University of Southern Queensland

Faculty of Health, Engineering & Sciences

# Smart Shaving Mirror

A dissertation submitted by

Peter Pitt

in fulfilment of the requirements of

**ENG4112 Research Project**

towards the degree of

**Bachelor of Engineering (Honours) (Electrical & Electronic Engineering)**

Submitted: October, 2020

# Abstract

Shaving of a face can be difficult for many people. This task can be made even more challenging for a person who encounters difficulty with either movement or vision. This project investigated the use of smart mirror technology to produce a mirror that would assist the user in the shaving process. The smart mirror was intended to intuitively provide an enlarged image of the area of the user's face that is being shaved. The selected region would be determined by tracking the movement of the shaving device being used.

Based on the literature review, three tracking algorithms were produced and tested. Two of these were based on the K-means algorithm. The first used the traditional method of K-means clustering, while the second implemented two different optimisation techniques discovered in the literature.

The first technique reused the cluster centres from the previous frame. This technique lowered the number of iterations required to complete the clustering and therefore, the computational time decreased. The second method checked if the Euclidean distance from the pixel to the cluster had increased as a test to determine if the pixel needed to have its cluster assignment recalculated. The combination of these two techniques resulted in a time saving of over 60% when incremental differences were present in the frames.

The third tracking technique implemented was the cross-correlation algorithm. The cross-correlation algorithm in its original form is very heavy computationally. However, through the use of the integral image technique, the algorithm operated with high efficiency.

Implementing the software in a more efficient language, such as C++, has the potential for completing a working smart mirror with the desired capabilities.

# Certification of Dissertation

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.
I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Peter Pitt

[redacted]

# Acknowledgments

I will start by thanking Dr Jason Brown for his supervision and direction within this project and also for coming up with the idea.

Thank you also to my current employer, Dart Switchboards, for being considerate and flexible when time off work was required to complete work.

Finally, and most importantly, I wish to thank my wife, Rebecca and three children, Isaac, Kading and Alyssa. Thank you for your understanding and acceptance of the many times that I was unable to participate in your lives in the way I would have liked. And thank you for the encouragement and support you have offered along the way. Not only during this year but throughout the eight years of study.

Peter Pitt

# Contents

**Abstract** **i**

**Acknowledgments** **iv**

**List of Figures** **ix**

**List of Tables** **x**

**Chapter 1 Introduction** **1**

1.1 Motivation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1

1.2 Context . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

1.3 Research Objectives . . . . . . . . . . . . . . . . . . . . . . . . . . 3

1.4 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

**Chapter 2 Literature Review** **5**

2.1 Smart Mirror History- Focused on Object Tracking . . . . . . . . . . . . 5

2.2 Tracking Algorithms . . . . . . . . . . . . . . . . . . . . . . . . . . 8

2.2.1 K-means Algorithm . . . . . . . . . . . . . . . . . . . . . . . . 9

2.2.2 Mean Shift Algorithm . . . . . . . . . . . . . . . . . . . . . . . 12

2.2.3 Cross Correlation . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15

**Chapter 3 Methodology- Software Developement 19**

3.1 Smart Mirror Platform . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20

3.2 Camera Interface . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21

3.3 Shaving Software . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23

**Chapter 4 Tracking Algorithm Developement 25**

4.1 Basic K-means Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . 25

4.2 K-means with Modifications . . . . . . . . . . . . . . . . . . . . . . . . . 30

4.3 Cross-Correlation using an Integral Image . . . . . . . . . . . . . . . . . 32

**Chapter 5 Hardware Development 34**

5.1 Required Components . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34

5.2 Arrangement of Components . . . . . . . . . . . . . . . . . . . . . . . . 36

**Chapter 6 Results 40**

**Chapter 7 Conclusions and Further Work 43**

7.1 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 43

7.2 Future Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44

**References 47**

**Appendix A Project Specification 53**

**Appendix B Basic K-means Tracking Algorithm 55**

B.1 Python Code for K-means Tracking . . . . . . . . . . . . . . . . . . . . . . . 56

**Appendix C Modified K-means Tracking Algorithm 59**

C.1 Python Code for modified K-means Tracking . . . . . . . . . . . . . . . . . 60

**Appendix D Cross-correlation Tracking Algorithm 64**

D.1 Python Code for Cross-correlation with Integral Image . . . . . . . . . . . 65

# List of Figures

1.1 A component diagram of a smart mirror (Jin et al. 2018). . . . . . . . . . 2

2.1 Augmented imaged produced using the mirror (Darrell, Gordon, Woodfill & Harville 1998) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

2.2 Mean shift clustered image (Comaniciu, Ramesh & Meer 2003) . . . . . . 9

2.3 Graphical representation of (a) a flat and (b) a Gaussian kernel (Cheng 1995). 13

2.4 Processing time using FGT (Elgammal, Duraiswami & Davis 2003) . . . . 15

2.5 Flow chart of correlation using the Fast Fourier Transform method (Hii, Hann, Chase & Van Houten 2006). . . . . . . . . . . . . . . . . . . . . . . . 17

2.6 Calculation of the correlation using an integral image (Viola, Jones et al. 2001) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18

5.1 Basic representation of the front view of the assembled smart mirror. . . . 38

5.2 Rear elevation of proposed smart mirror design. . . . . . . . . . . . . . . . 38

5.3 Keyhole shape for mounting holes. . . . . . . . . . . . . . . . . . . . . . . 38

7.1 Screen capture of FTOP system analyses tool. . . . . . . . . . . . . . . . . 44

# List of Tables

6.1 Tracking Algorithm Testing Results. . . . . . . . . . . . . . . . . . . . . . . . 41

# Chapter 1

# Introduction

Smart home technology has become more prevalent in today's society. While many of the functions and features offered by these devices are for convenience and entertainment value, the technology can also be used to provide genuine assistance in the daily lives of the users.

This dissertation documents the investigation into the utilisation of smart mirror technology to assist a person in the shaving of their face. To achieve the goals of this project, both hardware and software options were assessed.

## 1.1 Motivation

Shaving one's face can sometimes be a difficult process for some people. This task can be made simpler by improving the ability for the person shaving to see their face. A mirror with the ability to magnify an image of the region being shaved would assist in this process.

The proposed design will be able to track any device used for shaving, whether it is an electric razor or a traditional blade. The position of the shaving implement would be used to determine the area of the face that the user is shaving and display an enlarged image of that area. Once the user moved to another region of the face, the smart mirror would detect this and refocus the image on the desired area.

While this smart mirror will prove to be beneficial to any user, there will be added advantages for those who have physical limitations that make shaving difficult. For example, someone with impaired vision may be able to shave without the difficulty of wearing glasses when using this mirror. The mirror will also prevent the user from needing to lean in closer to check if their face has been adequately shaved, which would be useful for those with mobility issues.

## 1.2 Context

The move towards smart homes accelerated significantly in 2014 when Google purchased Nest LABS, a company focussed on designing smart devices (Xu, Wang, Wei, Song & Mao 2016). This acquisition began a revolution that has seen smart devices becoming common in the daily lives of many people.

One such smart device that has been created is the smart mirror, or commonly called a magic mirror. The smart mirror consists of a screen placed behind a one-way glass panel. When the screen is illuminated, the image being displayed is seen through the glass when viewing the mirror. However, when the screen is not illuminated, the display appears to be an ordinary mirror. Figure 1.1 shows the construction diagram of one such smart mirror produced by Jin, Deng, Huang & Chen (2018). This design incorporated an infrared frame that was used for the touch screen capability of the mirror.

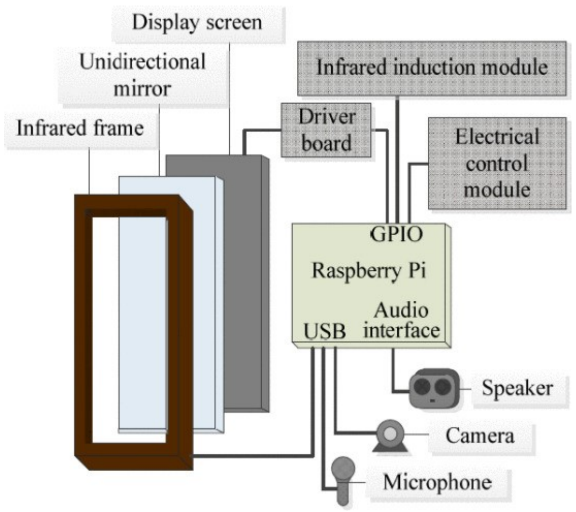


Figure 1.1: A component diagram of a smart mirror (Jin et al. 2018).

As the popularity of smart devices has increased, so has the interest in smart mirrors.

This interest has been demonstrated by the number of smart mirrors being produced, whether it is for commercial purposes, research purposes or by hobbyists (Gold, Sollinger & Indratmo 2016).

Most of these homemade smart mirrors possess the same basic functions. These include the ability to watch online content, such as YouTube, listen to music, view the latest news headlines, check the weather and traffic conditions and use social media platforms. Many also provide internet search results using a speech recognition interface such as Amazon Alexa or Google Assistant. While these are only a few of the functions these smart mirrors can perform, they are generally no different from what can be achieved using a tablet or a smartphone.

Academic literature on smart mirrors demonstrates the technology being used for more advanced functions, additional to those mentioned above. One such use for a smart mirror is to provide health tracking of the user. Colantonio, Coppini, Germanese, Giorgi, Magrini, Marraccini, Martinelli, Morales, Pascali, Raccichini et al. (2015) presented a smart mirror that assessed the users face over some time to determine their risk of cardio-metabolic health problems. In response to this, advice and encouragement would be given to make changes to their lifestyle to reduce the health risk.

Another smart mirror intended for assisting in maintaining the health of the user was created by Rahman, Iyer, Meusburger, Dobrovoljski, Stoycheva, Turkulov, Begum & Ahmed (2016). This mirror could determine the user's heart rate, respiratory rate, inter-breath-interval, blood pressure and drowsiness. From this assessment, information was given that was intended to assist in providing the user with appropriate medical treatment.

The examples above are given as a brief introduction to the functions a smart mirror can perform. A more in-depth analysis of the current state of smart mirror research has been undertaken in the literature review section.

## 1.3 Research Objectives

The research required for this project will involve an investigation into multiple areas. A thorough examination of the current state of smart mirror research has been included. Research into tracking algorithms was conducted with the aim of determining the most appropriate algorithm for use in the project. The goal was to find an algorithm to be implemented that worked while maintaining a low computational load and providing a robust method of determining the location of the shaving device baing used.

A design for the construction of the smart mirror was developed. This design was completed to provide a method of mounting all components of the smart mirror. As this is not a new design concept, existing smart mirror designs were closely followed. However, there were some alterations made to these designs to accommodate the inclusion or the Raspberry Pi camera.

## 1.4 Conclusion

While the uses of a smart mirror discussed previously are innovative and useful, they fail to maximise the original use of a mirror. Traditionally, a mirror was a device used to observe a reflection. While this can be done in these mirrors, it is not their primary use and therefore the application of these other functions to a mirror is not utilising the mirror for its initial and primary purpose. The smart mirror proposed in this document is using this new technology to enhance the user's ability to see a reflection in a mirror. In this case, the mirror will display an enlarged image of the facial region the user is currently shaving. The control of the mirror and zoom function will need to be intuitive and have the ability to acquire the target region with little or no input from the user. The smart mirror will determine the target area by tracking the position of the shaving implement being used.

This smart mirror, while useful for anyone that wishes to use it, would hold a more significant benefit for people with vision or mobility impairment. The magnified image will allow those with limited vision to see the image being reflected more clearly. It would also help those who find leaning close to the mirror challenging to get a closer look at their face without needing to move.

# Chapter 2

# Literature Review

## 2.1 Smart Mirror History- Focused on Object Tracking

In 2000, Meine (2000) lodged a patent for what he referred to as an ‘inventive mirror’. The proposed device was controlled using a touch screen, a mouse and a keyboard. The user could use the System to view their schedule, read news articles and read and respond to emails. By using the mirror for this purpose, the user could save time by performing these tasks simultaneously with personal grooming.

Since this time, there has been a great deal of research into the use of smart mirrors to assist the user in their daily activities and improving their physical and emotional wellbeing. A variety of methods of control have been implemented in the mirrors. Modern smart mirrors are no longer controlled using a keyboard and mouse. These have been replaced by voice control, touch screen, hand gestures or a combination of these. One mirror that has used all three of these control mechanisms is named ‘Eve’ (Bonnain 2018). ‘Eve’ operates on an android platform and has an app store built-in so the user can download and run over 500 apps on the mirror. There are many applications ‘Eve’ can be used for including games, social media, news headlines, watching online content, and listening to music. These types of functionality are not unique and have been incorporated into many such projects, including having the mirror integrated into the top of a coffee table (Channu, Bheemashappa & Sudharshan 2019).

While projects such as ‘Eve’ demonstrate the versatility of smart mirrors, they do not provide any functionality that cannot be achieved just as easily using a tablet. Yu, You & Tsai (2012) produced a smart mirror table that did not only have the standard smart mirror applications but was also able to determine the emotional state of the user based on facial expressions. Algorithms were used to examine an image of the user's face captured by a camera incorporated into the mirror. If the person was deemed to be in a negative mood, the mirror would play pre-recorded encouraging messages in the voice of a close friend or family member. The user's favourite music would accompany these messages.

Bianco, Celona & Napoletano (2018) also produced a smart mirror that we able to detect the emotions of the user. A Convolutional Neural Network algorithm was used to identify 68 facial landmarks was that are used for facial identification. A multitask learning approach based on a convolutional neural network was then used to estimate the age, gender and as many as 40 facial attributes. These attributes include sideburns, hair colour, eyebrow shape, the presence of glasses and earrings that are being worn. The mirror created a log of the user's emotional stated and analysed them over time to give a distribution of the emotions detected for each user.

Mirrors using facial recognition were common in the literature reviewed with many using it to customise the mirror for the specific user (Ding, Huang, Lin, Yang & Wu 2007, Gold et al. 2016, Jin et al. 2018). Hossain, Atrey & Saddik (2007) chose to use facial recognition as a security feature for the mirror. As the mirror was able to control other smart devices within the home, there was a need for access to these functions to be restricted to only authorised people. Access was granted only once the user was identified by both facial and voice recognition.

The ability to identify a human face in the image captured was also used by Darrell et al. (1998). This mirror was designed to capture a picture of the person standing in front of the mirror and return an augmented image of the person's face, as shown in Figure 2.1. A tracking system was implemented to enable the mirror to continue the facial augmentation even when the subject was moving.

