent that the main steps involved in achieving the project's goals would likely involve Image Preprocessing, Image Segmentation and Feature Extraction respectively. The Image Preprocessing and Image Segmentation used in this project were all directed at ensuring the most efficient feature extraction of the moles that were being analysed. From there, comparing the features of the mole using images under similar conditions at various stages of the cancers development will form the basis for the monitoring method used in this project, allowing us to measure and note any changes in the mole's appearance.

All of the Image processing techniques required for the software component of the project will be developed and tested separately before being combined at the latter stages of the development cycle, which ensures that the final program will be fully functional when added to the system.

Image Segmentation design

Initially, the original BGR image captured by the system will be converted to 8-bit grayscale to reduce its complexity when applying the image processing algorithms required to sufficiently segment the image.

Various Image Segmentation techniques were considered in this project's research stage. From these techniques, the watershed algorithm proved particularly effective in efficiently separating the mole from the background for this project as it could successfully segment numerous unrelated images of moles without requiring any prior knowledge of the mole's properties to produce these results. The watershed algorithm can be broadly described as a specific type of marker-based image segmentation which is based on an analogy of rain filling low lying regions in a landscape.

For the watershed algorithm to be applied, the grayscale image needs to be converted to a binary image using Otsu Thresholding. A binary image allows us to easily separate the main regions within the image to identify the image's foreground and background. The background of the image is firstly identified by applying Morphological dilation on the binary image. Once the background region is found, the same binary image is used to determine the foreground region by calculating the Euclidian distance transform of the image. The distance transform calculates the distance from each one pixel to its nearest zero pixel. After that, we apply a threshold to the distance transform to determine within a high probability that a given one pixel belongs to the foreground region.

When we are confident that the entire image has been analysed and the foreground of the image is adequately represented, we can begin to apply the watershed algorithm, and there are several functions in the Python library OpenCV that will enable us to apply the algorithm to the image. The watershed algorithm's main goals are to connect the one pixels in the binary image to a given region, label that region within the image and to merge the regions to find the boundary region. One of these regions from the algorithm should contain only the mole that we are interested in analysing.

Once the binary image's regions have been separated using labels, a new image can be created from the original image containing only the mole that we are interested in analysing. This would involve assigning all the other regions that did not contain the mole to appear as black, hence successfully segmenting the mole from the rest of the image.

Feature Extraction design

Once the mole has been successfully segmented from the original image, it can be analysed by thresholding the segmented image to measure the mole's properties. Thresholding in this case involves again converting the segmented image into a binary image using Otsu Thresholding. This binary image should have two distinct regions, one region should be white which will contain the mole that we are analysing and the rest of the image should be black. The region containing the mole is now isolated and we can investigate it by viewing the white region (i.e. mole) as a contour. A contour is a closed curve which is perfect for this application as any irregularly shaped moles can be represented in this way.

From there, the contour will be analysed using Image moments which help describe the features of the mole. Image moments in this context are the weighted average of the pixels intensities within the contour. The image moments of the contour can be used to calculate the centre point, area in terms of pixels, covariance matrix and orientation of the mole. These properties reveal a lot about the shape and features of the mole. Specifically, the covariance matrix from the previous step will be used to calculate the eigenvalues and eigenvectors of the contour, with the eigenvectors of this matrix corresponding to the major and minor axis of the contour. The major axis of the contour will be the diameter of the mole.

The final feature extraction method implemented in this system comprised of mathematical manipulation of formulas using the centre point and the four most extreme points (i.e. leftmost, rightmost, topmost and bottommost) in the contour to get a general idea of the overall shape of the mole. The formulas used were altered for this application as the coordinate of an image differ from a normal graph for which these formulas were initially created for. These mathematical manipulations involved calculating the distance from each extreme point to the centre point and drawing a line between these points. The slope of each line drawn is then calculated using each extreme point and the centre point of the contour. These slopes are used to calculate the angle between any two lines drawn previously. Once all the values from these formulas are calculated, the shape of the mole can be gathered as any outlier value will identify an irregularly shaped mole. A normally shaped mole will have equal distances from each extreme point and the centre point. The angle between each of the lines should also be similar.

While this method isn’t perfect and there is a lot of room for improvement in assessing the mole's shape this was the most effective method that could be implemented into the system in such a short period.

After the image of the mole is fully analysed, all the values of its features are compiled on a spreadsheet including the date in which the image was captured. This takes note of the mole's features at a specific point of the cancers development. Multiple images of the same mole are subsequently taken with our system under similar conditions, analysed with the same Image processing techniques and their features are added to our spreadsheet with their retrospective dates. These feature values are then compared with previous values to specify any change in the mole's appearance. Any changes in the mole's appearance might indicate that the cancer is growing which could be very dangerous, particularly for Melanoma as it could spread to other parts of the body.

In my opinion, this proved to be an effective method for monitoring the progress of melanoma or skin cancer on a patient’s arm. The area, diameter and eccentricity of the mole in pixels were used to determine if the shape of the mole has changed over time.

Design Testing & Evaluation

Hardware and Sensor Development

The ‘hardware and sensor’ aspect of the melanoma monitoring system was designed to effectively take an image of a malignant mole on a patient’s arm, process that image within the system and ultimately gather information about the mole's features to monitor its progress. Combining a Raspberry Pi with a camera module is essential in developing a system that is capable of this as the Raspberry Pi can easily store and analyse the images captured by the camera module.

From here, we encountered issues when trying to analyse the images taken by the camera module as the images were often of poor quality and it was difficult to take consistent images of the same mole. This was a result of poor lighting and a background that would often change with each image taken by the system. To improve the quality of the images taken by the system a condescend light source and a white backdrop was added to the area of interest which helped the mole stand out from the skin and create consistency when taking multiple images of the same mole.

One of the main problems when developing this project is that it is difficult to determine the actual size of the features of the inspected mole from the images taken by the system alone. Most of the potential solutions for this are very complex and unpractical for this project so instead I opted to add an ultrasonic sensor to the system to help create consistency across the different images taken of the same mole at various stages of its development. The ultrasonic sensor ensured that each image of the mole was captured an equal distance from the camera. The fact that each image is now captured using the same method allows us to compare the features of each of the various images of a mole with each other to measure any change in the appearance of the mole.

Figure 9 - Testing initial functionality of the system

Image Processing Development & Evaluation

Image Segmentation – Watershed algorithm

The process of determining what Image segmentation method was best for this project was done through trial and error. This proved to be a very long process, as it took a long time to test and develop each of the different Image Segmentation methods for separating the mole from the rest of the image. After testing several Image Segmentation methods with unsatisfactory results, it became apparent that the Watershed algorithm would be the best method for this project.

The Watershed algorithm proved to be a particularly effective method in efficiently separating the mole from the rest of the image as it could successfully segment numerous unrelated images of moles without requiring any prior knowledge of the mole's properties to produce these results. This is perfect for our project as the algorithm can segment any image of a mole captured by the system.

When using the Watershed algorithm functions with OpenCV, we will verify its functionality on a test image before seeing if it would be applicable to use in our system. These test images will be obtained by taking images of moles on my skin using a regular 1080p camera. This approach will allow us to develop the software required for a project like this while the hardware components are being delivered and set up.

Once we were confident that our program could adequately segment the mole from the background in several of our test images, we edited the code to allow the program to segment real world images taken by our system. This editing involved changing our code so that the program will be able to take an image of the patient's mole with the Raspberry Pi's camera module from just executing the program. With the new program, our system is now capable of taking an image of a mole on a patient's and segmenting that image to only show the mole with everything else in the image removed.

After the algorithm was fully developed, we encountered difficulties when we ran a variety of different tests trying to segment the same mole under different circumstances. It became clear from these tests that the algorithm couldn't accurately segment the mole in the images unless the mole was the main object in the image. This means that every time that we used the system, extra care needed to be taken to ensure that the mole was always the main object of the image.

In the below diagrams, we can see the image of the mole at various stages of the segmentation process. It shows the watershed algorithm being applied to the image of the mole to partition the image into meaningful regions, one of which will contain the mole we are trying to analyse. The boundary region is then outlined, which is where each region meets. Once the image is successfully partitioned, the mole can be separated from the rest of the image by setting all the regions that don't contain the mole to appear black. This creates a new image which only contains the mole. The last two images in the below diagram show the binary image of the mole which was used to extract information about the mole's features. All the alterations to the image to get a general idea of the mole's shape were also shown.

