Qualification Accredited

OCR
Oxford Cambridge and RSA

A LEVEL

Exemplar Candidate Work

COMPUTER SCIENCE

H446
For first teaching in 2015

H446/03 Summer 2017 examination series

Set A – High

Version 1

Contents

Introduction

These exemplar answers have been chosen from the summer 2017 examination series.

OCR is open to a wide variety of approaches and all answers are considered on their merits. These exemplars, therefore, should not be seen as the only way to answer questions but do illustrate how the mark scheme has been applied.

Please always refer to the specification (http://www.ocr.org.uk/Images/170844-specification-accredited-a-level-gce-computer-science-h446.pdf) for full details of the assessment for this qualification. These exemplar answers should also be read in conjunction with the sample assessment materials and the June 2017 Examiners' Report to Centres available on the OCR website http://www.ocr.org.uk/qualifications/.

The question paper, mark scheme and any resource booklet(s) will be available on the OCR website from summer 2018. Until then, they are available on OCR Interchange (school exams officers will have a login for this).

It is important to note that approaches to question setting and marking will remain consistent. At the same time OCR reviews all its qualifications annually and may make small adjustments to improve the performance of its assessments. We will let you know of any substantive changes.

Exemplar 1 – Set A (High)

Programming project (non exam assessment)

Learners will be expected to analyse, design, develop, test, evaluate and document a program written in a suitable programming language.

Computer Science Coursework – Hand Tracking Application

Contents

1

Computer Science Coursework – Hand Tracking Application

Development and testing

Evaluation

Final code

Analysis

Problem Identification

Currently there is no way to interact with a computer using motion controls without purchasing specific hardware to do so. Motion control has many benefits ranging from educational purposes (for both young students using interactive learning tools, and the elderly who may not be comfortable with using a computer traditionally), to hand muscle rehabilitation through gestures. Motion control can also be used to interact with games in new ways for entertainment.

The features of a computer system that would be required for this solution would a computer with hardware capable of image processing with computer vision, a webcam and a suitable operating system for the software to run on. The problem lends itself to a computational solution because it would be interacting with a computer. Describing it in its simplest form, the solution would take a picture of the user with the webcam, and the software would then identify the hand location and move the mouse cursor to that location, and even could perform other commands.

Stakeholders

The clients and demographic for this software would be users of a computer. Because of the range of applications for motion control the stakeholders will be a representative sample, ranging from older people who struggle with using computers, to gamers who are very familiar that would be able to test it in different circumstances.

The stakeholders for this software represent mostly casual gaming, but there are also people representing accessibility and educational games.

A large amount of casual games can be played using just the mouse and its buttons. Examples of these games are 'Sid Meier's Civilisation V' and the 'Sims' series. These games lend themselves to this software because it will mainly control the mouse movement with the option to add bindings to different actions. The stakeholders for the casual gaming demographic are Ryan Carver and Oliver Gill. Both are experienced gamers that both like casual gaming, especially with titles such as 'Sid Meier's Civilisation V' that will work with this software.

Another type of games that will be appropriate for this software is educational games for younger children where the gameplay relies on interaction and drag and drop mechanics. Allowing them to control the game with gestures can keep them interested in the game and learn for longer. The stakeholders for this area are Sarah Robson and Jon. Sarah is a tutor that has experienced working with small children in a classroom environment where they play these types of educational games. Jon is a five-year-old who is currently in year one, and he would be using this software to interact with the games.

Why it is suited to a computational solution

This problem lends itself to computational methods of finding and implementing a solution for a variety of reasons. The solution will be software that uses a webcam to detect motion and gestures. This will need to run on a computer simply because that is the environment in which the motion control will manipulate, and a computer is needed for the webcam. There is no alternative to solution that would not require a computer.

Computer Science Coursework – Hand Tracking Application

The aspects of the solution that would be controlled by the computer mainly is the webcam recording the user to gather data about where their hands are. The software would then process the image to identify where the hands are, and using this location move the computer's cursor.