To track the face of the user, Darrell et al. (1998) implemented a threefold approach. First, the census correspondence algorithm was implemented. This algorithm began by converting the image to greyscale then performing a census transform. The transformed images from two separate cameras were examined to give an accurate estimation of the distance of different regions within the images. The closest region that is approximately

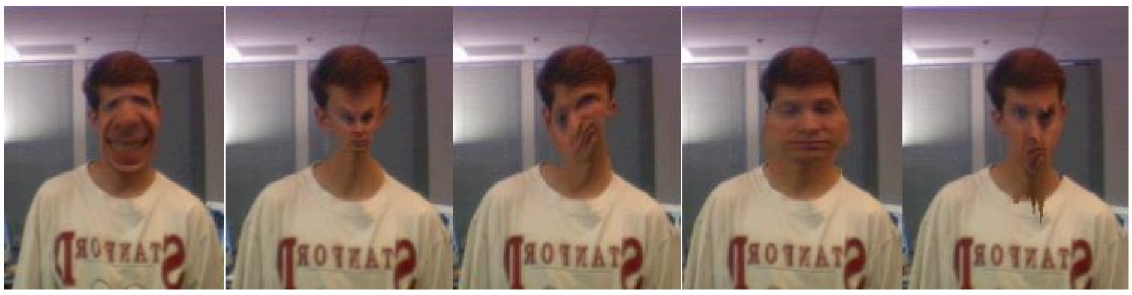


Figure 2.1: Augmented imaged produced using the mirror (Darrell et al. 1998) .

the size of a human face was then identified and tracked until it was no longer in the defined area.

Skin hue classification was then used to determine the regions of the identified object that were likely to be bare skin. Values were calculated using the red blue and green colour data from each pixel, and a Gaussian probability model was implemented to rate the likelihood of the pixel being skin coloured. To reduce the time taken to calculate the values, a lookup table was prepopulated with all possible combinations of the red, green and blue values, and the rating is determined by examining this table.

Finally, face pattern discrimination was employed to distinguish users faces from other skin-toned regions or arias misidentified as bare skin. The implementation of this was through the use of the CMU Face Detector Library, developed by Rowley, Baluja & Kanade (1998). This algorithm relied on pattern recognition and was trained using faces as well as entering false positives back into the algorithm to prevent a repetition of the error.

There are many instances where smart mirrors have been constructed with similar functionality to the one proposed in this document. They have been designed for use when applying makeup. One such mirror was designed and built by Iwabuchi, Nakagawa & Siio (2009). However, it was not presented as a smart mirror. Instead, it was a screen with cameras and proximity sensors attached. The mirror could zoom in on the users face when a makeup tool was brought close to the user's eyes. The location of the makeup tool was tracked by following a green sticker that we placed on it. A dynamic zoom function was also implemented that zoomed in as the user came closer to the mirror. An infra-red range sensor determined the distance of the user. There was no technical information provided regarding the development of the mirror or on the algorithms used for tracking or detection of the user.

Treepong, Mitake & Hasegawa (2018) also created a mirror intended to assist the user in the application of makeup. Instead of showing the actual user's face as makeup was being applied, the mirror would display a computer-generated three-dimensional image. The user would be able to see how different styles and method of makeup application would look by using makeup tools linked to the mirror. Virtual makeup would be applied to the image instead of the user and made it possible to try different makeup products before purchasing them. A Microsoft Kinect camera and retro-reflective stickers attached at each end of the tools were used for tracking the object. They also had a conductive material on the tip, which was used to detect contact with the user's face. The Kinetic camera was also responsible for detecting the position and movement of the face.

The idea of applying virtual makeup to a three-dimensional computer-generated image instead of the user was not unique to Treepong et al. (2018). Rahman, Tran, Hossain & Saddik (2010) also produced a mirror with these capabilities. This mirror was similar to the one made by Iwabuchi et al. (2009) since it did not represent a mirror but was a screen displaying an image. However, instead of using the Kinetic system to perform the tracking of the makeup tools, infra-red emitters were attached to them. The positioning of the face was also determined using infra-red transmitters attached to earrings being worn by the user. This mirror did not offer any magnification or zoom capability.

The literature mentioned above shows that a considerable amount of work has been completed regarding tracking by smart mirrors. However, in each of these cases where an object is being tracked, it is a marker or transmitter that the algorithm is trained to follow, not the object. Using these methods limits the number of items that can be tracked. Not all people use the same implement for shaving as there are many forms that a razor can take. The ability of a mirror to track the users shaving implement, regardless of their choice, and without needing markers or transmitters, would be highly advantageous to this design.

## 2.2 Tracking Algorithms

To determine the most appropriate method of tracking for the mirror, the tracking algorithms that are currently being used were examined. The algorithms mentioned in this section do not provide an exhaustive list as the number of tracking algorithms available is too significant to be covered in sufficient detail within this document. However, the algorithms discussed here are those that appear to be best suited for the desired application.

The first two algorithms discussed are both clustering algorithms. A clustering algorithm, when used for tracking or image analyses, is an algorithm that groups, or clusters, regions of an image according to a specific property. Clustering algorithms can be classified into two categories (Oliver, Munoz, Batlle, Pacheco & Freixenet 2006). These are hierarchical and partitional algorithms.

An example is shown in Figure 2.2 below. In this figure, an image that has been analysed with a mean shift clustering algorithm is presented (Comaniciu et al. 2003). The original image is displayed on the left while the clustered image is displayed on the right. Each cluster is distinguished with a white outline.

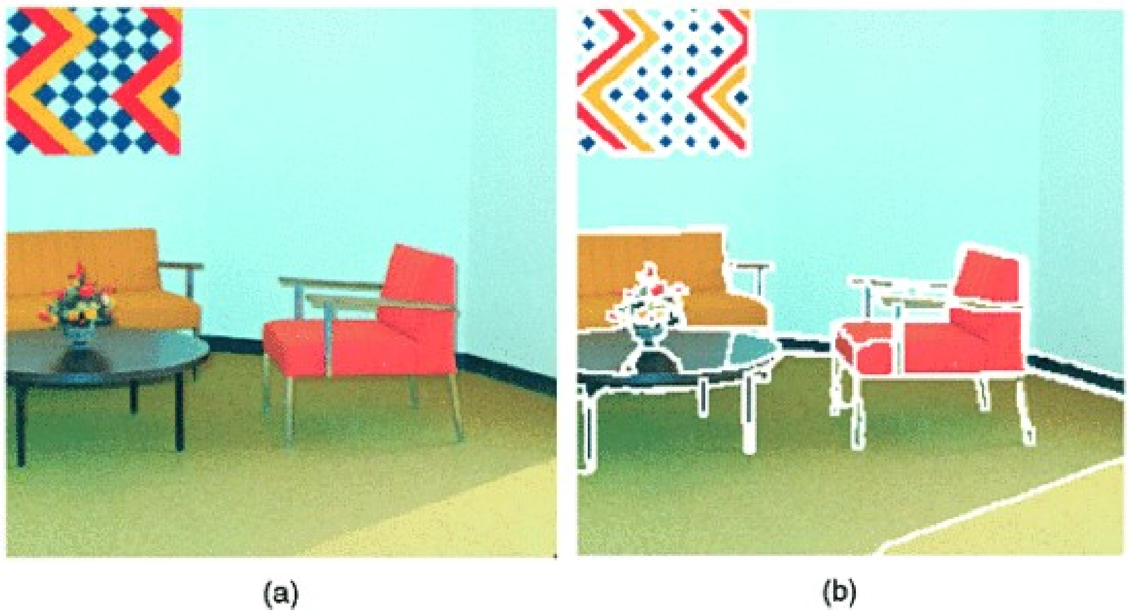


Figure 2.2: Mean shift clustered image (Comaniciu et al. 2003) .

### 2.2.1 K-means Algorithm

K-means clustering was first developed as a data clustering system in 1967 (Na, Xumin & Yong 2010). Since then, it has been applied to image clustering in literature numerous times. The K in the name refers to the number of clusters that are designated to be created within the image. This can create a challenge as this number must be determined before the clustering algorithm is begun. Therefore, it is beneficial if specific properties are known about the image before the process is started.

The K-means algorithm aims to reduce the total mean square error between the cluster centroids and the mean of the pixel values within the respective clusters(Na et al. 2010). This process is represented as:

$$E = \sum_{i=1}^{k} \sum_{x \in C_i} \left| \mathbf{x} - \mathbf{x_i} \right|^2 \tag{2.1}$$

where $E$ is the mean square error, $k$ is the number of clusters being used, $\mathbf{x}$ is the values of the pixel assigned the cluster, and $\mathbf{x_i}$ is the centre value of the cluster. These values take the form of a one by three vector due to the image data being stored in the red, green and blue values for each pixel.

To achieve the minimisation of the mean square error, there are a series of steps that are repeated until the cluster centres are found (Tatiraju & Mehta 2008). First, the desired number of clusters is determined. For each cluster, a seed is created by generating a vector of three random values.

Once this is completed, each pixel is assigned to the cluster with the lowest Euclidean distance between the pixel value and the seed (Steinley 2006). The Euclidean distance is calculated by treating the two sets of values as position vectors and determining the distance between them, as shown below:

$$d\left(\mathbf{x}, \mathbf{x_i}\right) = \sqrt{\sum_{n=1}^{3} \left(x_n - x_{in}\right)^2} \tag{2.2}$$

The pixels are then assigned to the nearest cluster. The mean values for the red, green and blue of the are then calculated for each pixel assigned to the cluster. The result then replaces the seads as the cluster centre. The process starts again by reassigning the pixels to the new cluster centres in the same way as before. This process is repeated until the mean of the clusters remains the same.

Ray & Turi (1999) identify one of the most significant weaknesses of the K-means algorithm as being the need to predetermine the number of clusters to be created. In their research, they presented a method of automatically calculating the number of clusters that would give the best result. Their approach was to produce multiple clustered images with varying numbers of clusters using the K-means algorithm. They would then calculate the total mean square error, as shown above, and then divide it by the smallest distance between the cluster centres. The best selection of cluster numbers was the one that gave the lowest number from this process.

This method was mostly successful in determining the best possible number of clusters to be used in the clustering process. If this method were implemented as part of the tracking algorithm, it would create too high a computational load and therefore slow the algorithm more than would be acceptable. There would be a possibility of implementing this method into the initial acquisition of the shaving device. Since the smart mirror is to be used for shaving, it will generally be used in a bathroom or at least in the privacy of one's house. It is unlikely that the background of the images captured when the mirror is in use would change at a great extent. The changes in the foreground will be the movement of the user as they are shaving. For this reason, there will be little changes from one image to the next. This will allow for the number of clusters used to be set at the beginning and remain the same throughout the process.

The computational load of an algorithm is not critical when used for analyses of still images. However, when the algorithm is implemented for visual tracking, it needs to run repeatedly in a limited timeframe. Fahim, Salem, Torkey & Ramadan (2006) created a K-means based algorithm that enabled the number of calculations required to be reduced. They called it the enhanced K-means algorithm, and it reduces the computational complexity from $O(nkl)$ to approximately $O(nk)$ where $n$ is the number of pixels in the space being examined, $k$ is the number of clusters, and $l$ is the number of iterations required to complete the clustering.

Their algorithm worked by calculating the Euclidean distance between each pixel and the cluster centre to which is already assigned. This calculation is completed after the update of the cluster centre using the mean of the pixel values. If the distance is equal to or less than the distance to the old cluster centre, the cluster remains assigned to that cluster. If the cluster has moved further away from the pixel being examined, the distance to the other centres is then calculated, and the pixel is reassigned to the cluster to which it is now closest.

The results of this modification of the algorithm were considered successful. The processing time for the algorithm was reduced by up to 80% without compromising on accuracy or reliability.

Heisele, Kressel & Ritter (1997) also worked on improving the efficiency of a k-means tracking algorithm. Their research was used for following multiple objects in a series of images captured by a moving camera on a vehicle. The new method they proposed worked on the assumption that each picture in a video stream will only have an incremental difference from the previous image. Due to this, the cluster centres found for one frame can therefore be transferred as the seed points for the following image. The number of iterations required to reach the final clusters was significantly reduced as a result. Higher accuracy in the tracking of an object is also possible as varying the starting seed positions usually results in different final cluster positions. Using the cluster centres from the previous image ensures that the clusters are as similar as possible for each frame.

The results of testing of the proposed algorithm were successful with the tracking of pedestrians and vehicles proving to be robust and efficient. It proved that this method was successful in tracking non-rigid objects and handling occlusions.

### 2.2.2 Mean Shift Algorithm

The mean shift algorithm was first developed in 1975 by Fukunaga & Hostetler (1975). It is a non-parametric method of determining the modes of a probability density function (PDF). Using a selected similarity measure, the PDF shows the likelihood of a region of pixels containing specific properties (Fisher, Breckon, Dawson-Howe, Fitzgibbon, Robertson, Trucco & Williams 2014). This function can be used for clustering regions of similar pixels, and by examining the local maxima, it can be determined if that group of similar pixels represent the position of the tracked object in the image.

The mean shift differs from other mode finding algorithms, in that the PDF is not calculated but instead estimated (Szeliski 2010). Instead of creating a PDF for the whole image and then finding the mode, the mean shift function predicts the gradient of the PDF at a given point and then employs a hill-climbing technique to find the modes. This method means that when an area of an image, also known as a kernel, is examined, the mean shift algorithm returns the gradient of the PDF at the centre of the kernel. The gradient is determined by calculating the location of the mean of the values within the kernel. The centre of the kernel is then set to be the mean of the pixels contained within the kernel. The kernels are merged with adjacent kernels that exhibit mean values that are close to each other, and the mean of the new kernel is recalculated. At this point, the local mode of the PDF has been found.

The equation for the mean shift algorithm generally takes the form:

$$m\left(x\right)=\frac{\sum\limits_{s\in S}K\left(s-x\right)s}{\sum\limits_{s\in S}K\left(s-x\right)} \tag{2.3}$$

where $K$ is the kernel function, $S$ is the kernel, $x$ is the centre value of the kernel and $s$ is each of the pixels within the kernel (Cheng 1995).

As previously stated, the kernel is the area of the image being examined. The kernel function applies a weighting to each of the values within the kernel with respect to their distance from the centre of the kernel. The selection of the kernel function to be used affects the performance of the mean shift algorithm. Szeliski (2010) compared the Epanechnikov kernel function with the Gaussian kernel function. The Gaussian kernel produces a smoother result but at the expense of speed when compared Epanechnikov kernel. These are only two of many kernel functions that are used when implementing a means shift algorithm. Figure 2.3 below shows a graphical representation of the weighting of a flat and a Gaussian kernel.