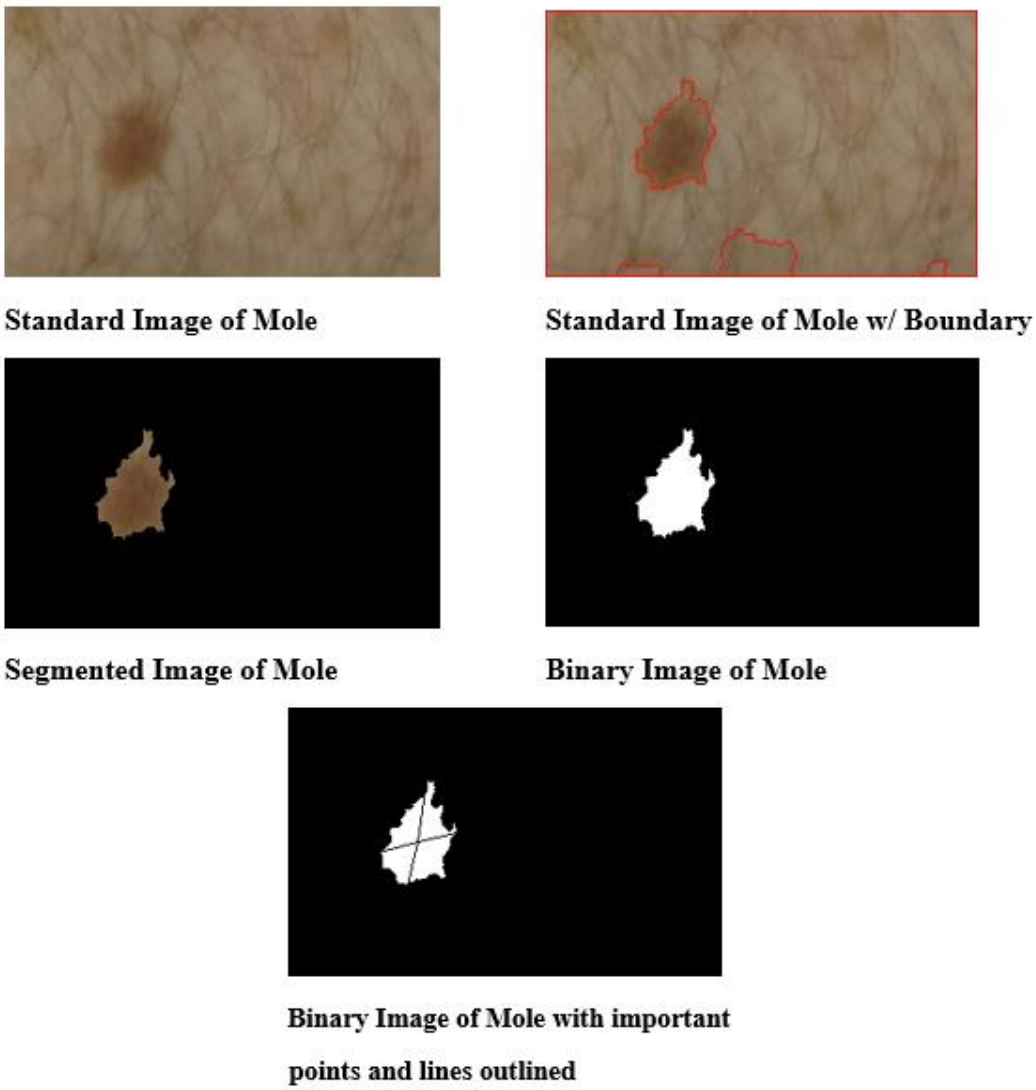


Figure 10 - Stages of the Image Segmentation process

Finally, when the program is fully executed, a marker diagram showing all the different regions detected in the image and their labels is shown to the user. Each of the different regions in the image is displayed in a different colour with a colourbar showing the difference in each of the regions. It is included to help the user to understand the image outputted from the watershed algorithm.

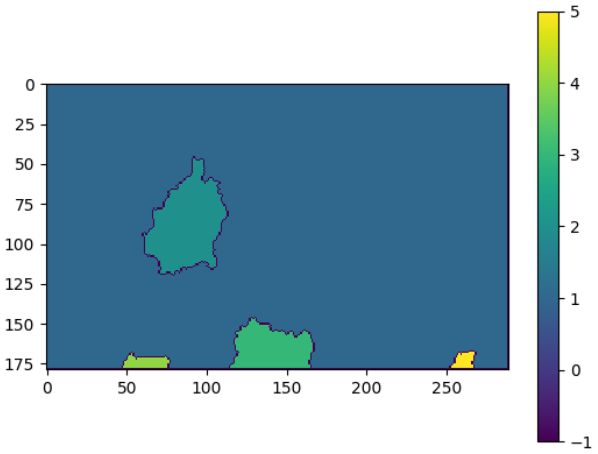


Figure 11 - Watershed marker image

Feature Extraction Development

Once we have a successfully segmented image of a mole, various feature extraction techniques can be used to analyse that mole's properties. The feature extraction methods used in this project was inspired by the ABCD method (Asymmetry, Border Irregularity, Colour Variation and Diameter), which is commonly used by GPs and Dermatologists to assess the severity of a suspicious mole. The mole's shape and size will be the main features analysed in each image taken by our system to determine if the mole has an irregular border or is asymmetrical, as well as the moles' overall size and diameter. These features shall hence be used as comparators with which the severity of all successive moles will be measured with by the system. There are some functions within the OpenCV library which will help us implement these feature extraction methods into our program.

During the research stage of the project, several different approaches for analysing the mole's features were considered. The most effective approach for this project's purposes involved converting the segmented images obtained from the previous section into a binary image, and then using image moments to calculate the properties of the only contour that is in that binary image which will represent the mole that we are trying to analyse.

As referred to a previous section Image moments are the weighted average of the pixel intensities which describe the images features and more importantly it gives us a visual representation of the mole's characteristics and its potential impact on the patient. The zeroth moment will represent the mole's area by calculating all the pixels it contains in the contour of the binary image. The first order moments, which is located at the mole's centre point will then be used to calculate the mole's second order moments, which refers to the contour's rotational inertia, which can then be used to calculate the mole's orientation, as well as its major and minor axes. The major axis will be the diameter of the mole in terms of pixels. The eccentricity of the mole can also be calculated from the second order moments which is a measure of how circular the mole is.

From here, we need to incorporate a method which enables us to anticipate the mole's overall shape. The initial plan was to generate a signature of the contour to represent the shape of the mole, with a signature being a 1-D functional representation of the contour's boundary, generated by plotting the distance from the contour's centre point to its boundary as a function of the angle at which the line created meets the boundary. However, having attempted to implement it into the system, it was decided not to use this method to assess the

mole’s shape as it was both difficult to implement the software required and it would require advanced knowledge which would inhibit many potential users from interpreting the results generated by the device. After this method proved too complex to incorporate into the system within the allotted timeframe, we decided to create a new method for assessing the mole’s shape which involved mathematical manipulation of formulas using the contour’s centre point and points that were in the contours boundary to anticipate the mole’s shape. This proved to be a useful feature extraction method to incorporate into our system as it could generate results on the mole’s shape, which would enable an average user with no prior Machine vision knowledge to interpret the results and to access and derive the greatest benefit from the device’s modalities.

Once our program for measuring the mole's features was fully functional, we believed that it would be useful to add a function to the program which could measure the colour variation across the mole. This could theoretically be done by plotting a histogram of the segmented image which would represent the distribution of pixel in the mole. It turned out that the histograms produce from each of the images didn’t tell us much about the colour variation in the mole as the histograms were often unreadable and it didn’t seem practical to compare histograms across images. Due to this it was decided not to include a function to produce a histogram in the final program.

Below you can see the sample output from the program, which is similar to what the user would see when the program is fully executed.