Computational methods that the solution lends itself to:

Problem recognition:

The overall problem is finding a way of translating hand movement into mouse movement, whereas the actual underlying problem is being able to identify a hand when provided with an image of the user. When this is overcome the rest of the solution is simply applying the information about where the hand is to another task, in this case moving the mouse. If gestures are involved then not only does the hand need to be identified, but also contoured in a way that gestures can be recognised.

Problem decomposition

This problem can be decomposed into a set of much smaller steps. This is an initial idea of what these steps are:

1. Using the webcam take a picture of the user.
2. Apply a colour filter to identify the location of skin in the image.
3. Using computer vision contouring identify the hand and its location in the image.
4. Using this location, move the mouse of the screen.

When this is done, the process is completed, and it is done fast enough that there is no lag. With computer vision the computer is not normally looking at a video, it is looking at a set of still images and processing them quickly one after another.

Divide and conquer

These smaller steps challenging on their own, but very doable. Solving these steps on their own and then combining them into a modular program makes use of the divide and conquer method of problem solving.

Abstraction

When identifying the hand in an image of the user abstraction is used. An image contains a huge amount of data, almost all of it we don't need. A blur is applied to limit the variation of colour change in a small space. A mask is then applied to delete everything that is not skin coloured, as it is not needed.

The output of the program will be the mouse movement, meaning that overall the software will provide a means of controlling the computer without using the keyboard or mouse directly. This allows a different kind of experience for interacting with the computer that would benefit education games where the interaction would help the child to not get board of the program. It could be used for interacting with casual games for any age of gamers that want a different kind of experience with the game.

Interview

Interview questions

I will outline some of the key questions that I will ask each stakeholder, but during the interview I may ask follow up questions or ask them to elaborate on certain answers. The questions will find their opinion on the software and how they would like to use it.

Casual gaming

My questions for Ryan and Ollie (who represent casual gaming) are as follows:

1. Are you satisfied with your current casual gaming setup?
2. Have you ever experimented with motion controls?
3. If yes, what was your experience and how did they enhance the game?
4. What do you think of using your hands in the air to control the game?
5. How would you like the software to be operated?
6. How would you like to use your hands (gestures etc.)?
7. Do you have anything else to add?

Questions one and two establish their history with gaming and using motion controls. This is important as it is necessary to know if their opinion is based on previous motion control experience or not.

Question 3 investigates any issues they may have had with motion controls if they had used them before, or what make the motion controls successful and fun to play.

From question 4 the questions start to ask about this software. Question 4 asks if they are happy with the main control system of this solution (motion control with moving your hands in the air).

Questions 5 and 6 investigate the usability of the software and how they would like to control it, focusing on both the user interface and the motions that they would have to perform.

The questions conclude with asking if they have anything to say that may have been missed.

Educational

Questions for Sarah (a tutor):

1. Do the children you work with use educational games?
2. If yes, what are the main controls of the games?
3. Do you think children would like motion control?
4. Do you think children would benefit from using motion control?
5. How would you like the software to be operated?
6. What gestures would the children be able to do?
7. Do you have anything else to add?

Questions one and two identify what she thinks are the main aspects of the games that the children use. This can be used when designing what the software can control so that it is compatible with these games.

Questions three and four ask if the children would like to use the motion control in the games, and more importantly whether it would be of use to them for learning, or whether it would just be a distraction.

Questions five and six ask how she would like the software to be operated, considering that children may be difficult to work with if calibration is needed. Questions six asks what gestures the children would be able to do, considering that they would be less able to perform complex gestures and would need more pronounced movements. They might also not be able to remember a large set of gestures.

Questions for Jon (a child):

1. How often do you play computer games at school?
2. Do you enjoy playing them?
3. Would you like to be able to play them by controlling the computer with your hands in the air?
4. What hand movements would you like to do?