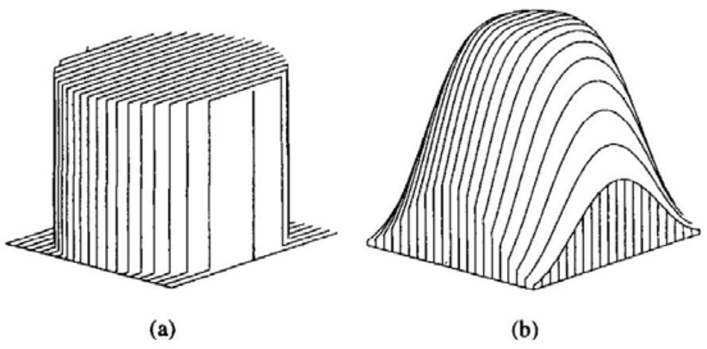


Figure 2.3: Graphical representation of (a) a flat and (b) a Gaussian kernel (Cheng 1995).

The flat kernel in the image above is truncated around the centre at a distance equal to the bandwidth ($\lambda$). The equations for these kernel functions ($K\left(x\right)$) are displayed below.

Equation 2.4 represents the flat kernel while Equation 2.5 is used for a Gaussian kernel:

$$K(x) = \begin{cases} 1 \text{ if } \|x\| \leq \lambda \\ 0 \text{ if } \|x\| > \lambda \end{cases} \tag{2.4}$$

$$K(x) = e^{-\beta\|x\|^2} \tag{2.5}$$

where $\beta$ represents the bandwidth of the Gaussian curve.

Mean shift clustering has been used extensively for tracking applications due to its robust nature, ability to be used for non-rigid objects and ease of implementation (Leichter, Lindenbaum & Rivlin 2010). However, the main disadvantage of this method is the large number of calculations required. The computational load created by this has made it impractical for use in real-time tracking without the inclusion of large amounts of computing power. As a result, there has been a great deal of research completed into methods of reducing the computational load of the algorithm.

Greengard & Strain (1991) manipulated the Hermite and Taylor series to create what they called the Fast Gauss Transform (FGT). Their work enabled the number of calculations required for each iteration of the Gaussian kernel to be reduced from $O\left(N^2\right)$ to $O(2N)$ where $N$ represents the number of pixels in the kernel. Elgammal et al. (2003) successfully took this method and applied it to improve the processing time required for the mean shift algorithm. In 2003, when their research was published, the FGT was a relatively new mathematical method, and they considered it to be revolutionary in the way that it was changing the way that numerical analysis was being undertaken.

The graph in Figure 2.4 demonstrates the savings in the time required to complete the evaluation using the mean shift. The results that were given state that once the number of points to be assessed reaches between 60 and 80 points, it becomes more efficient to use the FGT than directly applying the Gaussian kernel and assessing the image. This value depends on the degree of accuracy desired, which is determined by the number of terms retained during the series transforms.

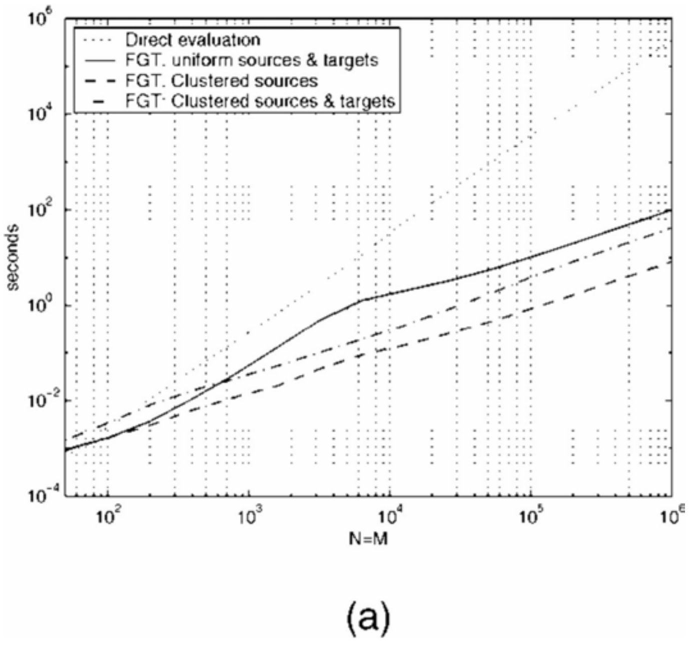


Figure 2.4: Processing time using FGT (Elgammal et al. 2003) .

### 2.2.3 Cross Correlation

Correlation-based tracking works by taking a sample image of the object to be tracked, called a template, and comparing it with the picture in which it is to be found. Szeliski (2010) notes that using correlation for tracking requires image edges or changes in pixel properties within the template to be effective as opposed to clustering algorithms which require relative uniformity within sections of the target object.

Cross correlation is performed by calculating the correlation between the template and a region of the image using correlation (Hii et al. 2006). The area that exhibits the highest correlation value is considered to be the position of the object. The template needs to be tested at each possible position within the image. At each location, the correlation is calculated (Sebastian & Yap Vooi Voon 2007). The following formula describes this calculation for finding a $N_x$ by $N_y$ template within a $M_x$ by $M_y$ image:

$$c\left(u,v\right) = f\left(x,y\right) t\left(x-u, y-v\right) \tag{2.6}$$

where $t\left(x,y\right)$ is the template, $f\left(x,y\right)$ is the image being examined, $u \in \{0,1,2,...,M_x - N_x\}$, $v \in \{0,1,2,...,M_y - N_y\}$ and are used to indicate the position of the template within the image.

The most commonly implemented variant of correlation tracking is Normalised Cross Correlation (NCC). By normalising the light intensities in the image, the tracking algorithm becomes more robust as it is able withstand variations in lumination and shadows. Normalisation also provides a higher resilience to noise within the image.

When implementing the NCC algorithm for a template with the dimensions $N_x$ by $N_y$ within an image with the dimensions $M_x$ by $M_y$, a normalised cross correlation matrix is calculated using the following formula (Hii et al. 2006):

$$\gamma(u,v) = \frac{\sum_{x,y}\left[f(x,y) - \overline{f}_{u,v}\right]\left[t(x-u,y-v) - \overline{t}\right]}{\sqrt{\sum_{x,y}\left[f(x,y) - \overline{f}_{u,v}\right]^2 \sum_{x,y}\left[t(x-u,y-v) - \overline{t}\right]^2}} \tag{2.7}$$

where $\overline{t}$ is the mean of the template and $\overline{f}_{u,v}$ is the mean of the region of the image being examined.

While the NCC algorithm is considered a reliable, robust method of tracking, it does come with challenges. The computational load makes it impractical for real-time tracking purposes. The number of calculations required is in the order of $N_xN_y\left(M_x - N_x\right)\left(M_y - N_y\right)$. Tracking of a non-rigid object also poses a challenge for NCC as rotations or variations in the shape or size of the object reduces the similarity to the template.

One commonly used method for reducing the computational load of the NCC algorithm is by implementing a Fast Fourier Transform (FFT) for both the image and the template. The numerator used in Equation 2.7 can be calculated more efficiently in the frequency domain. Since correlation in the spatial domain is equivalent to multiplication in the frequency domain, the process can be expressed as:

$$\begin{aligned} & r(u,v) = \sum_{x,y} f(x,y)\, t(x-u,y-v) \\ & \Downarrow \\ & R(u,v) = F(u,v)\, T(u,v) \\ & \Downarrow \\ & r(u,v) = \Im^{-1}\left(R(u,v)\right) \end{aligned} \tag{2.8}$$

As a result of this, the number of calculations is reduced to the order of $M_xM_y \log\left(M_xM_y\right)$. The process is demonstrated in Figure 2.5.

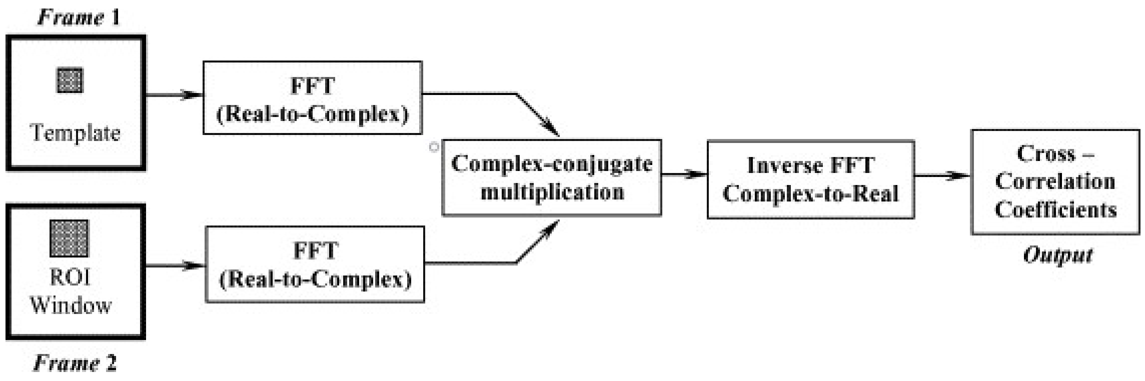

Frame 1
Template
ROI Window
Frame 2
FFT (Real-to-Complex)
FFT (Real-to-Complex)
Complex-conjugate multiplication
Inverse FFT Complex-to-Real
Cross – Correlation Coefficients
Output

Figure 2.5: Flow chart of correlation using the Fast Fourier Transform method (Hii et al. 2006).

A further reduction in the processing load of the correlation algorithm was achieved by Viola et al. (2001). Their work made it possible to produce polynomials for each pixel that provided an approximation of the correlation coefficient at that point. The polynomials are found by implementing the integral image of the picture in which the object is to be found. The integral image is of a $u$ by $v$ image is expressed as:

$$I(u,v) = \int\limits_{x=0}^{u}\int\limits_{y=0}^{v} f(x,y)\,dxdy \tag{2.9}$$

where $f(x,y)$ is an integrable function of the image values.

The discrete representation of this is:

$$I(x,y) = \sum_{x'=0}^{x}\sum_{y'=0}^{y} f\left(x',y'\right) \tag{2.10}$$

The following formula is used to implement this in practical terms:

$$s(x,y) = s(x,y-1) + f(x,y)\,, I(x,y) = I(x-1,y) + s(x,y) \tag{2.11}$$

where $s(x,y)$ is the cumulative column sum. The resulting integral image at any point $(x,y)$ gives the sum of the pixels above and to the left of that point.

To find the position of the template within an image, the integral image is first calculated for both the template and image being examined. The value of the integral image of the template is compared to the image to determine where the closest fit is. In Figure 2.6 below, it is shown how the correlation is determined when the integral image of the template is compared to the integral image of the picture. The formula is:

$$D_{correlation} = 4 + 1 - (2 + 3) \tag{2.12}$$

where the numbers represent the integral image values as shown in Figure 2.6. The region marked as $D$ is the location of at which the template is being tested.

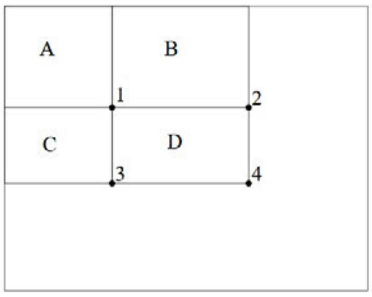


Figure 2.6: Calculation of the correlation using an integral image (Viola et al. 2001) .

As can be seen, once the integral image has been calculated for both the template and the picture, the calculations required for each template comparison is significantly reduced. Some accuracy is lost in the comparison as this method only provides an approximation of the correlation. However, the results of initial testing were positive. When this method was used for detection of faces, there was close to 100% detection rate and was approximately 15 times faster than other algorithms to which it was compared.

# Chapter 3

# Methodology- Software Developement

The completion of this project required the work to be divided into two categories- software and hardware. By organising the project in this manner, it was easier to track the progress of tasks and ensure that other workstreams were not delayed by other incomplete work. Tasks required for the project often overlapped in their execution, or needed to be completed before other tasks could be started. They are addressed separately with this and the following two chapters. This chapter is dedicated to the development of the software. The next chapter covers the tracking algorithms and the chapter following that details the hardware component.

In the initial stages of the project, it became apparent that the software development would heavily depend on the selection and implementation of a tracking algorithm. This process was achieved by the examination of previous academic work to discover how tracking algorithms had been integrated into smart mirrors. A substantial volume of research relevant to this was found and is summarised in the literature review section. As is also stated there, none of the literature found contained details of a mirror using pure visual tracking from a single camera. All other designs required more complex hardware options. It was decided that for simplicity and cost-effectiveness, a tracking algorithm would be implemented that only needed a feed from a single video stream.

Due to a lack of knowledge or previous experience with tracking algorithms, a review of the literature covering this topic was undertaken. The amount of research completed and the number of algorithms in existence was found to be far too numerous for an in-depth investigation of them all. For this reason, many algorithms were viewed briefly. However, only those that appeared on the first investigation to be appropriate were considered and researched more thoroughly.

It was decided that existing smart mirror software would be selected to serve as a platform on which this mirror was to be built. This decision was made for two reasons. The first is that this is not a new concept, and the required software had already been produced. The second factor influencing the decision was the time restraints of the project. To create a smart mirror program would require an extensive amount of work and therefore limit the working time to be sent on other aspects of the project.

Multiple viable options were found that met the required criteria. The software needed to be opensource and modular in design. Being opensource would make the software readily available and remove copyright concerns when implementing changes to the program for use in the project. Modular software was also advantageous as it would simplify the process of adding the required module and facilitate in removing unwanted features.

Before any other aspect of the program could be implemented, a method of controlling the camera was required. There are many different packages and libraries available to facilitate the control of the camera. However, many of these were found to be ineffective as problems were encountered in both installation and operation.

## 3.1 Smart Mirror Platform

Finding opensource smart mirror platforms that were available online was not a difficult task. A search on GitHub.com for ”smart mirror” returned over 2500 results (*GitHub search results- smart mirror* 2020). Many of these platforms provided the same basic functionalities and even presented a very similar appearance. After considering many different platforms and reading reviews, it was decided that the MagicMirror$^2$ platform was the most appropriate.