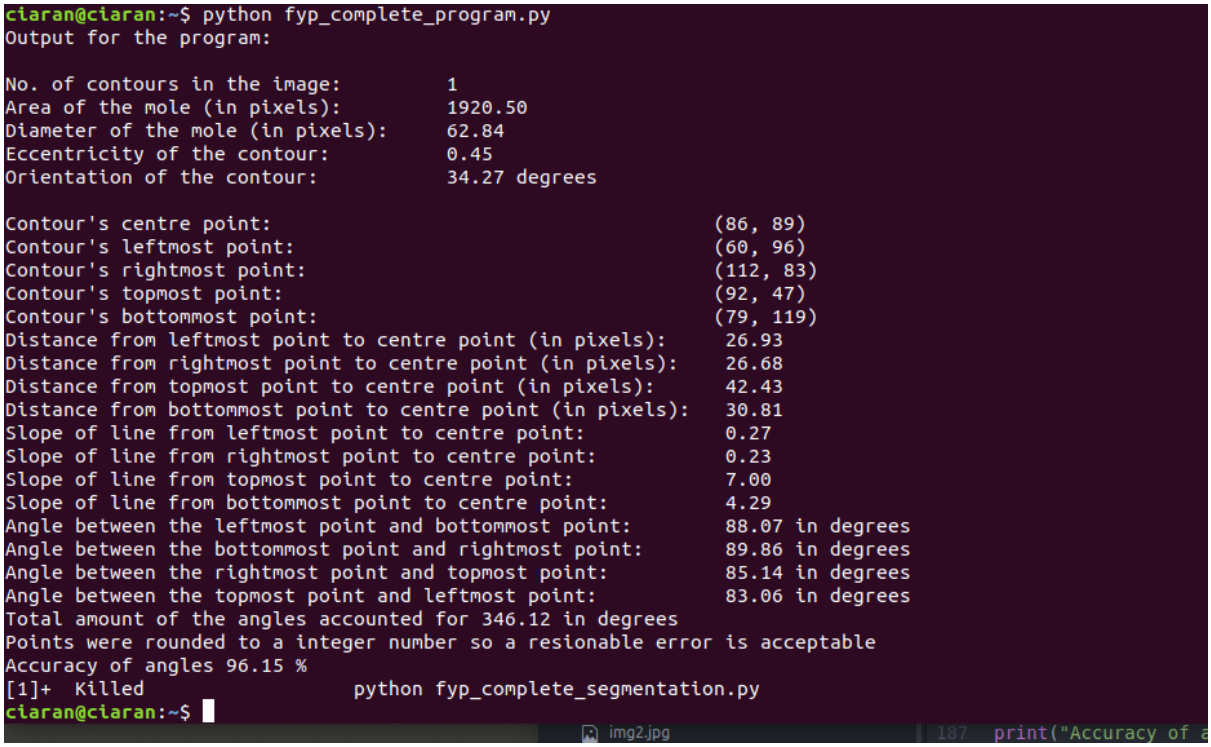


Figure 12 - Output from the program

Comparing the features of the mole

One of the project's main aims is to develop a way of monitoring the progress of a malignant mole associated with Melanoma or skin cancer. This requires us to compare the features of the mole throughout its development to determine if the mole's shape is changing. Any changes in the mole's shape can indicate if the cancer is growing which could be potentially dangerous for the patient as the cancer could spread to other parts of their body.

To compare these features, multiple images will be taken of the mole under similar conditions throughout the cancer's development and the results from the mole's features will be compiled into an Excel spreadsheet. The patient's mole will primarily be assessed based on its area and diameter in pixels, as well as its eccentricity, orientation and general shape in each image to fully monitor the changes that occur to the mole. We can then compare the results from our program by adding an ultrasonic sensor to the system to ensure consistency across all the images taken by the system.

Now that each image of the mole is captured an equal distance from the camera module, we can use the feature values in pixels to monitor any changes to the mole's appearance. Once we have a number of sets of values from the system, the patient's spreadsheet will enable us to learn more about the malignant mole and to make predictions about cancer's progression, which could assist healthcare professionals in creating treatment plans.

Below you can see an example of how the data from the program could be used to determine if there were any changes in the mole's appearance over a period of time. For our example, we were examining image of a mole on my skin so there were very little changes in our results as I don't have Melanoma. Though, the results from a patient with Melanoma might vary greatly with this method which would make these results from the program more useful as it would give the user a practical way of measuring any chance in their mole's appearance.

User's name	Ciaran	Ciaran	Ciaran	Ciaran
Date	06/03/ 2018	13/03/ 2018	20/03/2 018	27/03/ 2018
No. of contour in image	1	1	1	1
Area of the mole (in pixels)	1920.50	1856.35	1879.63	1892.67
Diameter of mole (in pixels)	62.84	64.23	63.96	63.96
Eccentricity of the mole	0.45	0.47	0.42	0.44
Orientation of the mole (in degrees)	34.27	36.32	35.63	33.46
Contour's centre point	(86, 89)	(84, 87)	(83, 86)	(85, 88)
Contour's leftmost point	(60, 96)	(62, 97)	(60, 95)	(61, 94)
Contour's rightmost point	(112, 83)	(115, 81)	(113, 79)	(114, 83)
Contour's topmost point	(92, 47)	(96, 42)	(93, 45)	(95, 43)
Contour's bottommost point	(79, 119)	(76, 123)	(77, 121)	(75, 122)
Distance from leftmost point to centre point (in pixels)	26.93	24.35	25.56	25.23
Distance from rightmost point to centre point (in pixels)	26.68	24.63	25.45	26.56
Distance from topmost point to centre point (in pixels)	42.43	43.85	44.21	44.13
Distance from bottommost point to centre point (in pixels)	30.81	32.19	31.23	32.63
Slope of line from leftmost point to centre point	0.27	0.32	0.30	0.31
Slope of line from rightmost point to centre point	0.23	0.21	0.24	0.25
Slope of line from topmost point to centre point	7.00	6.85	6.93	7.02
Slope of line from bottommost point to centre point	4.29	4.35	4.31	4.33
Angle between the leftmost point and bottommost point (in degrees)	88.07	87.93	88.10	87.89
Angle between the bottommost point and rightmost point (in degrees)	89.86	89.79	89.68	89.67
Angle between the rightmost point and topmost point (in degrees)	85.14	85.23	85.01	85.12
Angle between the topmost point and leftmost point (in degrees)	83.06	82.96	81.36	81.89
Total amount of angles accounted for (in degrees)	346.12	345.91	344.15	344.57
Accuracy of the angle values (%)	96.15	96.09	95.60	95.71

Implementation – Image Processing in Python

Previously we have demonstrated the effects of implementing the Image processing steps in the overall aim of monitoring the progress of a mole associated with Melanoma or skin cancer on a patient's arm, and the techniques used throughout the project that have considerably contributed towards this goal. In this section, we will explain how these Image processing steps were implemented into the project with Python.

The main two Image processing steps required to accomplish the project's goals are as follows:

- Image segmentation using the Watershed algorithm
- Feature Extraction

Once established, these steps will be developed and tested separately to ensure that, once these steps are combined, the final program will be fully functional.

The user may wish to have direct access to the images and results produced by this system, and to enable this kind of access, the Python script executed directly in the Raspberry Pi's terminal will produce five separate images and a custom output showing the results relating to the mole's features. Each of these images and the feature values from the execution will be available to the user in the Raspberry Pi's directories. These images will be:

- Standard Image of Mole (No image processing, displayed in BGR)
- Segmented Image of Mole
- Regions detected by the Watershed algorithm
- Binary image of the Mole
- Binary image of the Mole (including the centre point, extreme points and lines drawn)

Each of the five images produced by the system will be stored in their respective directory within the Raspberry Pi. The code used in each of the main Image processing steps is explained as follows.

Image segmentation using the Watershed algorithm

```
from time import sleep
from picamera import PiCamera
from matplotlib import pyplot as plt
import numpy as np
import cv2
import numpy.linalg as la
import math

camera = PiCamera()
camera.resolution = (1024, 768)
camera.start_preview()

sleep(5) # gives 5 seconds to adjust camera before taking image

# Taking a normal image of the mole with the raspbery pi camera module
standardPhoto =
camera.capture('/home/pi/Standard_Image/Normal_{:%Y_%m_%d_%H:%M}.jpg'
.format(datetime.datetime.now()))
```

Initially, the necessary libraries will be imported into the program. These libraries were installed onto the Raspberry Pi using the package manager Miniconda, which is a lighter version of Anaconda that includes only conda and its dependencies.