These questions are designed to get a feel for how a child would react to this software, and if they think it is a good idea. Because of his age, Jon may not be able to give useful feedback about how the software could work, so instead these ask what he would expect for hand gestures (indicating what gestures children are able to perform).

Interview

Casual gaming

Ryan

1. **Are you satisfied with your current casual gaming setup?**

"Yes, I'm more than satisfied. I've got dual monitors, a mechanical keyboard, and a fancy mouse with loads of buttons, and a headset. I play casual games once a week."

2. **Have you ever experimented with motion controls?**

"Yes I have, I've used the Nintendo Wii that has motion controllers to play the games, but I have not used motion control with my computer."

3. **If yes, what was your experience and how did they enhance the game?**

"The motion controls were not great, you have to calibrate and sync the controllers. I found that it was limited because you had to move in a confined space and could hit things with the controllers. They made the games more accurate and immersive as what you do translated to what happens inside the game, which gives you a form of freedom that you would not get from buttons."

4. **What do you think of using your hands in the air to control the game?**

"If you only use hands to move the cursor it would be a bit limited, more controls would be needed, like using gestures. It seems more like a gimmicky way to play games and does not look viable in terms of playing the game well (like competitively), but would be fun"

5. **How would you like the software to be operated?**

"I would like to be able to turn on the software and have it to work straight away. I would like the option to customise the controls so that I can change gestures to work with different games."

6. How would you like to use your hands (gestures etc.)?

"Different finger movements and different hand positions. Not everything should be static, some gestures could involve movement. I do not want to feel confined by this, so that I can be further from the screen. Because of this I would like to know when my hands are in the camera's frame, so maybe a pop-up could warn you when you move outside the frame."

7. Do you have anything else to add?

"No, I think I have shown what I want."

Ollie

1. Are you satisfied with your current casual gaming setup?

"I think so, I currently have a desktop computer which I control with a keyboard and mouse. Most of the time I play casual games like the Sims, and I mostly use the mouse."

2. Have you ever experimented with motion controls?

"Not yet, but it is something that I plan on looking into. Using motion control would be an interesting way of controlling the games."

3. If yes, what was your experience and how did they enhance the game?

Ollie has not used motion controls before.

4. What do you think of using your hands in the air to control the game?

"Using hands would be a good way of controlling games as you would not need a controller which can be uncomfortable. Because it uses a webcam, something that I already own, I would not need to but an expensive controller. I like the idea."

5. How would you like the software to be operated?

"I'm not very good with setting stuff up, so I would need it to be very simple. It would need to explain what I have to do to control the computer, and what I need to avoid like moving where the camera cannot see me."

6. How would you like to use your hands (gestures etc.)?

"Simple gestures would be better because I do not want to have to remember lots of complicated gestures. I am used to using a smartphone so having similar gestures like pinch to drag would be useful."

7. Do you have anything else to add?

"With consoles when the controllers run out of battery it breaks the game's immersion, and I think this software might have the same affect if it loses where your hands are, so it would need some good calibration so that this doesn't happen."

Computer Science Coursework – Hand Tracking Application

Analysis

Neither of the casual gaming stakeholders have used motion controls with their computers. Ryan had used motion controls with consoles, with his primary complaint being the constant calibration that the controllers needed, and how they could hit objects around him while playing. The solution would need a robust calibration system that would mean that you would not have to constantly re-calibrate the software. Because there are no physical hand held controllers for this, there is no risk of them hitting objects like monitors.

Ryan pointed out that there would need to be a range of controls needed to play most games, and so gestures could be used to activate different key presses (for example making a fist to click the mouse). This implementation would need to be not very complicated, as it may become difficult to set up (as pointed out by Ollie). There needs to be clear instruction on how to set up the software, and an explanation of what you can and can't do.

Educational games

Sarah

1. **Do the children you work with use educational games?**

   "Occasionally, yes. They are used as a reward for the children, so we try not to use them too often, we use them a few times a week normally."