This software was developed by Michael Teeuw, who is credited as being the first to create an opensource smart mirror software which was called MagicMirrror (Teeuw n.d.). The original MagicMirror platform has been superseded by MagicMirror$^2$. This application was by far the most utilised of those found on GitHub. Github uses a star system to allow people to both follow a project listed on the site and show appreciation for the work of the programmer. MagicMirror$^2$ had over 12,700 stars attributed to it while the next closest smart mirror platform, smart-mirror by Even Cohen, had less than 20% of that number. It has also been voted the best Raspberry Pi project by the MagPI, the official Raspberry Pi magazine (*The MagPi- Issue 50* 2020).

Out of the smart mirror platforms examined, MagicMirror$^2$ was the only program that was genuinely modular in its design. The selected modules for the software are placed in a folder and then referenced within the configuration file. The range of modules available for the platform is extensive and is not rivalled by any other of the available programs. There are over 1000 modules available for MagicMirror$^2$ and these account for a considerable proportion of the over 2500 smart mirror search results found on GitHub. This high number of modules is due to the approach that Mr Teeuw has taken with the project. He has not only made the software available on GitHub but also set up a separate webpage containing a great deal of information on the platform. This website includes detailed instructions on the installation of the software and installation and configuration of the different modules. A list of modules is also found on the website with a description for each. Within this list, there is a template and instructions included to assist developers in creating new modules. All of the features mentioned above made the selection of MagicMirror$^2$ the logical choice to be used as a platform for this project (Teeuw 2016).

MagicMirror$^2$ was able to be installed and run on a Raspberry Pi during the project. However, many of the additional modules did not. With many of the modules being as old as six years, some relied upon superseded libraries and outdated dependencies and therefore required updating before they were functional.

## 3.2 Camera Interface

Most smart mirror software utilises existing web browsers to display the image on the screen (Gold et al. 2016). This method leads to the projects being written in JavaScript as the browsers can execute the code directly. Mr Teeuw (2016) used JavaScript to produce MagicMirror$^2$. It was therefore desirable to implement the camera capture and display using JavaScript.

Research into methods of accessing the video stream to perform the tracking revealed that there were multiple JavaScript libraries created to control the camera. However, few of those examined provided a method of accessing the data stream in which the tracking algorithm could be used. Most were written to take the input from the camera either save it to the Raspberry Pi or stream it to a website or a remote computer. Others only supported still image capture. Attempts were made to install multiple of these libraries that promised to fulfil the requirements; however, none were successful. Each module that was installed either failed in the installation process or received errors during implementation.

After unsuccessfully trying to interface with the camera using JavaScript, a decision was made to achieve this using Python. Despite the MagicMirror$^2$ platform being written in JavaScript, many of the available modules produced to run on the platform contained Python scripts. These scripts were called by the modules written in JavaScript. An example of this is the MMM- Selfie module (de Tena Rojas 2017). The configuration of the module and all user interfaces are implemented using JavaScript, but whenever the camera is accessed, a Python script is called to achieve this. As a result, it was decided to implement the camera access and tracking algorithm using Python.

Picamera offers a python interface for connecting with the Raspberry Pi camera that is widely implemented within the programming community. All modules that were examined relied on Picamera for access to the raspberry pi camera. The Raspberry Pi official website lists the Picamera as the library to be used for programming with the camera (Raspberry Pi Foundation n.d.*a*). The full documentation of the library can be found online, including programming examples (Jones 2016).

Picamera offers an extensive list of functions for capturing both still images and video from the Raspberry Pi camera module. While many of these were not used within the project, some were. Picamera allows for multiple configurations of the camera capture to be set using the splitter function. This function enabled the capture of two images with dissimilar properties to be taken in close succession. These images could be used for different purposes. The advantage of this was seen when a low-resolution image in the YUV colour format was required for tracking at the same time as a high-resolution image in RGB colour format was needed for display. More detail of the application of these functions will be provided in the section detailing the development of the tracking algorithms.

## 3.3 Shaving Software

While the implementation of the shaving module for the MagicMirror$^2$ platform has not been completed, the basic functionality of the shaving software has been designed. There are three main functions of the software that must be considered when writing the program. These are:

- The method of acquiring the object to trace
- Determining the region of the face to be magnified
- The method of selecting the degree of magnification.

The intent was for the smart mirror to be able to track any object the user decided to use as a shaving implement. It was decided that the simplest way to achieve this was to have a timer that ran each time the shaving module began. This timer would count down for a period of 5 seconds at the end of which, the image captured would be used for acquisition.

Depending on the tracking method used, the mirror would display an identifying mark in the centre of the captured image, demonstrating where the shaving utensil was to be held for acquisition. For the K-means method, crosshairs would be displayed indicating the user was to position the shaving tool in the centre. The clustering algorithm would then be run when the timer finished. Once the clusters were found, as described in the next chapter, the cluster containing the pixel in the centre of the crosshairs would be identified as the target cluster. For each frame after this, the geographical centre of the target cluster would be regarded as the central location of the object.

The pixel values for each image were stored as a three-dimensional array in the program. In Python, the order of the indices are rows, columns and layer. By finding the mean of the first and second indices for each pixel within the target cluster, the location of the cluster centre would be determined.

In a similar manner, when using the cross correlation algorithm with the integral image, a box would be displayed in the image indicating the location from where the template would be taken. On completion of the timer, the integral image of the template would be calculated and stored for use in subsequent frames.

Moving the area of the face that the program is magnifying with each frame would make it difficult for the user to gain a clear view of progress when shaving. For this reason, the smart mirror needs to select a region of the image that is to be magnified for display until the user is ready to proceed to another part of the face. The simplest method of achieving this would be for the user to hold the shaving implement stationary in the position they are about to shave. When the program detects that there has been little or no movement for a predetermined time, the centre of focus would be shifted. The mean of the positions that the shaving instrument has been held during this period would be calculated, and this would become the new centre of the magnified display.

# Chapter 4

# Tracking Algorithm Developement

Within this chapter, the development of the tracking algorithms will be discussed. The three algorithms that were developed are addressed separately in the order in which they were written. The basic K-means algorithm was written first and used as a platform for the development of the other algorithms. As a result, the communication with the camera and display of the image, along with other features, utilised the same string of commands in all three algorithms.

## 4.1 Basic K-means Algorithm

After an initial understanding of the MagicMirror$^2$ software was acquired, it was decided that implementing the tracking algorithm was the next logical task to complete. The algorithm was written and tested as a standalone program as proof of its effectiveness.

Due to having never used Python before, code for a simple program to receive data from the camera and display it on the screen was found online and used as a framework on which the program was to be based. The code used was written by Adrian Rosebrock (2015) and demonstrated the correct methods for setting the parameters to be used by the camera. The code also utilised the OpenCV library to display the image captured by the camera. In the MagicMirror$^2$ platform, all display features are handled by JavaScript code. Therefore, OpenCV will not be used in the final implementation of the mirror as it is a Python library. It was, however, utilised in the standalone tracking programs for display and development purposes.

The YUV colour format was selected for use in the tracking algorithms. This format gives the captured pixel data in luminance, represented by Y, and chrominance values, represented by the U and V. This colour format provides two advantages when used for tracking applications. Johnston, Bailey & Gribbon (2005) discusses the ability to remove the reliance on the brightness by only using the U and V components of the pixel values. The result is an algorithm that demonstrates a higher level of robustness when used in changing light conditions or where shadows may pass over an object being tracked.

The second advantage of using only the chrominance values is that each calculation of the Euclidean distance becomes a two-dimensional vector operation instead of a three-dimensional calculation. As a result, the computational load of the algorithm is reduced.

However, the use of the YUV format posed separate issues that would not have been present had RGB format been selected. OpenCV is unable to correctly output the image when the image data is in YUV format. The luminance values were treated as the red values, and the two chrominance values were treated as the green and blue values, respectively. OpenCV contains commands that translate the YUV values into RGB for display. The numerical method for converting the YUV values to RGB is given as (Jones 2016):

$$\begin{aligned} R &= 1.164Y + 0.000U + 1.596V \\ G &= 1.164Y - 0.3192U - 0.813V \\ B &= 1.164Y + 2.017U + 0.000V \end{aligned} \tag{4.1}$$

Performing this calculation for each pixel or using the inbuilt function to do so created another point of delay in the tracking algorithm and was therefore not viable.

The solution to this was found by implementing the splitter function within the picamera library. This command enables multiple camera configurations to be established and called when required. Two image formats were used for the tracking algorithms. The first configuration was used to capture the images for display. The images were taken in the RGB format with a resolution of 368 by 640 pixels. The second configuration captured images in the YUV format and was only 94 by 160 pixels in size. This time the captured frames were used for tracking. The difference in image size was an attempt to reduce the computational load of the tracking while maintaining higher image quality for displaying on the mirror.

Before a K-means algorithm can be implemented, the number of clusters must be determined. A considerable amount of research has been completed into this topic. Kodinariya & Makwana (2013) performed an investigation into different methods of determining the optimum number of clusters to be used when implementing the K-means algorithm. Many of the techniques covered in the paper required repeating the clustering process using different numbers of clusters. Each time the algorithm was run, key factors were recorded and compared to find the most appropriate number of clusters. Using a method that involves multiple iterations of the algorithm is not possible in this case as there would be a further increase in the delay in the tracking of the shaving implement.

One suggested method of determining the number of clusters to be used was the implementation of a "rule of thumb". The formula for this is given as:

$$k \approx \sqrt{n/2} \tag{4.2}$$

where $n$ is the number of data points or pixels. With the size of the low-resolution image used for tracking was 94 by 160 pixels, and this gives a result of approximately 87 clusters. This number is far too high for use in image tracking. From the observations made during testing, it could be seen that a compromise must be reached when selecting the number of clusters to be used. A higher number of clusters will more accurately group items separately without accidentally adding other similarly coloured objects to the cluster. However, increasing the number of clusters significantly increases the processing time for the algorithm. It was observed that using cluster numbers below 20 caused the clustering to be often incorrect, and the object was lost. Subsequently, the number of clusters used for testing of the tracking was set at 25.

As mentioned in Chapter 2, in the original K-means algorithm, the initial cluster centres were randomly assigned. To create these clusters, a function was written to generate random values in an array. These values ranged from 0 to 255 as the values for the pixels are each stored in an eight-bit format. When this function was implemented, it was observed that as many as 75% of the clusters failed to have any pixels assigned to them. While this did sometimes improve after subsequent iterations of the algorithm, there appeared to be a disproportionate number of pixels in a small number of clusters. Further investigation revealed that the U and V values within the image tended to closely follow a Gaussian distribution. To achieve a more even distribution of the pixels between the clusters, the random cluster centres were also made to have a Gaussian distribution. This was done by using a random number generation function that returned values with a predetermined mean and standard deviation. After experimenting with different values, it was decided that a mean of 128 and a standard deviation of 10 was most effective. The code detected any values that fell outside the 0 to 255 range and replaced them with another random number, this time with a standard deviation of 30.

Before the clustering began, the program ran for five seconds displaying the images captured by the camera. During this time, only the images for display were taken, and no tracking data was collected. A set of green crosshairs were displayed in the middle of the image by changing the red, green and blue values of the desired pixels in the image to be displayed to 0, 255, 0, respectively. The images were then displayed on the screen. On completion of the five seconds, the tracking would commence, and the second camera configuration for tracking would also be used.

The basic K-means algorithm was structured in a loop. On each iteration of the loop, the cluster function was called. This function was provided with a three-dimensional array containing the YUV values for the current image. The cluster function began by setting the required variables and calling another function that returned the randomised initial cluster centres. To assign the pixels to the closest cluster, a temporary variable was created with the value of positive infinity stored in it. The Euclidean distance between the first pixel being examined and each of the cluster centres was calculated in turn. After each of these calculations, if the calculated distance was less than what was stored in the variable, it would replace it. Every time a lower Euclidean distance between the pixel and cluster was found, the cluster number was also stored in a temporary variable. Once the calculation of the distances between the pixel and clusters was complete, the closest cluster number was stored in a two-dimensional array in the position corresponding with the pixel location in the image. Nested loops were implemented to cycle through the pixels enabling the closest cluster to be determined for each pixel.

At the same time as the cluster allocations were stored, an array to be used for calculation of the new cluster centres was populated. In the row corresponding to the cluster number that the pixel was assigned to, the pixels U and V values were summed with the second and third values, and the first value was incremented by one to provide a count of pixels assigned to the cluster. Once all pixels had been assigned to clusters, the new cluster centres were calculated by dividing the summed U and V values by the tally. These results were regarded as the new cluster centres. By comparing these new cluster centres with the previous cluster centres, it could be determined if the algorithm had reached equilibrium. If the two cluster sets were not identical, the process would restart, this time using the cluster centres that had been calculated in the previous iteration. Once the clustering had been completed, the function would return the cluster values and the array containing the cluster assignment information for the pixels.

On the first iteration of the cluster function, the cluster that contained the pixel at the centre of the crosshairs was considered the target cluster. In each iteration after this, the cluster returned that had the lowest Euclidean distance to the original target cluster was labelled the new target cluster.

The position of the centre of the target cluster was found by calculating the mean location of all pixels assigned to that cluster. The crosshairs were relocated to the cluster centre in the output image to show the position. Displaying the crosshairs will not be used when the tracking algorithm is implemented into the mirror program and is only used for indicating the effectiveness of the algorithm.

When programming the algorithm, there were often multiple methods that could be used for executing a calculation or a command. One example of this is the calculation of the Euclidean distance. As this calculation is effectively finding the magnitude of a vector, the standard formula used is:

$$|\bar{A}| = \sqrt{{a_1}^2 + {a_2}^2 ... + {a_n}^2} \tag{4.3}$$

where $a_1, a_2 ... a_n \in \bar{A}$.

This calculation could be achieved using the math.sqrt and pow (for power) functions built into the math library in Python. The same result could be found using the linalg.norm function found in the Numpy library. The latter of the two options mentioned was selected for the programs as testing revealed a slight improvement in computational time between the two.

Numpy is a library of functions written to address the limitations encountered when using arrays in Python. The Python language does not support arrays. Instead, there is a data type called lists. While lists can provide similar functionality to an array, they are only intended for one-dimensional implementation. Multidimensional lists can be created when an item in the list is itself a list. However, the indexing of a single item within this arrangement is very onerous. The use of the Numpy library simplifies the use of arrays as is required for image processing. It also provides multiple functions to interact with arrays, some of which were used in the algorithms. Two of these functions were the linalg.norm and the array_equal functions. The use of the former is described above. The array_equal function tests two arrays, element by element, to determine if they are the same. This was used to assess if there had been any change in the cluster centres after each iteration of the clustering process.