Here is the basic setup of the camera to take an image using the PiCamera library. The camera is set to 1024x768 resolution. When the script is executed, camera.start_preview will display the camera output of the Raspberry Pi camera onto the screen it is connected to. It will do so for 5 seconds to give the user an opportunity to adjust the camera to ensure that the camera is taking a clear image of the mole.

Note how the file's name is specific to the time in which it was taken (`.format(datetime.datetime.now())`). This means that each time the script is run, a new photo is outputted into the directory specified (in the case of standardPhoto, it is saved directly into /home/pi/Standard_Image/).

```
# Convert the captured image to grayscale
imgGray = cv2.cvtColor(standardPhoto, cv2.COLOR_BGR2GRAY)

# Applys Otsu Thresholding on the grayscale image
ret, thresh = cv2.threshold(imgGray, 0, 255, cv2.THRESH_BINARY_INV
  +cv2.THRESH_OTSU)

# Using Morphological closing to remove noise and reduce the effect of hairs
kernel = np.ones((3,3),np.uint8)
closing = cv2.morphologyEx(thresh,cv2.MORPH_CLOSE,kernel, iterations = 1)

# Finding the background of the image
sure_background = cv2.dialate(opening, kernel, iterations = 1)

# Finding the foreground of the image
distance_transform = cv2.distanceTransform(opening,cv2.DIST_L2,3)
ret, sure_foreground = cv2.threshold(distance_transform,
  0.7*distance_transform.max(), 255, 0)

# Finding the unknown region of the image
sure_foreground = np.uint8(sure_foreground)
unknown_region = cv2.subtract(sure_background, sure_foreground)
```

OpenCV enables us to simplify the process of applying the Watershed algorithm to an image by providing built-in functions in its library for most of the Image processing techniques required for the algorithm. Applying the Watershed algorithm to an image comprises of executing specific Image processing techniques in a specific order to achieve the goal of segmenting the image.

Firstly, the standard image taken by the system is converted into Grayscale to create an image with an 8-bit range of pixels that is easier to process. The produced grayscale image can be viewed as a topography surface which is critical when trying to apply the Watershed algorithm. The *cv2.threshold* function is then used to convert the grayscale image into a binary image using Otsu Thresholding.

Next, morphological closing was applied to the binary image to remove noise and reduce the effect of hairs in the image. It also prevents oversegmentation which is a substantial issue people encounter when using the Watershed algorithm via removing small holes within the image. The *cv2.morphologyEx* function is used to apply a 3x3 kernel of ones to perform morphological dilation followed by morphological erosion on the image resulting in a clean binary image.

Once the binary image is ready to be processed by the algorithm, we can separate the foreground, background and boundary regions of the image. The background region can be represented in an image by using morphological dilation to increase the boundary into the background region creating a new image without the boundary. The *cv2.dilate* function is used to re-apply the kernel to perform morphological dilation on the binary image to output an image containing only the foreground and background regions, removing the boundary region.

The image's foreground region can then be found by calculating the distance transform of the binary image followed by applying a threshold using the resulting image from the distance transform step. The *cv2.distanceTransform* function uses the Euclidean distance transform to calculate the closest zero pixel for each pixel in the binary image with a distance mask of 3. The resulting image will show a representation of the distance transform of the binary image. This image will then be thresholded with the distance transform to determine with a high probability which pixels belong to the foreground of the binary image. Finally, the boundary region is calculated by subtracting the foreground region from the background region.

Now that the binary image's foreground, background and boundary regions can be represented, we can begin to implement the Watershed algorithm.

```
# Marker labelling
retval, markers = cv2.connectedComponents(sure_foreground)

# Add one to all labels so that sure background is not 0, but 1
markers = markers+1

# Now, mark the unknown region zero
markers[unknown_region==255] = 0

# Applying the watershed
markers = cv2.watershed(standardPhoto, markers)

# Setting the unwanted regions to appear black
# Marker 2 should be our mole
standardPhoto[markers==-1]=(0,0,255) # Boundary appears red
cv2.imshow('Original image showing the boundary region in red', standardPhoto)
cv2.imwrite('/home/pi/Image_Watershed/Watershed_with_boundary_{:%Y_%m_%d_%H:%M}.jpg'
  .format(datetime.datetime.now()), standardPhoto)
standardPhoto[markers==1]=(0,0,0) # background region
cv2.imshow('Original image with the background region removed', standardPhoto)
cv2.imwrite('/home/pi/Image_Watershed/Watershed_without_ background_{:%Y_%m_%d_%H:%M}.jpg'
  .format(datetime.datetime.now()), standardPhoto)
standardPhoto[markers==3]=(0,0,0)
standardPhoto[markers==4]=(0,0,0)
standardPhoto[markers==5]=(0,0,0)
standardPhoto[markers==6]=(0,0,0)
standardPhoto[markers==7]=(0,0,0)
standardPhoto[markers==8]=(0,0,0)
standardPhoto[markers==9]=(0,0,0)
standardPhoto[markers==10]=(0,0,0)

# Displays and saves the segmented image
cv2.imshow('Watershed', standardPhoto)
cv2.imwrite('/home/pi/Image_Watershed/Watershed_{:%Y_%m_%d_%H:%M}.jpg'
  .format(datetime.datetime.now()),standardPhoto)

# Displays and saves the regions detected by watershed algorithm
imgplt = plt.imshow(markers)
plt.colorbar()
plt.show()
cv2.imwrite('/home/pi/Image_Watershed/Watershed_Regions_{:%Y_%m_%d_%H:%M}.jpg'
  .format(datetime.datetime.now()),imgplt)
```

This image demonstrates the creation of a marker, which is an array the same size as the original image to label the regions inside. It will be labelled with the *cv2.connectedComponents* function which uses integers to add labels to each object in the image. The background region of the image is labelled with 0, while the other objects in the image are labelled with integers beginning with 1. These markers will be added by 1 so that the background region is labelled 1 instead of 0 for the watershed algorithm not to consider it to be the unknown region. The unknown region will then be labelled 0 so that it will be disregarded. Finally, the boundary region will be the spaces with which the foreground and background regions meet.

Once all the objects in the image and the boundary region are labelled in our marker array, we can then use the watershed algorithm to partition the original image with the different regions detected in the marker array. This can be easily accomplished with OpenCV as there are specific functions designed for this purpose. The *cv2.watershed* function applies the watershed algorithm to the original image taken by our system using the image markers to separate the different regions.

When the original image is sufficiently partitioned into the necessary regions, we can isolate the region we are interested in analysing (i.e. the region containing the mole) by editing all the other regions to appear black to enable the image to only display the mole.

Although this might vary, the mole will generally be the largest object in the image so it will be contained in the marker label 2. In scenarios where the mole is not the largest object in the image, the program can be easily edited to segment whatever marker label the mole appears in.

This marker-based watershed algorithm is an interactive segmentation, making it possible to specify which regions will appear in the image. For this reason, we will create and save new images for each of the main stages of the segmentation process. These images will include:

- Original image of the mole
- Image of the mole including the boundary region
- Image of the mole and boundary region with the background region removed
- Image showing only the mole with all other objects and regions removed

The marker array showing all the objects detected in the image will also be displayed and saved on the system for situations where the user is confused and cannot interpret the algorithm's results. All the objects in the marker array will be represented in different colours, along with a colour bar to allow the user to easily identify the object's label, and interpret the results.

Once the image is successfully segmented only to display the mole, we can begin to implement the feature extraction steps required for its analysis.