2. **If yes, what are the main controls of the games?**

   "The games are click based, meaning you click on what you want to do or where you want to go and the games that com up are all controlled by the mouse. Because of this we normally use a touch screen white board and let the children play."

3. **Do you think children would like motion control?**

   "I think they would. They love using the touch screen whiteboard, and I think it helps them to stay interested in the game. Motion controls would have the same effect on them, it would make it more fun and interesting, as long as they don't get in the way of the game."

4. **Do you think children would benefit from using motion control?**

   "I think so, it would help them to stay interested in the game and the learning. I wouldn't add much to the game as it can be played fully without motion control, but it would make the overall experience for the class fun."

5. **How would you like the software to be operated?**

   "It would need to be very simple to use, so we do not spend too much time setting it up or working out how it would work. If the children are using it then it would need to be easy to connect to them [calibration]."

6. **What gestures would the children be able to do?**

   "Nothing too difficult, only basic things like making a fist, making their hand flat or pointing their finger."

7. **Do you have anything else to add?**

8

"Not really, I think it would be good so long as it works well and the children don't have difficulty using it."

Jon

1. **How often do you play computer games at school?**

"Every week. We play more if we have been good."

2. **Do you enjoy playing them?**

"Yes, they are very fun. Most of the time we have to watch each other play, but I like playing them more. They are one person at a time games."

3. **Would you like to be able to play them by controlling the computer with your hands in the air?**

"Yeah! It would be like magic!"

4. **What hand movements would you like to do?**

Jon pointed with his index finger and waved it around the screen, saying that that is how he would want to move things. He also made a fist to pick things up.

Analysis

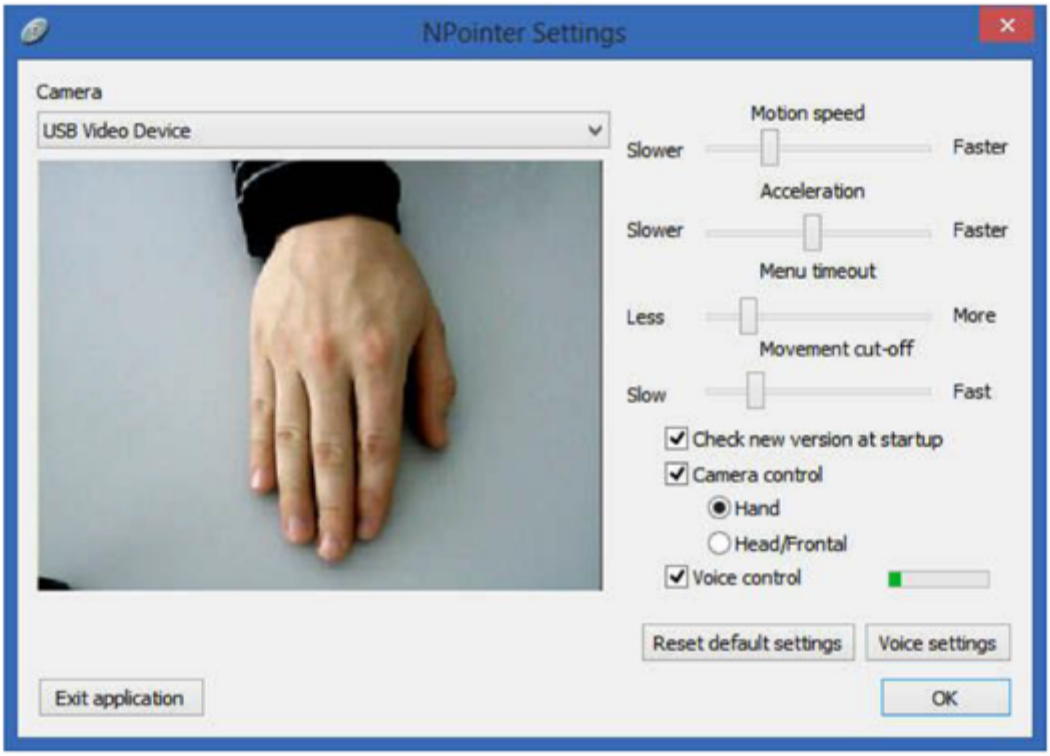
For educational games aimed at children the controls that they need to use are very simple, normally drag and drop, clicking etc. The children also would not be able to make complicated gestures, only making fists and pointing their fingers. Calibration would have to be very easy for the teacher to do so that the child can use the software.