The full Python script for the K-means test program is included in Appendix B.

## 4.2 K-means with Modifications

In the literature review, two methods of improving the speed of the algorithm by reducing the number of calculations required to complete the clustering were discussed. The first was developed by Fahim et al. (2006). The reduction in computational load was achieved by testing the Euclidean distance between each pixel and the cluster that they were assigned to in the previous iteration. If the distance had not increased, the pixel remained assigned to the same cluster.

The second optimisation method covered in the literature was the work of Heisele et al. (1997). Their work demonstrated that tracking of an object in a video stream could be made more efficient by reusing the cluster centres found by the algorithm in the next frame. The theory behind this method was that only an incremental change occurred from one image in the video to the next, and therefore, the clusters would be similar if not the same from one frame to the next.

By combining these two methods, it was believed that an even more significant reduction in the computational load could be achieved. The cluster centres form one frame would be used as the initial cluster values, sometimes called seeds, in the proceeding frame. The incremental changes in the image from one frame to the next would also mean that areas of the image that had little or no change could often be reassigned to the same clusters as before. This could be tested by assessing if the Euclidean distance between the pixels and the cluster centre to which they were previously assigned has increased. If it has not, it can be assumed that the pixel remains assigned to the same cluster. This method is the same as is used for subsequent iterations of the algorithm on an image.

The optimised K-means algorithm was programmed by modifying the basic K-means implementation that was described above. All functions remained the same except for the cluster function. One significant difference between this function in the two algorithms was the variables that were passed to and from the function. In the basic K-means algorithm, this function only requires the image data to be provided. However, for the optimised version, the additional variables needed by the function are:

- an array containing the previous cluster centres
- an array the same dimension as the pixel count of the image containing the previous cluster allocations
- the distances from each pixel to the assigned cluster.

The function was written to only call for the generation of random cluster centres on the initial iteration. Each time it is run after this, the previous cluster centres were used, and if the Euclidean distance had not increased, the distances to the other clusters would not be assessed.

The changes described above were made possible by returning additional variables from the function. In the basic K-means algorithm, only the list of cluster centres and an array showing which of these the pixels were assigned were returned. In addition to this, the modified K-means algorithm required the Euclidean distances also to be returned.

The complete listing of the program is included in Appendix C.

## 4.3 Cross-Correlation using an Integral Image

The implementation of the cross-correlation algorithm required more changes than was needed for the second K-means algorithm. The first change made was to replace the code that produced the crosshairs with a box indicating the outline of the template location.

In the same manner as the K-means algorithm operated, the cross-correlation program also waited for five seconds before starting the tracking. When the time was completed, the group of pixels in the centre of the tracking image were taken as the template.

The integral image of the template did not need to be calculated. Instead, the sums of the pixel values were calculated for the region of the image. When using the YUV format, only the U and V values were summed. Initially, the calculations were completed using nested loops that cycled through each of the pixels in the image. Another Numpy function, numpy.sum, was discovered that was able to find the sum of the items stored in an array. The two methods were tested and the numpy.sum function was selected as it returned the result in a shorter time.

The function that was written to perform the cross-correlation receives the image data and the sum of the template values previously calculated. The integral image is calculated first. There were multiple methods tested for calculating this. One method was to run the numpy.sum function for each point in the image. The function would be given a rectangular area or the image from the top-left pixel to the pixel being examined. The code required for this method was concise and required less programming than any other option. However, this method was not used. Other options that were tested proved to be more efficient. It was believed that the slower computation was due to repeated calculations that were not required in other methods.

Another method that was tested used the previously calculated integral image values to reduce the number of values to be added. The value of the integral image at the top left of the image is equal to the pixel value or:

$$II\left(1,1\right) = i\left(1,1\right) \tag{4.4}$$

where $II\left(x,y\right)$ is the integral image at the position $\left(x,y\right)$ and $i\left(x,y\right)$ is the pixel value at the position $(x, y)$.

The first row and first column were calculated using the formulas 4.5 and 4.6, respectively:

$$II\,(x,1) = II\,(x-1,1) + a\,(x,1) \tag{4.5}$$

$$II\,(1,y) = II\,(1,y-1) + a\,(1,y) \tag{4.6}$$

For a $N$ by $M$ pixel image, $x = 2, 3, 4...N$ and $y = 2, 3, 4...M$.

Once these values have been found, the remaining integral image values are found using the formula below:

$$II\,(x,y) = II\,(x-1,y) + II\,(x,y-1) - II\,(x-1,y-1) + i\,(x,y) \tag{4.7}$$

As in equations 4.5 and 4.6, $x = 2, 3, 4...N$ and $y = 2, 3, 4...M$.

This equation means that a maximum of four numbers needs to be added or subtracted for each integral image value. When using the numpy.sum method, the number of values needing to be used for each calculation is equivalent to the number of pixels above and to the left of the one being examined.

Once the integral image had been formed, Equation 2.12 was used to assess the correlation between the template and each location in the image.

The initial cross-correlation program was written using images for tracking in the YUV colour format. As explained earlier, the motivation for this was to reduce the computational load and provide better immunity to variations in light. Unfortunately, the use of this colour format was ineffective. The cross-correlation became highly inaccurate when using this format with the tracking repeatedly giving incorrect results. As a result, the code was changed to use all three pixel colour values and the images were captured in the RGB format. Testing proved that the accuracy of the algorithm outweighed the additional computational expense.

Appendix D contains the code produced for this algorithm.

# Chapter 5

# Hardware Development

Due to time constraints and limitations encountered as a result of COVID 19, a working prototype of the smart mirror has not been constructed. Some thought and research have been completed into the design, and many of the components have been acquired. There are numerous websites dedicated to the construction of a smart mirror that can be achieved with only minimal tools required. As previously stated, the basic makeup of a smart mirror is an LCD monitor behind one way or unidirectional glass. This is usually held together with a custom made frame that can support all components when mounted on a wall.

## 5.1 Required Components

The monitor chosen for the design was a Lenovo 24 inch backlit LCD monitor. This was chosen as it was available at a low cost and would be suitable for the task required. The selected monitor is not of great importance as this will not create any noticeable difference in the final result. The only significant difference this selection made to the design was the way that the frame mounted to the monitor.

A computer to run the smart mirror software on was also required. The computer was required to be small enough to be mounted within the frame holding the mirror together. There are multiple single-board computers currently available on the market. After reading reviews and looking at previous smart mirror builds, it was decided to purchase a

Raspberry Pi 4 Model B board. The 8Gb version was selected as this computer was the fastest available at the time the project was undertaken. There were many reasons for choosing the Raspberry Pi over other available systems.

The Raspberry Pi is very popular for both hobby and commercial applications. Due to this, there is an extensive range of online resources to assist in using the system. Multiple forums can be found with discussions of different applications of both hardware and software compatible with the system. This popularity has also lead to the vast majority of smart mirror programs available on the internet being intended for use on a Raspberry Pi. Discussions regarding this can be found in Chapter 3 and formed a considerable part of the decision to use a Raspberry Pi.

Another advantage of the Raspberry Pi over many other similar platforms is the wide range of accessories and auxiliary components that are compatible. Both the Raspberry Pi Foundation and third party manufacturers offer accessories that enhance and expand the capabilities of the Raspberry Pi. One such accessory used in this project was the Raspberry Pi Camera Board V2. There are multiple camera boards available for the Raspberry Pi. The V2 that was chosen is approximately the middle of the range and is capable of capturing 1080p30 video.

The Raspberry Pi does lack the ability to drive speakers directly from the board. There is an inbuilt 3.5mm audio jack located on the device. However, the power output is insufficient to drive anything other than a set of headphones. There are multiple options for connection of speakers to the device. One is to feed from the audio jack to an amplifier which in turn provides the required power to the speakers. PiAustralia, the official Raspberry Pi website in Australia, has modules available to perform this function (Raspberry Pi Foundation n.d.*b*). These modules also receive their power supply from the IO pins on the Raspberry Pi. While this reduces the hardware requirements, there are disadvantages to this configuration. The speakers will be limited in the volume they can achieve as the power supply for the Raspberry Pi is limited to 15W. Using the Raspberry Pi for power will also increase the heating of the unit and the power supply due to the increased current drawn. An externally powered amplifier feeding more powerful speakers would be desirable to enable the use of audible response from the mirror and streaming of videos and music.

Another option available for audio connection within the smart mirror would be to use a Bluetooth connection between the Raspberry Pi and the speaker. The Raspberry Pi 4 has Bluetooth 5.0 capability built into the unit, giving it the ability to send audio to two different devices simultaneously.

The final option for the audio output would be to select a monitor with inbuilt speakers. The connection between the Raspberry Pi and the monitor is through one of two onboard microHDMI ports. These ports can send both visual and audio information simultaneously to a monitor.

While the Raspberry Pi is a single-board computer system, it does not possess a hard drive. There is a microSD card slot for inserting a card to be used as a hard drive. The most common method of installing an operating system is to insert a microSD card containing the NOOBS (New Out Of the Box Software) operating system installation software. These can be purchased from many different suppliers, or any microSD card can be loaded with NOOBS by following the guide found on the Raspberry Pi Foundation website (Raspberry Pi Foundation n.d.*b*).

Interaction with the smart mirror was intended to be achieved using voice control. This requires a microphone to detect the user's speech. The Raspberry Pi does not have a microphone input port. One of the four available USB ports would be used for this purpose. Further work would need to be completed to determine the type of microphone that would be most appropriate for this application.

The final component to be considered in the design of the smart mirror was the power supplies. When all other components are connected and running, there will be three separate power outlets required. These are for the Raspberry Pi, the monitor and the selected speaker system. To reduce the number of cables needed for the unit, a power board would be installed on the inside of the frame. Four outlet power boards are inexpensive and readily available.

## 5.2 Arrangement of Components

The selection of the monitor will mainly determine the layout of the mirror. The frame holding the mirror together would be required to attach to the monitor securely as well as supporting the other components.

The position of the camera has an effect on the image captured. As the image is intended for assisting in shaving, it would be beneficial for the camera to be below the display rather than above. This will enable the user to gain a view under his chin while minimising the need to lift his head. If the camera were located above the screen, it would be more difficult to achieve this view.

Another consideration when planning the positioning of the components is the length of the connections between parts. The camera board is supplied with a ribbon cable of approximately 120mm in length. This limits the distance that the camera can be located from the Raspberry Pi. The Picamera library contains functions for rotating and flipping the images as desired without increasing the computational load.

Care must be taken during design and construction to prevent light coming through the rear of the mirror and hitting the back of the unidirectional mirror. Any light that does protrude through will stop the mirror from reflecting, and the outline of the components behind the glass will be visible. Some examples of smart mirrors that were encountered during research for this project used a fully enclosed back on the frame for this purpose. This is not the most desirable method as the airflow will be restricted and may cause heating issues for the equipment inside the frame.

The monitor has a series of four threaded holes in the rear. These holes are intended for use with a wall bracket or on a monitor stand and would form a secure method of attaching the monitor to the frame if a plate was placed directly across the rear of the frame.

A basic diagram has been included in Figure 5.1 of the front elevation of the proposed smart mirror design. This illustration shows the camera mounted below the monitor. When observing the mirror, this would not be visible due to the unidirectional glass.

The rear elevation shown in Figure 5.2 demonstrates a proposed mounting location for the main components. While they are not shown in the diagram, speakers would be housed in each side of the frame, and a microphone would be mounted in the bottom. The depth of the structure would partially be dependent on the size of the speakers selected. The timber used would need to be sufficiently wide to allow for a hole to be cut, allowing the speakers to be recessed into the wood.

The metal plate used in the frame will provide much of the structural support for the

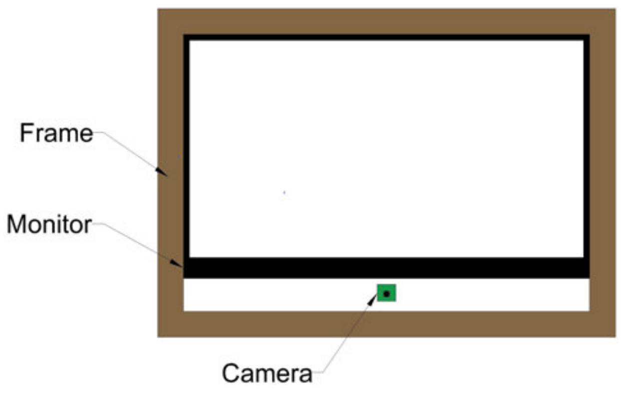


Figure 5.1: Basic representation of the front view of the assembled smart mirror.

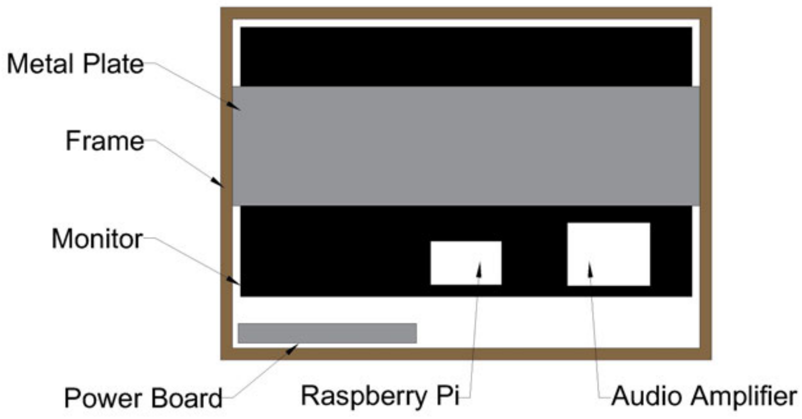


Figure 5.2: Rear elevation of proposed smart mirror design.

mirror. It will provide sufficient strength to support the monitor as well as the frame. This frame will, in turn, support all the other components. Holes will be able to be cut into the plate to allow for hanging of the mirror on a wall. The proposed holes should be made in a keyhole shape, as shown in Figure 5.3 to allow a screw head to fit through the larger bottom of the hole and slide into the slot to prevent the mirror falling. A series of these holes could be created to allow for different mounting situations and various support spacings.

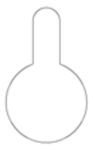


Figure 5.3: Keyhole shape for mounting holes.

As previously mentioned, a power board is to be mounted in the bottom of the frame, and all other components will receive their power from this. The direction and orientation of the powerboard will need to be decided upon to fit the required power plugs and adaptors for the selected equipment.