Feature Extraction

```
# Reads the segmented image of the mole
img_seg1 = img

# Convert the segemted image to grayscale
img_gray1 = cv2.cvtColor(img_seg1, cv2.COLOR_BGR2GRAY)

# Use Otsu thresholding to convert the segmented image to a binary image
ret, thresh2 = cv2.threshold(img_gray1, 0, 255,
  cv2.THRESH_BINARY+cv2.THRESH_OTSU)

# Displays and saves the binary image
cv2.imshow('Thresholded image of the mole', thresh2)
cv2.imwrite('/home/pi/Image_Segmented_Thresholded/Segmented_Image_
  Thresholded_{:%Y_%m_%d_%H:%M}.jpg'.format(datetime.datetime.now()),imgplt)
```

To prepare the segmented image of the mole for analysis, the image is converted into Grayscale to create an image with an 8-bit range of pixels that is easier to process. The cv2.threshold function is again used to convert the grayscale image into a binary image using Otsu thresholding. The resulting image will clearly show the mole we are interested in analysing in a black and white image, which will then be saved on the system in its directory.

```
# Finding the contours in the binary image
# The image should contain only two contours
im2, contours, hierarchy = cv2.findContours(thresh2, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

for cnt in contours:
    moments = cv2.moments(cnt) # Calculates image moments
    if moments['m00']!=0:
        # Approximation of the centre of the contour
        cx = int(moments['m10']/moments['m00'])    # cx = M10/M00
        cy = int(moments['m01']/moments['m00'])  # cy = M01/M00

        # Actual centre of the contour
        cx1 = (moments['m10']/moments['m00'])    # cx = M10/M00
        cy1 = (moments['m01']/moments['m00'])  # cy = M01/M00

        # Calculates the area of the contour
        area = moments['m00']
        round_area = round(area, 2)
```

Since the region containing the mole is now isolated, we can analyse its properties by viewing the region in the binary image as a contour, which is a closed curve in the binary image so any irregularly shaped mole can be represented in this way. Only one contour should appear in our binary image, and this will represent the mole we are analysing. The *cv2.findContours* function is used to find all the contours in the binary image. The *cv2.RETR_TREE* and *cv2.CHAIN_APPROX_NONE* parameters in the function retrieve all the contours, reconstructs a full hierarchy of nested contours and stores every contour point found within the image, which is particularly useful if there is a contour inside a contour in the image.

From here, the contour will be analysed using Image moments to describe the mole's features. Most of the values from these feature extraction steps will be approximated to improve readability from users. In the context of this problem, Image moments are the weighted average of the pixel intensities of the contour which represents the mole we are analysing. The *cv2.moments* function is used to calculate the zeroth, first and second order moments of the contour. The zeroth moment (M_{00}) will be the area in pixels of the contour (i.e. the mole). The first order moments (M_{10} and M_{01}) are the sum of the x and y coordinates over the area respectively in terms of pixels which is used to calculate the contours centre point. The second order moments (M_{20}, M_{11} and M_{02}) represent the orientation of the contour which will help us calculate the eccentricity, diameter and orientation of the contour.

```
# Finds second order central moments
mu20 = moments['mu20']
mu11 = moments['mu11']
mu02 = moments['mu02']

# Calculate the covariance matrix of the contour
covar_mat = np.array([[mu20, mu11],[mu11, mu02]])

# Calculates the eigenvalues and eigenvectors of the covariance matrix
evals, evects = la.eigh(covar_mat)

# Seperates the eigenvalues in the matrix
[eval1, eval2] = evals

# Calculates the eccentricity of the contour
eccent = eval1/eval2

# Finds the major and minor axis of the contour
w, l = 4*np.sqrt(evals/area)

# Sets the diameter of the contour to the longer axis
if (l > w):
    diameter = l
else:
    diameter = w

# Calculates the orientation of the contour
orient = 0.5*np.arctan((2*mu11)/(mu20-mu02))
orientation = orient * 100
```

The initial step for using the second order moments to describe the mole's features involve assigning each moment value to a variable, which simplifies the representation of the covariant matrix of the contour. The covariant matrix is a symmetric matrix that summaries the variance of the second order moments in the contour. The *numpy.linalg.eigh* function uses the covariance matrix to calculate the eigenvalues and eigenvectors of the contour. The values in the eigenvalues matrix from the previous step are then separated to calculate the

eccentricity of the contour. Eccentricity is a measure of how circular the contour or mole is and it is calculated as some ratio of the eigenvalues. The major and minor axes can also be calculated from the contours eigenvalues, with the larger axis representing the diameter of the contour or mole in pixels. The second order moments can also be utilized to calculate the contour's orientation by implementing a specific formula, which was previously explained in an earlier section of the project.

Now that the mole is fully analysed using Image moments, we can then add a new method to assess the mole's shape, which will be based on mathematical manipulation of formulae using the centre point and the four most extreme points in the contour to ascertain the overall shape of the mole. This method involves viewing the image as a pixel graph and each pixel as a coordinate. The y-axis in the pixel graph will be increasing in the opposite direction as expected so the formulae will need to be altered to account for this.

The code below will be altered from its appearance in our program to show the feature extraction process for one of the extreme points. These steps were executed three more times in our program for each of the extreme points, and was performed in this manner to save space in our document. The full code will be made available in the appendix of this document.

```
    # Finds the left most extreme points in the contour
    leftmost = tuple(cnt[cnt[:,:,0].argmin()][0])

    # Draw a circle at leftmost point in black
    cv2.circle(im2,(leftmost),1,(0,0,0),-1)

    # Seperates the x and y points of the leftmost point
    [lm_x, lm_y] = leftmost

    # Draws a line from leftmost point to the centre point in black
    cv2.line(im2, (cx, cy), (leftmost), (0,0,0), 1)

    # Calculates the distance from left point to the center point
    # Using the distance formula = square root of ((x2 - x1)^2 + (y2 - y1)^2)
    distance_lm2centre = math.sqrt(((cx - lm_x)**2) + ((cy - lm_y)**2))

    # Calculate the slope of each line leftmost point to the center point
    # Slope of points: m = (y2 - y1) / (x2 - x1)
    # The order of the points doesn't affect our results
    slope_lm2centre = ((lm_y - cy) / (lm_x - cx))

    # Since the images are viewed in the opposite direction
    # Each of the slopes calculates are the inverse of the expected values
    # Multiplying each slope by -1 solves this problem
    m_lm = slope_lm2centre * -1
```

In the above example, the program is searching the contour's tuple for the leftmost point. The leftmost point's coordinates will then be separated by assigning variables to each coordinate, which will prove important when we insert these points into our formulas. The points, and the line from the point to the centre point will then be added to the binary image of the segmented mole.

Now that we have the leftmost point and the centre point, we can then calculate the distance between the two points in pixels using the distance formula. The slope of the line created can also be calculated with these points using the slope of the line formula. The result from the slope will then be multiped by -1 to compensate for the difference the change in y-axis makes.

```
# Calculates the angles between each of the extreme points
# Setting the slopes in order to view the image in normal coordinates
new_m_lm = m_lm * 1
new_m_bm = m_bm * -1
tetha_lmvbm = abs(math.degrees(math.atan((new_m_lm - new_m_bm)/
    (1 + (new_m_lm * new_m_bm)))))

# Calculates the total angle between each of the extreme points
# and accuracy of results
total = tetha_lmvbm + tetha_bmvrm + tetha_rmvtm + tetha_tmvlm
tetha_accuracy = (total/360) * 100

# Draws the contour and centre point in black
cv2.drawContours(thresh2, [cnt],0,(0,0,0),1)
cv2.circle(im2,(cx,cy),1,(0,0,0),-1)
```

Finally, we can use the slopes to calculate the angles (in degrees) in between each of the extreme points. These angles will then be summed to determine how accurate the method is.

Once we have these values, we can use them to get a picture of the overall shape of the moles. Any outlier values from this method will indicate an irregularly shaped mole which will be noted.

Conclusion

As of the writing of this report, the system has reached a stage where it is capable of monitoring the progress of a mole associated with Melanoma on a patient's arm throughout the melanomas development. In the process of developing such a system, there has been a thorough investigation into the many of the different methods that would achieve the aims of the project.

From an improvement point of view, there is still work that could be done to improve the way the product interacts with its user and the portability of the entire system. This is where further work will be done in the future to have the project working at an optimal. Some features that could be integrated into the project might include, an LED circuit to provide the light source the system needs and an Android application that will allow users to interact with the product more efficiently.