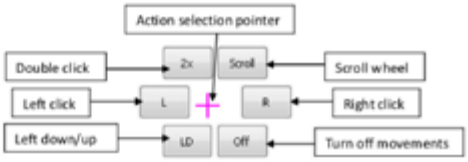
Research

Existing similar solution

NPointer

Overview:





NPointer has an 'action menu' pop-up near the hand.

NPointer is a free gesture control program for Windows XP and higher. Using the webcam and microphone you can control the mouse and other parts of the computer. They recommend using this with a webcam pointed down at a flat surface and then using your hand like you are using a mouse. It also works with the webcam pointed at the user, but can have interference with face detection, and favours using the face to control the mouse instead of the hand, which I found to be inconvenient. The user interface is very simple, consisting only of a window showing the webcam feed with limited menus with sliders that control the movement speed of the mouse and other settings. I found these settings to be lacking as I could not specify some things like gestures.

Parts that I can apply to my solution:

The concept of NPointer is similar to my solution, but my solution will have more of an emphasis on its applications. The window showing the webcam feed was very useful for me to set up the camera so that my hands could be seen, although if I used this in my solution I would have an overlay showing an outline of the hand or something similar to show what the software is gathering from the webcam image. Some of the settings were easy to change, particularly the sliders. The program

seemed lightweight and fast to open and the window was small and did not get in the way, something that I would like to carry forward to my solution.

Flutter

Overview:

Flutter is a gesture control program that focuses only on gestures (no hand tracking) to control the user's media. An example of how someone would use this is to sit in front of their webcam and make a sweeping gesture, indicating t the software to change the song playing. It has support for Windows Media Centre, and other media locations like YouTube. It worked well most of the time, however I found it more of a gimmick and not useful as it only controlled a limited number of things. Media control on its own did not seem like enough to make the software worthwhile. However, it did have good and easy to follow instructions.

Parts that I can apply to my solution:

My solution will also use a webcam to detect gestures, although this is more of a coincidence than something that is inspired by Flutter. The instructions were very clear and simple, consisting of a few images in a slide show format as you open the application. I think that having simple instructions will benefit my solution greatly, especially if I end up implementing a calibration screen.

Leap Motion

Overview:

The Leap Motion is a piece of hardware that used infrared light to detect the location of your hands in a 3D space. This can be used to control games in a 3D way, where your hands can be recreated on screen. The main emphasis of this product is that it detects depth with hand tracking, something that makes it unique. There are many third part applications that use this hardware ranging from games, to accessibility software resembling the software that I will be creating. Some applications allow the Leap Motion to be used to control the mouse of the computer so that it can be used to control any part of the computer, which has many uses for people with accessibility requirements.

Parts that I can apply to my solution:

One of the main purposes of my solution will be that it uses the webcam and has no need for any proprietary hardware, so the leap motion is not very applicable to what I need, as it is a piece of hardware. It gives the impression that it is more of an enthusiast grade product as setting it up is not very straight forward. Despite this I would like my solution to recreate the feel that the Leap Motion has when using it to control the computer.

Intel Realsense 3D Camera

Overview:

The Intel Realsense 3D Camera is a camera that has two parts to it, a traditional camera, and an infrared camera. The infrared camera allows the camera to see depth instead of just a two-dimensional image, opening it up to a range of motion control applications. Support for this camera is mainly in the form of interactive educational games and less so in general computer control.