# Chapter 6

# Results

This chapter will discuss some of the final results found while completing the project. Many of the discoveries made during the implementation and testing of the tracking algorithms are included in Chapter 4. The reason for this was that the information found had a strong influence on the ongoing development of the tracking programs.

The two algorithms that were implemented and tested were the K-means and the cross-correlation algorithms. The K-means was tested in the original form as well as with modifications applied to improve the computational efficiency. The techniques tested to improve the speed were:

- reusing cluster centre values from one frame to the next and
- testing if the Euclidean distance to the previously assigned cluster had increased before testing the distance to other cluster centres.

The traditional cross-correlation technique requires an extremely high number of calculations. For a $N_x$ by $N_y$ image using a $M_x$ by $M_y$ template, the calculation of a single correlation value in one position the uses Equation 2.6. For the more robust normalised cross-correlation, the formula is displayed in Equation 2.7. This equation results in a considerable increase in the complexity and computational load of each calculation. For this reason, the cross-correlation algorithm was only implemented using the integral image, as previously discussed.

Both variations of the K-means algorithms produced and the cross-correlation tracking algorithms were successful in tracking an object during testing. Various pens were selected as test specimens for the tracking. This provided a wide range of colours and patterns within the targets needing to be required. Successful tracking of different targets was needed as the smart mirror is intended to acquire any shaving implement the user chooses.

The efficiency of the K-means algorithm was improved by using the YUV colour format rather than the RGB format. Conversely, the cross-correlation algorithm was unable to accurately track the object using the YUV format but demonstrated a much higher accuracy using RGB formatted images.

The greatest challenge when implementing the algorithms was reducing the time taken for the object to be acquired within the following frames. The times taken for performing the tracking in a single frame are demonstrated in Table 6.1. These times are given as average values. For all but the modified K-means algorithm, the observed tracking times were consistent from one frame to the next. The variation in the time taken for the modified K-means can be attributed to the degree of change from one frame to the next. When there was very little change in the image, the tracking was completed in a much faster time than when large differences were observed.

Table 6.1: Tracking Algorithm Testing Results.

| | **Basic K-means** | **Modified K-means** | **Cross-correlation** |
|---|---|---|---|
| Initial Acquisition | 22s | 24s | 0.74s |
| Proceeding Frames | 22s | 9.2s - 24.5s | 0.74s |

It was also noted that the modified K-means algorithm required a longer time to perform the initial acquisition process. While this was not expected, it can be explained. The modifications to the algorithm do successfully reduce the number of computations required; however, there is an increase in the number of variables used and stored. In the basic K-means algorithm, only the cluster centres and a record of which cluster each pixel was assigned to was stored. With the modifications, the distances to the closest cluster for each pixel was also stored and returned.

Checking the distance to the assigned cluster before checking the distance to other clusters also slowed the algorithm down. In the code used, with 25 clusters, this would mean that the pixel distance would need to be calculated for 26 clusters instead of 25. However, once the initial clustering was complete, these additional processing tasks were outweighed by the reduction in computational load as previously discussed.

One disadvantage of the cross-correlation algorithm discussed in the literature was a weakness when tracking non-rigid objects. During the testing of this algorithm, the pen being tracked was rotated in all directions at different times. The algorithm did not have an issue with continuing the tracking even when this occurred. It is believed that there are two contributing factors influencing this unexpected outcome. First, the use of the integral image means that instead of seeking the exact template, on approximation is used. This approximation only tests for the sum of the pixel values instead of the exact correlation of each pixel. While this can cause a reduction in the accuracy when tracking an object, it has the benefit of removing the reliance on the orientation of the template.

The second influencing factor that may have increased the resilience to non-rigidity is the use of a square template. Had a rectangular template been used instead of a square one, the rotation of the object would have hade a greater influence on the pixels selected in the template. This was not deliberately implemented in the algorithm, but on reflection, may have been beneficial for the algorithm.

The cross-correlation algorithm also proved resilient to occlusions and displayed the ability to reacquire the object once it had left the view of the camera and returned. As the algorithm was set to continue seeking the same template, once the camera was able to gain a clear view of the object again, the tracking would continue. This would prove beneficial if a traditional razor was being used that required rinsing the hair from the blade. The K-means algorithm did not display the same ability as once the clustering was performed without the object in the frame, the cluster centres had moven from the desired values and therefore acquired the wrong pixels.

The testing of the algorithms demonstrated that significant savings could be made in the time required by selecting and implementing the correct algorithms. Using additional optimisation techniques can also produce savings in the computational load. From the algorithms chosen and implemented, the cross-correlation algorithm was by far the more efficient algorithm.

# Chapter 7

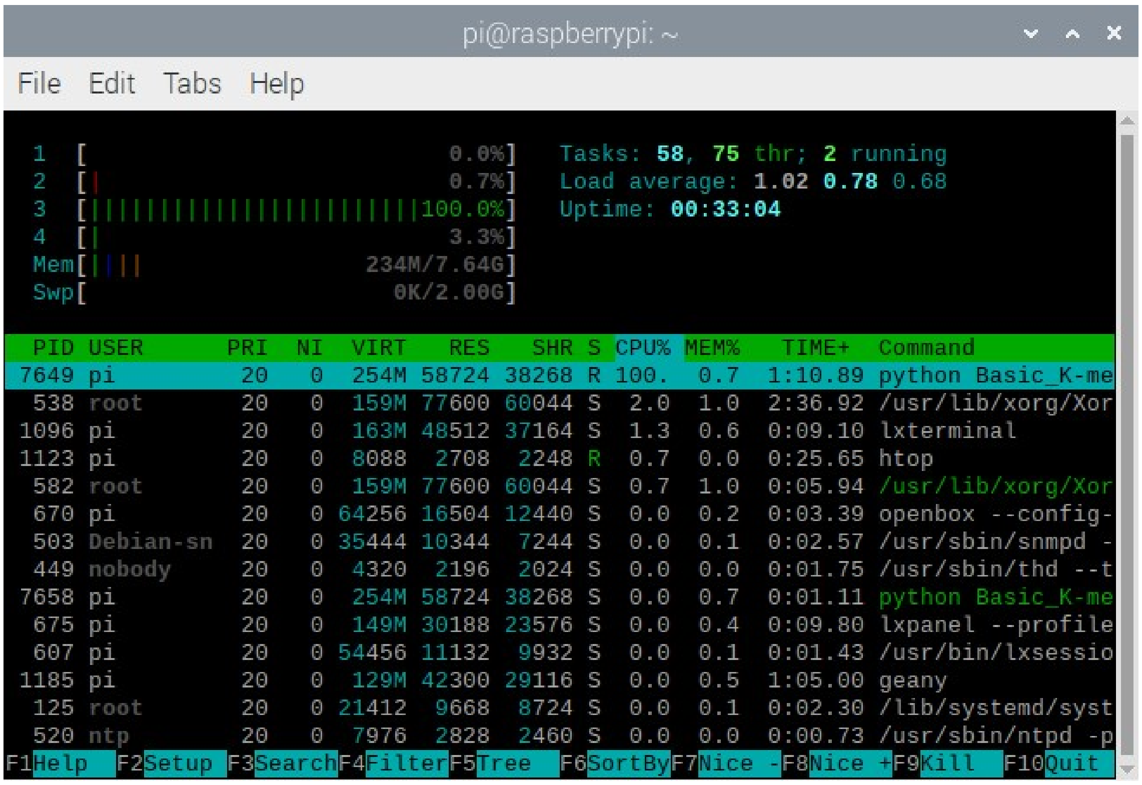
# Conclusions and Further Work

## 7.1 Conclusion

There were two main research objectives to be achieved within this project. The first was the development of a tracking algorithm to be used by the mirror. As can be seen from the previous chapters, both of the K-means and the cross-correlation algorithms were successful in the tracking of a representative sample. However, the time taken for the location of the object within the image is beyond what is considered acceptable in the current form. The results of the cross-correlation algorithm proved promising with an acquisition time that was much lower than that of the K-means variations. In the future work proposed below, suggested actions are provided to achieve an acceptable outcome concerning the tracking.

Construction of a working smart mirror was not achieved during this project. In Chapter 5, required components and an outline of the design were discussed. Many of the factors that need to be considered during the construction were also addressed. Not having a completed mirror did not hinder the ability to continue with the development of the tracking algorithms as the essential components had been acquired and were able to be used and tested arranged on a desk.