Although the initial plan of building a self-contained enclosure to provide all the functions that our Monitoring system needed didn't end up working out, the changes that had to be made to the project's design still achieved the same objectives as was initially set out. As well as this, the benefit of learning the operation of the image processing techniques used in the project helped shape a broader understanding of machine vision. It was also really fulfilling to apply these techniques to a real-world application whose impact could potentially improve the treatment and outcome of people diagnosed with melanoma.

Acknowledgements

Firstly, I would like to thank both of my supervisors, Dr. Sean McGrath & Dr. Colin Flanagan for their help throughout the project. They were both extremely helpful and approachable.

I would also like to especially thank Dr. Colin Flanagan for his help with the subject material, his help in the understanding the theory of the concepts used in the project proved vital in its development.

References

[1] Wiki.python.org. (2018). *BeginnersGuide/Overview – Python Wiki*. [Online] Available: https://wiki.python.org/moin/BeginnersGuide

[2] Opencv.org. (2018). *About – OpenCV library*. [Online] Available: https://opencv.org/about.html

[3] M. Ramezani, A. Karimian & P. Moallem, "Automatic Detection of Melanoma using Macroscopic Images," in Journal of Medical Signals and Sensors, 2014, pp. 281-290.

[4] C O'Dwyer, "Smartphone Medical Toolkit," in Department of Electronic and Computer Engineering, University of Limerick, Limerick, Ireland 2017.

[5] Irish Skin Foundation. (2017). *Melanoma & Skin Cancer – Irish Skin Foundation*. [Online]. Available: https://irishskin.ie/melanoma-skin-cancer/

[6] Ireland's Health Service. (2017). *Introduction – Ireland's Health Service*. [Online] Available: http://www.hse.ie/eng/health/az/C/Cancer,-skin-melanoma-/

[7] Labbookpages. (2018). *Otsu Thresholding – The Lab Book Pages.* [Online] Available: http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html

[8] Scipy-lectures. (2018). *Otsu thresholding – Scipy lecture notes.* [Online]. Available: http://www.scipy-lectures.org/packages/scikit-image/auto_examples/plot_threshold.html

[9] Docs.opencv. (2018). *OpenCV: Morphological Transformations.* [Online]. Available: https://docs.opencv.org/trunk/d9/d61/tutorial_py_morphological_ops.html

[10] C. Solomon and T. Breckon, *Fundamentals of Digital Image Processing – A Practical Approach with examples in MATLAB.* 1st ed. Chichester, England: Wiley-Blackwell, 2011.

[11] Docs.opencv. (2018). *OpenCV: Image Segmentation with Watershed Algorithm.* [Online]. Available: https://docs.opencv.org/3.1.0/d3/db4/tutorial_py_watershed.html

[12] C. Flanagan, Lecture notes, Topic: "Segmentation", University of Limerick, 2018.

[13] Cases for your RaspberryPi. (2018). *HC-SR04 Ultrasonic Range Sensor on the Raspberry Pi.*[Online]. Available: https://www.modmypi.com/blog/hc-sr04-ultrasonic-range-sensor-on-the-raspberry-pi

[14] Raspberry Pi Spy. (2018). Simple Guide to the Raspberry Pi GPIO Header and Pins – Raspberry Pi Spy. [Online]. Available: https://www.raspberrypi-spy.co.uk/2012/06/simple-guide-to-the-rpi-gpio-header-and-pins/

Appendices

FYP Poster

Melanoma (Skin Cancer) Long-Term Monitoring

UNIVERSITY of LIMERICK
OLLSCOIL LUIMNIGH

Student name: Ciaran Carroll
ID Number: 13113259
Programme of study: Electronic & Computer Engineering (LM118)

Introduction

The specification of this project is to demonstrate the capabilities that Image processing techniques has in a medical context.

Specifically, we will be designing a system to facilitate the monitoring process of Melanoma or other types of skin cancer. The usefulness of such a product could have huge benefits & implications for the medical sector at large, particularly for patients in monitoring any changes in their malignant moles.

Due to the unpredictable nature of Melanoma, and the importance of monitoring its progress, the aiding of the monitoring process will lead to a reduction in the cost, skill level and knowledge required to monitor melanoma. As well as this, it would provide dramatic improvements in the treatment and outcome of diagnosed patients.

This is made possible as the malignant moles associated with the disease are visible on the skin.

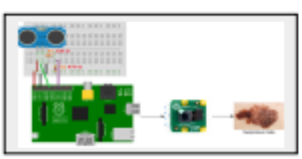
Aim

The overall aim of the system can be split into three separate sections:

- Design a suitable environment for the monitoring system, powered by combining a camera module with a Raspberry Pi, to apply the Machine Vision.
- Using the Raspberry Pi camera, take an image of the malignant mole. Furthermore, incorporate various Image processing techniques to separate the mole from its background to further extract information from the mole.
- Create a method which monitors the progress of a malignant mole, by comparing and analysing images of the mole at various stages of its development.

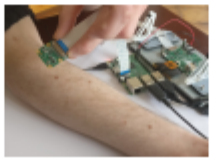
Method

* A Raspberry Pi and camera module was acquired, which will allow the system to take an image of a suspicious mole, process it and save it on the Raspberry Pi.
* The combination of the camera module connected to a Raspberry Pi, condescend light sources and white backdrop create a suitable environment for the monitoring system.
* An ultrasonic sensor was integrated with the system to measure the distance of the mole from the camera.
* Voltage was applied to the ultrasonic sensor using the built-in Raspberry Pi GPIO (3.3V for Vin).
* The Machine Vision aspect of the project required combining various steps for Image Processing such as Preprocessing, Segmentation and Feature Extraction.
* Preprocessing the image of the mole involved using Morphological operations to remove the effect of hairs and noise from the image.



Basic Diagram showing how the hardware and sensor of the system interact with one another

- A marker-based Image segmentation method was then used to partition the preprocessed image into meaningful labelled regions which was then used to separate the mole in the image from the skin and background. The Watershed algorithm proved useful for this application.
- Various Image processing techniques were applied to the new image of the mole to extract information about its features.
- A second image is taken of the mole in the same environment and processed with the same Image Processing steps.
- The features of both images are compared forming the basis for the melanoma monitoring method. Any changes in the mole's appearance will be measured.



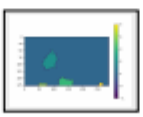
Testing the functionality of the Machine Visions application by taking images of my arm and processing it

Results

- Python script created on the Raspberry Pi will take an image of a mole with the camera module, undergo various Image Processing techniques on the image taken, and save the image at specific stages of the program on the system. The script is designed so that no overwriting occurs (new photo with a new name is created each time the script is executed).
- Gaussian filter applied to the image, gives unsatisfactory results (hairs are still visible and defined in the image).
- Specific type of Morphological operation called closing was applied instead of a filter, giving us our desired result (hairs are not defined in the image).
- Marker-based Image segmentation using the Watershed Algorithm achieved our goal of partitioning and labelling the image into meaningful regions. One of these regions contained the mole that will be analysed.
- A new image will be created with all the other labelled regions modified to appear black creating the segmented image containing only the mole.

Original image

Image w/ labelling applied



Watershed marker image

Segmented image

- Images above show stages of the image segmentation while using the watershed algorithm.
- Segmenting the image will allow us to further analyse the mole using multiple Feature Extraction stages.
- The Feature Extraction methods we will use in this project will be based on the ABCD method which is commonly used by GPs and Dermatologists to determine whether a mole is benign or malignant. The ABCD in this context stands for Asymmetry, Border irregularity, Diameter and Colour Variation.
- Various Image Processing techniques will be required to prepare the image for feature extraction such as Shape descriptors, Otsu Thresholding and changing the colour space.

Description of the function of the main feature extraction steps applied to the segmented region:

- Counting the pixels in the region containing the moles allows us to calculate the moles pixel area.
- Calculating a histogram of the region allows us to graphically see the number of pixels for each pixel value which shows the Colour Variation in the region.
- Calculating the average distance the centre of the mole is to each of the edges allowed us to calculate the moles diameter.
- Using the angle at which the centre point meets each of the edges allows us to determine whether the moles border was irregular in shape which would indicate that the mole is asymmetrical.