Parts that I can apply to my solution:

Although the Realsense 3D Camera is a webcam, it is not commonly found in computers and so I would count it as separate hardware that you would need to go out of your way to buy to use, and so it falls into the same category as the Leap Motion. The educational games that are compatible with this camera give an insight into the types of games that could be used with my solution.

Computer Science Coursework – Hand Tracking Application

Touch Screens

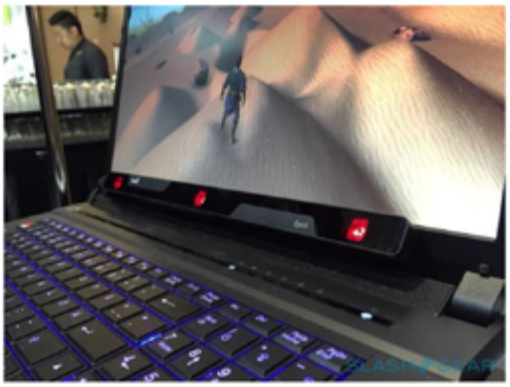
Overview:

Touch screens are a type of screen on a computer that allows the user to touch the screen and move the mouse. Windows has built in touch screen support, so they can be used with any Windows application. They are becoming very common on laptop and all-in-one computers (computer that have the main computer and the screen attached into one item). They allow gesture control via built in windows support. They can be great for motion control, especial with operating systems that were built with that in mind (for example Windows 8).

Parts that I can apply to my solution:

Touch screens are not applicable to my solution as they provide motion control but in a different way that what I need. My solution will never include touch screens as a form of input as any computer that has one already has gesture support.

Steelseries Sentry



Overview:

The Steelseries Sentry is a piece of hardware that tracks the users eye movement and moves the mouse accordingly, or performs another relevant action depending on the application. It is meant for use in gaming with genres such as racing games where you can steer by looking at where you would like to go. It is an expensive piece of hardware that uses USB and drivers. The calibration screen has lots of dots that you stare at to train it to lock onto your eye's gaze. The calibration user interface is simple to follow.

13

Parts that I can apply to my solution:

The Steelseries Sentry is very similar to my solution in that it is a means of controlling the computer without using a keyboard or mouse, and moves the cursor. However, this is designed to work alongside the keyboard and mouse offering people another way to control the computer. It is a separate piece of hardware, which is what I am trying to avid in my solution (apart from generic webcams).

Features of the proposed solution:

Initial concept of my solution considering this research:

My solution will be an application that when started will bring up a simple set of instructions that are easy to skip if necessary, and when open will have a window showing the image that the webcam can see along with an overlay showing what the software can identify (hands or face, etc.). It will have a simple set of sliders that can control basic settings like mouse speed, but will have the option to change some more advanced settings. The software will use the image captured by the webcam to work out where the user's hands are and move the mouse accordingly so that they can control the computer hands-free.

Limitations of my solution:

The main limitation of my solution will be that it does not detect depth, making any gestures limited. It cannot detect depth because it is using a webcam to gather image data. Theoretically it could work out depth using the principal of if the hand is smaller it is further away, but this would be difficult to implement and unreliable.

Another limitation of this solution would be your immediate environment. If you are in a simple, well lit room the software would work well. However, if you are in a busy environment it may be harder for the software to determine where your hands are, and if the lighting is constantly changing (like if you are outside and the sun goes behind the clouds) then it may find it hard to keep tracking the hands.

Further meeting with stakeholders:

This is an email that I sent to my stakeholders (Ryan, Ollie, and Sarah):

"Hi,

I've been looking at ways that I can make the software and have come up with some specific ideas of what it could include, and I would like your input. When the software starts, it will have clear instructions of how to calibrate it (involving making sure you are in the camera image and clicking 'next', nothing technical). When this is complete it will bring you to the main screen of the application (which is a small window intended to be tucked away in the corner). From the main screen, you will have the option to calibrate it again, start and stop the mouse tracking, and have access to simple settings such as the mouse speed. The software will be designed to look neat and will be easy to use. So far it looks like the system requirements will include a fairly capable computer, amongst other software requirements.