One additional finding that came as a result of the testing result is the limitations that exist within the Python language. While there are a vast variety of functions available, making the language versatile, not everything desired was achievable. One reason for the unexpected slow tracking speed was found to be the way that the Python uses the processor cores. When running HTOP, a performance analyses tool, on the Raspberry Pi, it was noted that only one of the four cores available were being used for the calculations. Python does support multiprocessing; however, when trying to implement this, it was found that it is not supported within a loop for repeated calls. The process would work on the first iteration of the loop but would stop and return an error on the second. The error stated that a process could only be called once, thereby meaning it could not be implemented within a loop. As the camera acquisition is performed within a loop, due to the indefinite duration of the capture period, this made the use of multiprocessing unachievable. Figure 7.1 demonstrates a screen capture from the Raspberry Pi displaying the processor and memory used during the running of the basic K-means tracking algorithm.



```
  1  [                          0.0%]   Tasks: 58, 75 thr; 2 running
  2  [|                         0.7%]   Load average: 1.02 0.78 0.68
  3  [|||||||||||||||||||||||100.0%]   Uptime: 00:33:04
  4  [|                         3.3%]
  Mem[||||                234M/7.64G]
  Swp[                      0K/2.00G]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 7649 pi         20   0  254M 58724 38268 R 100.  0.7  1:10.89 python Basic_K-me
  538 root       20   0  159M 77600 60044 S  2.0  1.0  2:36.92 /usr/lib/xorg/Xor
 1096 pi         20   0  163M 48512 37164 S  1.3  0.6  0:09.10 lxterminal
 1123 pi         20   0  8088  2708  2248 R  0.7  0.0  0:25.65 htop
  582 root       20   0  159M 77600 60044 S  0.7  1.0  0:05.94 /usr/lib/xorg/Xor
  670 pi         20   0 64256 16504 12440 S  0.0  0.2  0:03.39 openbox --config-
  503 Debian-sn  20   0 35444 10344  7244 S  0.0  0.1  0:02.57 /usr/sbin/snmpd -
  449 nobody     20   0  4320  2196  2024 S  0.0  0.0  0:01.75 /usr/sbin/thd --t
 7658 pi         20   0  254M 58724 38268 S  0.0  0.7  0:01.11 python Basic_K-me
  675 pi         20   0  149M 30188 23576 S  0.0  0.4  0:09.80 lxpanel --profile
  607 pi         20   0 54456 11132  9932 S  0.0  0.1  0:01.43 /usr/bin/lxsessio
 1185 pi         20   0  129M 42300 29116 S  0.0  0.5  1:05.00 geany
  125 root       20   0 21412  9668  8724 S  0.0  0.1  0:02.30 /lib/systemd/syst
  520 ntp        20   0  7976  2828  2460 S  0.0  0.0  0:00.73 /usr/sbin/ntpd -p
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

Figure 7.1: Screen capture of FTOP system analyses tool.

## 7.2 Future Work

From the previous discussion, it can be seen that a working mirror was not able to be produced during this project. The steps required to achieve this will be discussed as suggested work to be completed in the future.

Further development of the tracking algorithms is required for the successful completion of the smart mirror. The K-means algorithm was proven to be more efficient when using the modifications. However, the time taken for each iteration was far too long. Two methods of reducing the computational load that have not yet been tested were formulated. In the first method, the calculation of the Euclidean distance to the assigned cluster would not be calculated. Instead, a record of the cluster centres that had moved in the previous iteration would be kept. If a cluster centre had not moved, the pixels assigned to that cluster would not be recalculated. This has the advantage of requiring less Euclidean distance calculations to be made. If the cluster centre was found to have moved, the process would continue for each pixel assigned to that cluster in the same way as it is currently executed.

The second method devised takes into account that the magnitude of distance will always be greater than or equal to the $x$ or $y$ components of the same distance. Because of this, the Euclidean distances will be calculated only for the cluster centres that exhibit a $x$ and $y$ component less than or equal to the previous Euclidean distance. This will result in many of the calculations required being a less onerous addition instead of the usual Euclidian calculations that require both squaring of components and the square root function.

While these two suggested methods have the potential to reduce the load created by the algorithm, it is not expected to achieve a sufficient reduction in time to make the algorithms practical for real-time tracking in this situation. Therefore, the cross-correlation algorithm appears to be the most viable of the options explored.

Another language could be used to achieve an algorithm that is capable of running with sufficient speed on a Raspberry Pi. Briggs (2006) performed research to analyse the efficiency of performing mathematical calculations in Python, C and C++. The results demonstrated a vast difference in the speed at which the calculations were completed. C++ was found to provide a significantly more efficient method of completing calculations. Additional computational savings could be made by implementing the code in C++. C++ offers more comprehensive and robust multiprocessing libraries than Python.This would enable the use of all four of the Raspberry Pi cores. While this has not been tested, there is the potential for a reduction in computational time by more than 75%.

Access to the camera in the C++ language would be able to be achieved using a library such as raspicam‘(Aplications of Artificial Vision n.d.). This library offers full control of the camera for both still and video functions.

There are other methods that may be considered when trying to implement the tracking of the shaving device. The tracking algorithm could be run as a separate program or process. By doing this, the main process would be able to continue providing the image to the screen while the secondary process performs the tracking. As the focal point of the image will not be changed with each frame, the program can continue supplying a video stream trained on a stationary point until the tracking process gives the command to change the focal point. Given that this focal point will not need to relocate frequently, the tracking algorithm can be set to operate at a lower frequency than that of the video display. The standard frame rates for a video is either 30 or 60 frames per second. This method would allow the image to be displayed on the screen at 30 frames per second, while the tracking algorithm runs only three times per second. This would require the tracking to be complete in 333ms, which may be achievable given what has been previously discussed.

# References

Akshaya, R., Raj, N. N. & Gowri, S. (2018), Smart mirror- digital magazine for university implemented using raspberry pi, *in* '2018 International Conference on Emerging Trends and Innovations In Engineering And Technological Research (ICETIETR)', pp. 1–4.

Ali, A., Jalil, A., Niu, J., Zhao, X., Rathore, S., Ahmed, J. & Iftikhar, M. A. (2016), 'Visual object tracking—classical and contemporary approaches', *Frontiers of Computer Science* **10**(1), 167–188.

Aplications of Artificial Vision (n.d.), 'Raspicam', `http://www.uco.es/investiga/grupos/ava/node/40`. [Online; accessed October-2020].

Bianco, S., Celona, L. & Napoletano, P. (2018), Visual-based sentiment logging in magic smart mirrors, *in* '2018 IEEE 8th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)', pp. 1–4.

Bonnain, S. (2018), 'Interactive mirror with an app store', `https://www.kickstarter.com/projects/743717037/eve-smart-mirror-interactive-smart-mirror-with-an/description`. [Online; accessed April-2020].

Briggs, K. (2006), 'Implementing exact real arithmetic in python, c++ and c', *Theoretical Computer Science* **351**(1), 74 – 81. Real Numbers and Computers.
**URL:** *http://www.sciencedirect.com/science/article/pii/S0304397505006080*

Channu, M. H., Bheemashappa, N. S. & Sudharshan, K. (2019), 'Smart mirror table', *International Journal of Scientific Research and Review* **7**(3), 2533–2537.

Cheng, Y. (1995), 'Mean shift, mode seeking, and clustering', *IEEE transactions on pattern analysis and machine intelligence* **17**(8), 790–799.

Colantonio, S., Coppini, G., Germanese, D., Giorgi, D., Magrini, M., Marraccini, P., Martinelli, M., Morales, M. A., Pascali, M. A., Raccichini, G. et al. (2015), 'A smart mirror to promote a healthy lifestyle', *Biosystems Engineering* **138**, 33–43.

Collins, R. T. (2003), Mean-shift blob tracking through scale space, *in* '2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.', Vol. 2, IEEE, pp. II–234.

Comaniciu, D., Ramesh, V. & Meer, P. (2003), 'Kernel-based object tracking', *IEEE Transactions on pattern analysis and machine intelligence* **25**(5), 564–577.

Darrell, T., Gordon, G., Woodfill, J. & Harville, M. (1998), A virtual mirror interface using real-time robust face tracking, *in* 'Proceedings Third IEEE International Conference on Automatic Face and Gesture Recognition', pp. 616–621.

Davies, E. R. (2012), *Computer and machine vision: theory, algorithms, practicalities*, Elsevier Science & Technology.

de Tena Rojas, A. (2017), 'Mmm- selfie', `https://github.com/Txukie/MMM-Selfie`. [Online; accessed July-2020].

Ding, J.-R., Huang, C.-L., Lin, J.-K., Yang, J.-F. & Wu, C.-H. (2007), Magic mirror, *in* 'Ninth IEEE International Symposium on Multimedia (ISM 2007)', IEEE, pp. 176–185.

Elgammal, A., Duraiswami, R. & Davis, L. S. (2003), 'Efficient kernel density estimation using the fast gauss transform with applications to color modeling and tracking', *IEEE transactions on pattern analysis and machine intelligence* **25**(11), 1499–1504.

Evens, P. (2020), 'Build a Magic Mirror', `https://magpi.raspberrypi.org/articles/build-a-magic-mirror`. [Online; accessed May-2020].

Fahim, A., Salem, A., Torkey, F. A. & Ramadan, M. (2006), 'An efficient enhanced k-means clustering algorithm', *Journal of Zhejiang University-Science A* **7**(10), 1626–1633.

Fisher, R. B., Breckon, T. P., Dawson-Howe, K., Fitzgibbon, A., Robertson, C., Trucco, E. & Williams, C. K. I. (2014), *Dictionary of Computer Vision and Image Processing*, John Wiley & Sons, Incorporated, Somerset.

Fukunaga, K. & Hostetler, L. (1975), 'The estimation of the gradient of a density function, with applications in pattern recognition', *IEEE Transactions on information theory* **21**(1), 32–40.

*GitHub search results- smart mirror* (2020), `https://github.com/search?q=smart+mirror`. [Online; accessed August-2020].

Gold, D., Sollinger, D. & Indratmo (2016), Smartreflect: A modular smart mirror application platform, *in* '2016 IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)', pp. 1–7.

Greengard, L. & Strain, J. (1991), 'The fast gauss transform', *SIAM Journal on Scientific and Statistical Computing* **12**(1), 79–94.

Heisele, B., Kressel, U. & Ritter, W. (1997), Tracking non-rigid, moving objects based on color cluster flow, *in* 'Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition', IEEE, pp. 257–260.

Hii, A., Hann, C., Chase, J. & Van Houten, E. (2006), 'Fast normalized cross correlation for motion tracking using basis functions', *Computer methods and programs in biomedicine* **82**(2), 144–156.

Hossain, M. A., Atrey, P. & Saddik, A. E. (2007), 'Smart mirror for ambient home environment', *IET Conference Proceedings* pp. 589–596(7).

Iwabuchi, E., Nakagawa, M. & Siio, I. (2009), Smart makeup mirror: Computer-augmented mirror to aid makeup application, *in* J. A. Jacko, ed., 'Human-Computer Interaction. Interacting in Various Application Domains', Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 495–503.

Jin, K., Deng, X., Huang, Z. & Chen, S. (2018), Design of the smart mirror based on raspberry pi, *in* '2018 2nd IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)', IEEE, pp. 1919–1923.

Johnston, C., Bailey, D. & Gribbon, K. (2005), Optimisation of a colour segmentation and tracking algorithm for real-time fpga implementation, *in* 'Proceedings of Image and Vision Computing New Zealand', pp. 422–427.

Jones, D. (2016), 'Picamera documentation', `https://www.picamera.readthedocs.io/release-1.12/index.html`. [Online; accessed August-2020].

Kim, Y., Park, H. & Paik, J. (2018), Deep tracking using convolutional features and adaptive frame update, *in* '2018 IEEE 8th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)', pp. 1–3.

Kodinariya, T. M. & Makwana, P. R. (2013), 'Review on determining number of cluster in k-means clustering', *International Journal* **1**(6), 90–95.

Leichter, I., Lindenbaum, M. & Rivlin, E. (2010), 'Mean shift tracking with multiple reference color histograms', *Computer Vision and Image Understanding* **114**(3), 400–408.

Meine, R. K. (2000), 'System and method for displaying information on a mirror'. US Patent 6,560,027.

Miotto, R., Danieletto, M., Scelza, J. R., Kidd, B. A. & Dudley, J. T. (2018), 'Reflecting health: smart mirrors for personalized medicine', *NPJ digital medicine* **1**(1), 1–7.

Na, S., Xumin, L. & Yong, G. (2010), Research on k-means clustering algorithm: An improved k-means clustering algorithm, *in* '2010 Third International Symposium on Intelligent Information Technology and Security Informatics', pp. 63–67.

Oliver, A., Munoz, X., Batlle, J., Pacheco, L. & Freixenet, J. (2006), Improving clustering algorithms for image segmentation using contour and region information, *in* '2006 IEEE International Conference on Automation, Quality and Testing, Robotics', Vol. 2, pp. 315–320.

Parzen, E. (1962), 'On estimation of a probability density function and mode', *The Annals of Mathematical Statistics* **33**(3), 1065–1076.
**URL:** *http://www.jstor.org/stable/2237880*

Rahman, A. S. M. M., Tran, T. T., Hossain, S. A. & Saddik, A. E. (2010), Augmented rendering of makeup features in a smart interactive mirror system for decision support in cosmetic products selection, *in* '2010 IEEE/ACM 14th International Symposium on Distributed Simulation and Real Time Applications', pp. 203–206.

Rahman, H., Iyer, S., Meusburger, C., Dobrovoljski, K., Stoycheva, M., Turkulov, V., Begum, S. & Ahmed, M. U. (2016), Smartmirror: an embedded non-contact system for health monitoring at home, *in* 'International Conference on IoT Technologies for HealthCare', Springer, pp. 133–137.

Raspberry Pi Foundation (n.d.*a*), 'Getting started with the camera module', `https://www.projects.raspberrypi.org/en/projects/getting-started-with-picamera/4`. [Online; accessed July-2020].

Raspberry Pi Foundation (n.d.*b*), 'Raspberry pi home page', `https://www.raspberrypi.org`. [Online; accessed July-2020].

Ray, S. & Turi, R. H. (1999), Determination of number of clusters in k-means clustering and application in colour image segmentation, *in* 'Proceedings of the 4th international conference on advances in pattern recognition and digital techniques', Calcutta, India, pp. 137–143.

Rosebrock, A. (2015), 'Access the raspberry pi camera with opencv and python', `https://www.pyimagesearch.com/2015/03/03/accessing-the-raspberry-pi-camera-with-opencv-and-python/`. [Online; accessed July-2020].

Rowley, H. A., Baluja, S. & Kanade, T. (1998), 'Neural network-based face detection', *IEEE Transactions on pattern analysis and machine intelligence* **20**(1), 23–38.

Sebastian, P. & Yap Vooi Voon (2007), Tracking using normalized cross correlation and color space, *in* '2007 International Conference on Intelligent and Advanced Systems', IEEE, pp. 770–774.

*sklearn.cluster.MeanShift* (2019), `https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MeanShift.html#sklearn.cluster.MeanShift`. [Online; accessed May-2020].

Steinley, D. (2006), 'K-means clustering: a half-century synthesis', *British Journal of Mathematical and Statistical Psychology* **59**(1), 1–34.

Szeliski, R. (2010), *Computer vision: algorithms and applications*, Springer Science & Business Media.

Tatiraju, S. & Mehta, A. (2008), 'Image segmentation using k-means clustering, em and normalized cuts', *Department of EECS* **1**, 1–7.

Teeuw, M. (2016), 'MagicMirror$^2$', `https://magicmirror.builders/`. [Online; accessed May-2020].

Teeuw, M. (n.d.), 'Magic mirror', `https://michaelteeuw.nl/tagged/magicmirror`. [Online; accessed August-2020].

*The MagPi- Issue 50* (2020), `https://magpi.raspberrypi.org/issues/50`. [Online; accessed August-2020].

Treepong, B., Mitake, H. & Hasegawa, S. (2018), 'Makeup creativity enhancement with an augmented reality face makeup system', *Computers in Entertainment (CIE)* **16**(4), 1–17.

Viola, P., Jones, M. et al. (2001), 'Robust real-time object detection', *International journal of computer vision* **4**(34-47), 4.

Wren, C. R., Azarbayejani, A., Darrell, T. & Pentland, A. P. (1997), 'Pfinder: real-time tracking of the human body', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **19**(7), 780–785.

Xu, K., Wang, X., Wei, W., Song, H. & Mao, B. (2016), 'Toward software defined smart home', *IEEE Communications Magazine* **54**(5), 116–122.

Yang, C., Duraiswami, R. & Davis, L. (2005), Efficient mean-shift tracking via a new similarity measure, *in* '2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)', Vol. 1, IEEE, pp. 176–183.

Yu, Y., You, S. D. & Tsai, D. (2012), 'Magic mirror table for social-emotion alleviation in the smart home', *IEEE Transactions on Consumer Electronics* **58**(1), 126–131.

Yuan, X.-T., Hu, B.-G. & He, R. (2010), 'Agglomerative mean-shift clustering', *IEEE Transactions on Knowledge and Data Engineering* **24**(2), 209–219.

Zabih, R. & Woodfill, J. (1994), Non-parametric local transforms for computing visual correspondence, *in* J.-O. Eklundh, ed., 'Computer Vision — ECCV '94', Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 151–158.

# Appendix A

# Project Specification

ENG 4111/2 Research Project

## Project Specification

| | |
|---|---|
| For: | Peter Pitt |
| Title: | Smart Shaving Mirror |
| Major: | Electrical and Electronics Engineering |
| Supervisor: | Jason Brown |
| Enrollment: | ENG4111 - EXT S1, 2020<br>ENG4112 - EXT S2, 2020 |
| Project Aim: | To develop, construct and test a smart mirror to assist people while shaving. The mirror will be based on an existing smart mirror platform, running on a raspberry pi. It will use a video camera to provide an enlarged image focusing on the face of the user in the area they are shaving. This mirror will be beneficial for anyone who uses it while shaving weather they are vision impaired or not. |

**Program: Version 2, 8th April 2020**

1. Research the background and current use of smart mirrors.
2. Research current methods of object tracking used in computer vision.
3. Decide on the most appropriate tracking algorithm. Modifications may need to be made to make the algorithm more suitable for the project.
4. Implement the algorithm to smoothly track a razor.
5. Construct and test the smart mirror.

*As time and resources permit:*

1. Improve the functionality by adding another camera to allow the user to view their face from another angle or making the position and angle of the existing camera adjustable.
2. Improve functionality by adding dynamic zoom dependent of the section of the face that is being shaved.

Agreed:

| | |
|---|---|
| Student Name: | Peter Pitt |
| Date: | |
| Supervisor Name: | Jason Brown |
| Date: | |

# Appendix B

# Basic K-means Tracking Algorithm

The code included below was produced to perform the K-means tracking in Python.

## B.1 Python Code for K-means Tracking

Listing B.1: A basic Windows program.