Comparing these features in images of a mole at various stages of the cancers development has proved to be a useful method for monitoring the progress of melanoma. The pixel areas, histograms, diameters and eccentricity of the moles will be used to determine if the feature of the mole has changed over time.

Conclusion and personal reflection

I believe the design & implementation of this above system have been achieved effectively. While there is further room for improvement required for making this a complete product, the overall functionality provides a solid foundation from which to refine and perfect.

Certain aspects of the project, such as the way the product interacts with its users and the portability of the entire system could have been improved. This could have been achieved by integrating a LED circuit to provide the light source and developing an Android application to allow users to easily interact with the product.

Overall, I think this project as a whole was perfect for giving me a platform to go on to learn many skills (such as Python programming and Machine Vision). At the same time, it allowed me to apply these skills to a real-world application whose impact could potentially improve the treatment and outcome of patients diagnosed with melanoma.

Acknowledgements

I would like to thank both of my supervisors, Dr Sean McGrath & Dr Colin Flanagan. Their help & assistance proved vital within both the design & implementation process of the project.

E&CE Department of Electronic & Computer Engineering

fyp_complete_program.py

```
## ------------------------------------------------------------------------------------
##
## Student name:    Ciaran Carroll
## Student Id:      13113259
##
## Implementation of Image Segmentation using  Morphological Watershed
##
## Uses a test image to develop a working prototype
##
##------------------------------------------------------------------------------------

from time import sleep
from time import datetime
import picamera import PiCamera
import numpy as np
import cv2
from matplotlib import pyplot as plt
import numpy.linalg as la
import math

camera = PiCamera()
camera.resolution = (1024, 768)
camera.start_preview()

sleep(5) # gives 5 seconds to adjust camera before taking image

# Taking a normal image with the raspbery pi camera module
standardPhoto = camera.capture('/home/pi/Standard_Image/Normal_{:%Y_%m_%d_%H:%M}.jpg'
  .format(datetime.datetime.now()))

# Convert the captured image to grayscale
imgGray = cv2.cvtColor(standardPhoto, cv2.COLOR_BGR2GRAY)

# Image Segmentation using the watershed algorithm
# Apply Otsu thresholding on the grayscale image
retval, threshold = cv2.threshold(imgGray, 0, 255, cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)

# Use Morphological closing to remove noise and reduce the effect of hairs
# Is also prevents oversegmentation by removing small holes
kernel = np.ones((3,3), np.uint8)
closing = cv2.morphologyEx(threshold, cv2.MORPH_CLOSE,kernel, iterations = 1)

# Use Morphological dialation to find areas of the image that are surely background
sure_background = cv2.dilate(closing, kernel, iterations=1)

# Use Euclidean distance transform to calculates the distance from every binary image
# to the nearest zero pixel
distance_transform = cv2.distanceTransform(opening, cv2.DIST_L2, 3)

# Apply a threshold to the distance transform to determine with a high probability
retval, sure_foreground = cv2.threshold(distance_transform,0.7*distance_transform.max(),255,0)
```

```
# Convert the sure foreground image from a float32 array to uint8
# Finding the unknown region by subtracting the sure forground from sure background
sure_foreground = np.uint8(sure_foreground)
unknown = cv2.subtract(sure_background,sure_foreground)

# Marker labelling to build the dams (barriers) to stop the water from merging
# Labelling the background region 0, and the other objects with integers starting with 1
retval, markers = cv2.connectedComponents(sure_foreground)

# Add one to all labels so that sure background is not 0, but 1
# This is so that the watershed doesn't consider it as the unknown region
markers = markers+1

# Label the unknown region zero
markers[unknown==255] = 0

# Now we let the water fall and our barriers be drawn in red
# Apply the watershed and convert the result back into uint8 image
markers = cv2.watershed(standardPhoto, markers)

# The watershed algorithm labels each region order of size
# The second region should be our mole
standardPhoto[markers==-1] = (0,0,255) # Boundary region
cv2.imshow('Original image showing the boundary region in red', standardPhoto)
cv2.imwrite('/home/pi/Segmented_Image/Watershed_with_boundary_{:%Y_%m_%d_%H:%M}.jpg'
  .format(datetime.datetime.now()),standardPhoto)
standardPhoto[markers==1] = (0,0,0) # Background region
cv2.imshow('Original image showing the boundary region removed', standardPhoto)
cv2.imwrite('/home/pi/Segmented_Image/Watershed_without_background_{:%Y_%m_%d_%H:%M}.jpg'
  .format(datetime.datetime.now()),standardPhoto)
standardPhoto[markers==-1] = (0,0,0) # Boundary region
standardPhoto[markers==3] = (0,0,0)
standardPhoto[markers==4] = (0,0,0)
standardPhoto[markers==5] = (0,0,0)
standardPhoto[markers==6] = (0,0,0)
standardPhoto[markers==7] = (0,0,0)
standardPhoto[markers==8] = (0,0,0)
standardPhoto[markers==9] = (0,0,0)
standardPhoto[markers==10] = (0,0,0)

# Displays and saves Segmented image
cv2.imshow('Watershed', standardPhoto)
cv2.imwrite('/home/pi/Segmented_Image/Segmented_Image_{:%Y_%m_%d_%H:%M}.jpg'
  .format(datetime.datetime.now()),standardPhoto)

# Displays and saves the regions detected by watershed algorithm
imgplt = plt.imshow(markers)
plt.colorbar()
plt.show()
```

```
cv2.imwrite('/home/pi/Image_Watershed/Watershed_{:%Y_%m_%d_%H:%M}.jpg'
  .format(datetime.datetime.now()),markers)

# Reads the segmented image of the mole
img_segl = standardPhoto

# Reads the segmented image of the mole
img_segl = cv2.imread('Segmented_mole.jpg', cv2.IMREAD_UNCHANGED)

# Convert the segemted image to grayscale
img_grayl = cv2.cvtColor(img_segl, cv2.COLOR_BGR2GRAY)

# Use Otsu thresholding to convert the segmented image to a binary image
ret, thresh2 = cv2.threshold(img_grayl, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)

# Shows the binary image
cv2.imshow('Thresholded image of the mole', thresh2)
cv2.imwrite('/home/pi/Image_Segmented_Threshold/Segmentated_Image_Thresholding_{:%Y_%m_%d_%H:%M}.jpg'
  .format(datetime.datetime.now()), thresh2)

# Feature Extraction of the segmented image

# Finding the contours in the binary image
# The image should contain only two contours
im2, contours, hierarchy = cv2.findContours(thresh2, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

for cnt in contours:
    moments = cv2.moments(cnt) # Calculates image moments
    if moments['m00']!=0:
        # Approximation of the centre of the contour
        cx = int(moments['m10']/moments['m00'])   # cx = M10/M00
        cy = int(moments['m01']/moments['m00'])  # cy = M01/M00

        # Actual centre of the contour
        cxl = (moments['m10']/moments['m00'])   # cx = M10/M00
        cyl = (moments['m01']/moments['m00'])  # cy = M01/M00

        # Calculates the area of the contour
        area = moments['m00']
        round_area = round(area, 2)

        # Finds second order central moments
        mu20 = moments['mu20']
        mu11 = moments['mu11']
        mu02 = moments['mu02']

        # Calculate the covariance matrix of the contour
        covar_mat = np.array([[mu20, mu11],[mu11, mu02]])

        # Calculates the eigenvalues and eigenvectors of the covariance matrix
```

```
        # Calculates the eigenvalues and eigenvectors of the covariance matrix
        evals, evects = la.eigh(covar_mat)

        # Seperates the eigenvalues in the matrix
        [eval1, eval2] = evals

        # Calculates the eccentricity of the contour
        eccent = eval1/eval2

        # Finds the major and minor axis of the contour
        w, l = 4*np.sqrt(evals/area)

        # Sets the diameter of the contour to the longer axis
        if (l > w):
            diameter = l
        else:
            diameter = w

        # Calculates the orientation of the contour
        orient = 0.5*np.arctan((2*mu11)/(mu20-mu02))
        orientation = orient * 100

        # Finds the four most extreme points in the contour
        leftmost = tuple(cnt[cnt[:,:,0].argmin()][0])
        rightmost = tuple(cnt[cnt[:,:,0].argmax()][0])
        topmost = tuple(cnt[cnt[:,:,1].argmin()][0])
        bottommost = tuple(cnt[cnt[:,:,1].argmax()][0])

        # Draw a circle at each of the extreme points in black
        cv2.circle(im2,(leftmost),1,(0,0,0),-1)
        cv2.circle(im2,(rightmost),1,(0,0,0),-1)
        cv2.circle(im2,(topmost),1,(0,0,0),-1)
        cv2.circle(im2,(bottommost),1,(0,0,0),-1)

        # Seperates the x and y points in each of the extreme points
        [lm_x, lm_y] = leftmost
        [rm_x, rm_y] = rightmost
        [tm_x, tm_y] = topmost
        [bm_x, bm_y] = bottommost

        # Draws a line from each extreme point to the centre in black
        cv2.line(im2, (cx, cy), (leftmost), (0,0,0), 1)
        cv2.line(im2, (cx, cy), (rightmost), (0,0,0), 1)
        cv2.line(im2, (cx, cy), (topmost), (0,0,0), 1)
        cv2.line(im2, (cx, cy), (bottommost), (0,0,0), 1)
```

```
        # Calculates the distance from each extreme point to the center
        # Using the distance formula = square root of ((x2 - x1)^2 + (y2 - y1)^2)
        distance_lm2centre = math.sqrt(((cx - lm_x)**2) + ((cy - lm_y)**2))
        distance_rm2centre = math.sqrt(((cx - rm_x)**2) + ((cy - rm_y)**2))
        distance_tm2centre = math.sqrt(((cx - tm_x)**2) + ((cy - tm_y)**2))
        distance_bm2centre = math.sqrt(((cx - bm_x)**2) + ((cy - bm_y)**2))

        # Calculate the slope of each line from extreme point to the center
        # Slope of points m = (y2 - y1) / (x2 - x1)
        # The order of the points doesn't affect the results
        slope_lm2centre = ((lm_y - cy) / (lm_x - cx))
        slope_rm2centre = ((rm_y - cy) / (rm_x - cx))
        slope_tm2centre = ((tm_y - cy) / (tm_x - cx))
        slope_bm2centre = ((bm_y - cy) / (bm_x - cx))

        # Since the images is viewed in the opposite direction
        # Each of the slopes calculates are the inverse of the expected values
        # Multiplying each slope by -1 solves this problem
        m_lm = slope_lm2centre * -1
        m_rm = slope_rm2centre * -1
        m_tm = slope_tm2centre * -1
        m_bm = slope_bm2centre * -1

        # Calculates the angles between each of the extreme points
        # Setting the slopes in order to view the image in normal coordinates
        new_m_lm = m_lm * 1
        new_m_bm = m_bm * -1
        tetha_lmvbm = abs(math.degrees(math.atan((new_m_lm - new_m_bm)/(1 + (new_m_lm * new_m_bm)))))

        new_m_bm = m_bm * 1
        new_m_rm = m_rm * -1
        tetha_bmvrm = abs(math.degrees(math.atan((new_m_bm - new_m_rm)/(1 + (new_m_bm * new_m_rm)))))

        new_m_rm = m_rm * 1
        new_m_tm = m_tm * -1
        tetha_rmvtm = abs(math.degrees(math.atan((new_m_rm - new_m_tm)/(1 + (new_m_rm * new_m_tm)))))

        new_m_tm = m_tm * -1
        new_m_lm = m_lm * 1
        tetha_tmvlm = abs(math.degrees(math.atan((new_m_tm - new_m_lm)/(1 + (new_m_tm * new_m_lm)))))

        # Calculates the total angles between each of the extreme points and accuracy of results
        total = tetha_lmvbm + tetha_bmvrm + tetha_rmvtm + tetha_tmvlm
        tetha_accuracy = (total/360) * 100

        # Draws the contour and centre point in black
        cv2.drawContours(thresh2, [cnt],0,(0,0,0),1)
        cv2.circle(im2,(cx,cy),1,(0,0,0),-1)
```

```
    cv2.imshow('Output', im2)
    cv2.imshow('Output Thresholded image of the mole', im2)
    cv2.imwrite('/home/pi/Image_Segmented_Threshold/Segmentated_Image_Thresholding_
      Marked_{:%Y_%m_%d_%H:%M}.jpg'.format(datetime.datetime.now()), thresh2)

print("Output for the program: ")
print("")
print("No. of contours in the image:         %d"  % len(contours))
print("Area of the mole (in pixels):         {0:.2f}".format(area))
print("Diameter of the mole (in pixels):     {0:.2f}".format(diameter))
print("Eccentricity of the contour:          {0:.2f}".format(eccent))
print("Orientation of the contour:           {0:.2f}".format(orientation),"degrees")
print("")
print("Contour's centre point:                                    ", (cx, cy))
print("Contour's leftmost point:                                  ", leftmost)
print("Contour's rightmost point:                                 ", rightmost)
print("Contour's topmost point:                                   ", topmost)
print("Contour's bottommost point:                                ", bottommost)

print("Distance from leftmost point to centre point (in pixels):     {0:.2f}"
  .format(distance_lm2centre))
print("Distance from rightmost point to centre point (in pixels):    {0:.2f}"
  .format(distance_rm2centre))
print("Distance from topmost point to centre point (in pixels):      {0:.2f}"
  .format(distance_tm2centre))
print("Distance from bottommost point to centre point (in pixels):   {0:.2f}"
  .format(distance_bm2centre))

print("Slope of line from leftmost point to centre point:            {0:.2f}".format(m_lm))
print("Slope of line from rightmost point to centre point:           {0:.2f}".format(m_rm))
print("Slope of line from topmost point to centre point:             {0:.2f}".format(m_tm))
print("Slope of line from bottommost point to centre point:          {0:.2f}".format(m_bm))

print("Angle between the leftmost point and bottommost point:        {0:.2f}"
  .format(tetha_lmvbm),"in degrees")
print("Angle between the bottommost point and rightmost point:       {0:.2f}"
  .format(tetha_bmvrm),"in degrees")
print("Angle between the rightmost point and topmost point:          {0:.2f}"
  .format(tetha_rmvtm),"in degrees")
print("Angle between the topmost point and leftmost point:           {0:.2f}"
  .format(tetha_tmvlm),"in degrees")
print("Total amount of the angles accounted for {0:.2f}".format(total),"in degrees")
print("Points were rounded to a integer number so a resionable error is acceptable")
print("Accuracy of angles {0:.2f}".format(tetha_accuracy),"%")

cv2.waitKey(0)
cv2.destroyAllWindows()
```