```
# import the necessary packages
from picamera.array import PiRGBArray
from picamera.array import PiYUVArray
from picamera import PiCamera
import time
import cv2
import numpy as np
import math
import random

# size of image arrays
sizeTrack = np.array([94, 160, 3])
sizeDisplay = np.array([368, 640, 3])
acqComplete = False

# Variables for tracking
k = 25
startTime = time.time()

#Create initial cross hairs
def setCrossHairs(image, centre):
    x_position = [int(centre[0] - sizeDisplay[0]/10),
                  int(centre[0]+sizeDisplay[0]/10),
                  int(centre[0]-1), int(centre[0]+1)]
    y_position = [int(centre[1]-sizeDisplay[0]/10),
                  int(centre[1]+sizeDisplay[0]/10),
                  int(centre[1]-1), int(centre[1]+1)]
    #line accross
    image[x_position[0]:x_position[1], y_position[2]:y_position[3], :]
    = [0, 255, 0]
    #line down
    image[x_position[2]:x_position[3], y_position[0]:y_position[1], :]
    = [0, 255, 0]
    return image

#create initial random cluster centres
def randomClusters():
    clusters = np.zeros([2,k])
    for y in range(2):
        for x in range(k):
            clusters[y, x] = int(random.gauss(128, 10))
            while clusters[y, x]>=255 and clusters[y, x]<=0:
                clusters[y, x] = int(random.gauss(128, 30))
    return clusters
```

```
# Pertform basic k-means clustering
def cluster(image):
    steady = False
    grouped = np.zeros([sizeTrack[0], sizeTrack[1]])
    tempClusters = np.zeros([2, k])
    tempClusters.astype(int)
    clusters = randomClusters()
    clusters.astype(int)
    while steady == False:
        clusterCalcs = np.zeros([3, k])
        for x in range(sizeTrack[0]):
            for y in range(sizeTrack[1]):
                tempMinDist = float('inf')
                tempMinCluster = 0
                for z in range(k):
                    tempDist = np.linalg.norm(image[x, y, 1]
                                               - clusters[0, z],
                                               image[x, y, 2]
                                               - clusters[1, z])
                    if tempDist < tempMinDist:
                        tempMinDist = tempDist
                        tempMinCluster = z
                grouped[x, y] = tempMinCluster
                clusterCalcs[0, tempMinCluster] += 1
                clusterCalcs[1:3, tempMinCluster] += image[x, y, 1:3]
        for x in range(k):
            if clusterCalcs[0, x]!=0:
                tempClusters[0, x] = int(clusterCalcs[1, x]
                                     / clusterCalcs[0, x])
                tempClusters[1, x] = int(clusterCalcs[2, x]
                                     / clusterCalcs[0, x])
            else:
                tempClusters[:, x] = clusters[:,x]
        steady = np.array_equal(clusters, tempClusters)
        clusters = tempClusters
    return tempClusters, grouped

def clusterCentre(clustered, target):
    x_pos = 0
    y_pos = 0
    count = 0
    for x in range(sizeTrack[0]):
        for y in range(sizeTrack[1]):
            if clustered[x, y] == target:
                x_pos += x
                y_pos += y
                count += 1
    centre = [(sizeDisplay[0]*x_pos)/(sizeTrack[0]*count),
              (sizeDisplay[1]*y_pos)/(sizeTrack[1]*count)]
    return centre
```

```
def selectCluster(clusters, target):
    dist = 255
    for x in range(0, k):
        tempDist = np.linalg.norm(clusters[0, x] - target[0],
                                  clusters[1, x] - target[1])
        if dist > tempDist:
            dist = tempDist
            cluster = x
    return cluster

# initialize the camera and grab a reference to the raw camera capture
camera = PiCamera()
camera.resolution = (sizeDisplay[1], sizeDisplay[0])
camera.framerate = 20
camera.hflip = True
captureDisplay = PiRGBArray(camera, size=(sizeDisplay[1],
                                          sizeDisplay[0]))
captureTrack = PiYUVArray(camera, size=(sizeTrack[1],
                                        sizeTrack[0]))

# allow the camera to warmup
time.sleep(0.1)

# capture frames from the camera
for frame in camera.capture_continuous(captureDisplay, format="bgr",
                                       use_video_port=True,
                                       resize=(640, 368),
                                       splitter_port=1):
    if startTime>time.time()-5:
        centre = [sizeDisplay[0]/2, sizeDisplay[1]/2]
    else:
        camera.capture(captureTrack, format="yuv", use_video_port=True,
                       resize=(160, 96), splitter_port=2)
        clusters, clustered = cluster(np.array(captureTrack.array))
        if acqComplete == False:
            targetCluster = clustered[sizeTrack[0]/2, sizeTrack[1]/2]
            targetValue = clusters[:, int(targetCluster)]
            acqComplete = True
        else:
            targetCluster = selectCluster(clusters, targetValue)
            centre = clusterCentre(clustered, targetCluster)
    # Display image
    image = setCrossHairs(np.array(captureDisplay.array), centre)
    cv2.imshow("Frame", image)
    # Clear before next frame
    captureDisplay.truncate(0)
    captureTrack.truncate(0)
    key = cv2.waitKey(1) & 0xff
    if key == ord("q"):
            break
```

Appendix C

# Modified K-means Tracking Algorithm

The code included below was produced to perform the K-means tracking with the modifications discussed in Python.

## C.1 Python Code for modified K-means Tracking

Listing C.1: A basic Windows program.

```
# import the necessary packages
from picamera.array import PiRGBArray
from picamera.array import PiYUVArray
from picamera import PiCamera
import time
import cv2
import numpy as np
import math
import random

# size of image arrays
sizeTrack = np.array([94, 160, 3])
sizeDisplay = np.array([368, 640, 3])
acqComplete = False
groupedImage = np.zeros(sizeTrack[0:2])
eDist = np.zeros(sizeTrack[0:2])

# Variables for tracking
k = 25
startTime = time.time()

#Create initial cross hairs
def setCrossHairs(image, centre):
    x_position = [int(centre[0]-sizeDisplay[0]/10),
                  int(centre[0]+sizeDisplay[0]/10),
                  int(centre[0]-1), int(centre[0]+1)]
    y_position = [int(centre[1]-sizeDisplay[0]/10),
                  int(centre[1]+sizeDisplay[0]/10),
                  int(centre[1]-1), int(centre[1]+1)]

    #line accross
    image[x_position[0]:x_position[1], y_position[2]:y_position[3], :]
    = [0, 255, 0]
    #line down
    image[x_position[2]:x_position[3], y_position[0]:y_position[1], :]
    = [0, 255, 0]
    return image

#create initial random cluster centres
def randomClusters():
    clusters = np.zeros([2,k])
    clusters.astype(int)
    for y in range(2):
        for x in range(k):
            clusters[y, x] = int(random.gauss(128, 10))
            while clusters[y, x]>=255 and clusters[y, x]<=0:
                clusters[y, x] = int(random.gauss(128, 30))
    return clusters

# Pertform basic k-means clustering
def cluster(image, clusters, grouped, distances):
    steady = False
    if acqComplete == False:
        grouped = np.zeros([sizeTrack[0], sizeTrack[1]])
        grouped.astype(int)
        distances = np.zeros([sizeTrack[0], sizeTrack[1]])
    tempClusters = np.zeros([2, k])
    tempClusters.astype(int)
    while steady == False:
        clusterCalcs = np.zeros([3, k])
        print('loop')
        for x in range(sizeTrack[0]):
            for y in range(sizeTrack[1]):
                tempMinDist
                = np.linalg.norm([image[x, y, 1]
                                  - clusters[0, int(grouped[x, y])],
                                  image[x, y, 2]
                                  - clusters[1, int(grouped[x,y])]])
                tempMinCluster = grouped[x, y]
                if tempMinDist > distances[x, y]:
                    for z in range(k):
                        tempDist = np.linalg.norm([image[x, y, 1]
                                                   - clusters[0, z],
                                                   image[x, y, 2]
                                                   - clusters[1, z]])
                        if tempDist < tempMinDist:
                            tempMinDist = tempDist
                            tempMinCluster = z
                grouped[x, y] = tempMinCluster
                clusterCalcs[0, int(tempMinCluster)] += 1
                clusterCalcs[1:3, int(tempMinCluster)]
                += image[x, y, 1:3]
                distances[x, y] = tempMinDist
        for x in range(k):
            if clusterCalcs[0, x] != 0:
                tempClusters[0, x] = int(clusterCalcs[1, x]
                                         / clusterCalcs[0, x])
                tempClusters[1, x] = int(clusterCalcs[2, x]
                                         / clusterCalcs[0, x])
            else:
                tempClusters[:, x] = clusters[:, x]
        equal = np.array_equal(clusters, tempClusters)
        clusters = tempClusters
        if equal:
            steady = True
    return clusters, grouped, distances

def clusterCentre(clustered, target):
    x_pos = 0
    y_pos = 0
    count = 0
    for x in range(sizeTrack[0]):
        for y in range(sizeTrack[1]):
            if clustered[x, y] == target:
                x_pos += x
                y_pos += y
                count += 1
    centre = [(sizeDisplay[0]*x_pos)/(sizeTrack[0]*count),
              (sizeDisplay[1]*y_pos)/(sizeTrack[1]*count)]
    return centre

# initialize the camera and grab a reference to the raw camera capture
camera = PiCamera()
camera.resolution = (sizeDisplay[1], sizeDisplay[0])
camera.framerate = 20
camera.hflip = True
captureDisplay = PiRGBArray(camera, size=(sizeDisplay[1],
                                          sizeDisplay[0]))
captureTrack = PiYUVArray(camera, size=(sizeTrack[1], sizeTrack[0]))

# allow the camera to warmup
time.sleep(0.1)

# capture frames from the camera
for frame in camera.capture_continuous(captureDisplay, format="bgr",
                                       use_video_port=True,
                                       resize=(640, 368 ),
                                       splitter_port=1):
    if startTime>time.time()-5:
        centre = [sizeDisplay[0]/2, sizeDisplay[1]/2]
    else:
        if acqComplete == False:
            clusters = randomClusters()
        camera.capture(captureTrack, format="yuv",
                       use_video_port=True, resize=(160, 96),
                       splitter_port=2)
        t = time.time()
        clusters, groupedImage, eDist
        = cluster(np.array(captureTrack.array),
                  clusters, groupedImage, eDist)

        if acqComplete == False:
            targetCluster = groupedImage[sizeTrack[0]/2,
                                         sizeTrack[1]/2]
            acqComplete = True
        else:
            centre = clusterCentre(groupedImage, targetCluster)
        print(time.time()-t)
    # Display image
    image = setCrossHairs(np.array(captureDisplay.array), centre)
    cv2.imshow("Frame", image)
    # Clear before next frame
    captureDisplay.truncate(0)
    captureTrack.truncate(0)
    key = cv2.waitKey(1) & 0xff
    if key == ord("q"):
            break
```

Appendix D

# Cross-correlation Tracking Algorithm

The code included below was produced to perform cross-correlation using the integral image technique in Python.

## D.1 Python Code for Cross-correlation with Integral Image

Listing D.1: A basic Windows program.

```
# import the necessary packages
from picamera.array import PiRGBArray
from picamera.array import PiYUVArray
from picamera import PiCamera
import time
import cv2
import numpy as np
import math
import random
import threading

# size of image arrays
sizeTrack = np.array([94, 160, 3])
sizeDisplay = np.array([368, 640, 3])
acqComplete = False
firstRun = True
trackCentre = [sizeTrack[0]/2, sizeTrack[1]/2]

#t1 = threading.Thread(

#set template size half size used
tSize = int(sizeTrack[0]/20) * 2
displayCentre = [sizeDisplay[0]/2, sizeDisplay[1]/2]


#Create initial tepmlate outline
def setCrossHairs(image, centre):
    centre[0] = centre[0] * sizeDisplay[0] / sizeTrack[0]
    centre[1] = centre[1] * sizeDisplay[1] / sizeTrack[1]
    x_position = [int(centre[0]-sizeDisplay[0]/20),
                  int(centre[0]+sizeDisplay[0]/20)]
    y_position = [int(centre[1]-sizeDisplay[0]/20),
                  int(centre[1]+sizeDisplay[0]/20)]
    #top line
    image[x_position[0]:x_position[1], y_position[1]-2:y_position[1], :]
    = [0, 255, 0]
    #bottom down
    image[x_position[0]:x_position[1], y_position[0]:y_position[0]+2, :]
    = [0, 255, 0]
    #lhs line
    image[x_position[0]:x_position[0]+2, y_position[0]:y_position[1], :]
    = [0, 255, 0]
    #rhs line
    image[x_position[1]-2:x_position[1], y_position[0]:y_position[1], :]
    = [0, 255, 0]
    return image

def setTemplate(template):
    s = template.shape
    t = time.time()
    templateSum = np.zeros([3])
    templateSum[0] = np.sum(template[:, :, 0])
    templateSum[1] = np.sum(template[:, :, 1])
    templateSum[2] = np.sum(template[:, :, 2])
    for z in range(s[2]):
        temp = 0
        for x in range(s[0]):
            for y in range(s[1]):
                temp += template[x, y, z]
        templateSum[z] = temp
    print(templateSum, time.time()-t)
    return templateSum

def runCC(image, center, template):
    #create II
    II = np.zeros(image.shape)
    II.astype(int)
    II[0, 0, :] = image[0, 0, :]
    #t = time.time()
    for x in range(1, sizeTrack[0]):
        II[x, 0, :] = II[x-1, 0, :] + image[x, 0, :]
    for y in range(1, sizeTrack[1]):
        II[0, y, :] = II[0, y-1, :] + image[0, y, :]
    for x in range(1, sizeTrack[0]):
        for y in range(1, sizeTrack[1]):
            II[x, y, :]= II[x, y-1, :] + II[x-1, y, :] - II[x-1, y-1, :]
            + image[x, y, :]
    min_dis = float('inf')
    #CC[0, 0, :] = II[tSize, tSize] - template
    for x in range(sizeTrack[0] - tSize):
        for y in range(sizeTrack[1] - tSize):
            CC = II[x ,y , :] + II[x+tSize, y+tSize, :] - II[x+tSize, y, :]
            - II[x, y+tSize, :] - template
            CC_dis = np.linalg.norm(CC)
            if CC_dis < min_dis:
                min_dis = CC_dis
                templateCentre = [x, y]
    templateCentre[0] += tSize/2
    templateCentre[1] += tSize/2
    return templateCentre

# initialize the camera and grab a reference to the raw camera capture
camera = PiCamera()
camera.resolution = (sizeDisplay[1], sizeDisplay[0])
camera.framerate = 20
camera.hflip = True
captureDisplay = PiRGBArray(camera, size=(sizeDisplay[1], sizeDisplay[0]))
captureTrack = PiRGBArray(camera, size=(sizeTrack[1], sizeTrack[0]))

# allow the camera to warmup
time.sleep(0.1)
startTime = time.time()

# capture frames from the camera
for frame in camera.capture_continuous(captureDisplay, format="bgr",
                                       use_video_port=True,
                                       resize=(640, 368 ),
                                       splitter_port=1):
    if startTime>time.time()-5:
        trackCentre = [sizeTrack[0]/2, sizeTrack[1]/2]
    else:
        camera.capture(captureTrack, format="bgr", use_video_port=True,
                       resize=(160, 96), splitter_port=2)
        trackImage = np.array(captureTrack.array)
        #aquire template
        t = time.time()
        if acqComplete == False:
            trackCentre = [sizeTrack[0]/2, sizeTrack[1]/2]
            template = trackImage[int(trackCentre[0]-tSize/2):
                                  int(trackCentre[0]+tSize/2),
                                  int(trackCentre[1]-tSize/2):
                                  int(trackCentre[1]+tSize/2), :]
            templateVal = setTemplate(template)
            acqComplete = True
        else:
            trackCentre = runCC(trackImage, trackCentre, templateVal)

        print(time.time()-t)
    # Display image
    image = setCrossHairs(np.array(captureDisplay.array), trackCentre)
    cv2.imshow("Frame", image)
    # Clear before next frame
    captureDisplay.truncate(0)
    captureTrack.truncate(0)
    key = cv2.waitKey(1) & 0xff
    if key == ord("q"):
            break
```