fyp_complete_distance_sensor.py

```
# Distance sensor

# Import the GPIO and the time library
import RPi.GPIO as GPIO
import time

# GPIO Mode (BOARD / BCM)
GPIO.setmode(GPIO.BCM)

# Set GPIO Pins
GPIO_TRIGGER = 23
GPIO_ECHO = 24

print('Distance Measurement In Progress')

# Set the GPIO direction (IN / OUT)
GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
GPIO.setup(GPIO_ECHO, GPIO.IN)

# Ensure the Trigger pin set to low, and give the sensor a second to settle
GPIO.output(GPIO_TRIGGER, False)
print('Waiting For Sensor To Settle')
time.sleep(2)

# Set Trigger to HIGH
GPIO.output(GPIO_TRIGGER, True)

# Set Trigger after 0.01ms to LOW
time.sleep(0.0001)
GPIO.output(GPIO_TRIGGER, False)

StartTime = time.time()
StopTime = time.time()

# Save StartTime
while GPIO.input(GPIO_ECHO) == 0:
    StartTime = time.time()

# Save time of arrival
while GPIO.input(GPIO_ECHO) == 1:
    StopTime = time.time()

# Calculate Time difference between start and arrival
TimeElapsed = StopTime - StartTime

# Calculation of the Dinstance in cms
Distance = TimeElapsed / 0.000058

# Prints the Distance
print('Distance: {} cm'.format(Distance))
```

```
# Clean the GPIO pins to ensure that all the inputs/outputs are reset
GPIO.cleanup()
```