Thanks,

David"

These are the responses that I received:

Ryan:

"That sounds good! It has all the basics needed for me to use the software. My computer is a gaming computer, so it is more than capable of running this software. I will probably have the software minimised or hidden behind something else, so the design is not a worry for me, but I like the sound of a simple look for it."

Ollie:

"The calibration sounds complicated, so as long as it is easy to do I'll be okay with it. Maybe have some written instructions with it to tell me what to do if I can't work it out? My computer will probably be able to run this, but if it had an option to reduce the processing power the program needs that would be good."

Sarah:

"I'm pleased that the calibration will not be difficult if I am trying to calibrate it to a child! Other than that, this sounds promising. I definitely like the option of changing the mouse speed"

Computer Science Coursework – Hand Tracking Application

Requirements

Software and hardware requirements

Hardware

A computer capable of running the software, with standard IO peripherals – The software will use the webcam and will use computer vision to analyse the image. A computer with a fast enough processor is needed. Most laptops and desktop computer are capable of this. The standard peripherals include a monitor, keyboard and mouse for navigating the software.

Webcam – A webcam is used to take a video of the user, which will then be analysed to locate their hands. The webcam needs to be of sufficient resolution (around 480p or higher), the higher the resolution the better. If the resolution is very low then when the user moves their hands the mouse may move less smoothly, or jitter.

Software:

Windows, Linux or Mac operating system – These are the operating systems supported by both Python and OpenCV (the computer vision library used to analyse the images).

Python interpreter – The code will be written in Python, so a Python interpreter is needed.

OpenCV for Python - This is the computer vision library that I will be using. It allows simple and efficient analysis of images. This is important for detecting hands.

Numpy for Python – This is a library for python that is used with OpenCV.

Stakeholder requirements

Design

Requirement	Explanation
Simple main window with the webcam feed shown	So that the user can adjust their camera and see what the program 'sees' with hand detection
Lightweight design	The design needs to be simple to sue and not too confusing to use
Clear instructions with diagrams	There needs to be calibration instructions, and images would make this clearer
An obvious way of stopping the mouse movement (like a button or text saying which key to press)	If the user cannot control the mouse properly they need a clear way of stopping it being controlled via the camera
A simple calibration process	The calibration needs to be simple, so the user can just click 'next' and won't have to do anything complicated.

Functionality

Requirement	Explanation
Use of webcam to move the mouse	This is the main function of the program, using the webcam so that you can control the computer's cursor by moving your hands.

16

Computer Science Coursework – Hand Tracking Application

Simple and easy calibration	Because users that may not be familiar with setting up software could be using this a simple calibration is needed so that even children can calibrate it.
A stop button to regain control of the mouse	If the hand detection goes wrong the mouse may start moving too much, so a stop button is needed so that the user can regain control of the mouse and close the program.
Changeable controls linked to gestures, namely a gesture to click the mouse	A set of gestures needs to be mapped to different controls for different games, and for children using the computer who may only be able to do simple gestures, some must have to option to be disabled.
Calibration instructions, and instructions on what you can and can't do when using the software	Instructions are needed to make the calibration process easy to do. Instructions on what you can and cannot do are needed to show the user things like they cannot move their hands outside of the frame.
An option to change the mouse speed	So the mouse can be sped up to be more sensitive, or slowed down to be less sensitive
A settings menu and an information menu	A settings menu that contains all the settings that is in a separate window, and an information window that explains how to use the program and how to calibrate it

Hardware and software

Requirement	**Explanation**
Standard peripherals: Computer with a keyboard, mouse and monitor	The user needs a basic computer to use and navigate the software. The computer must have standard items such as storage and a processor. The program size will be less than a gigabyte.
A USB webcam that is at least 480p	The webcam is used to image the user. The higher the resolution, the smoother the mouse can move.
Computer specs: **[enough to run the software, amend this later]**	The computer needs to be able to run some image processing.
Windows, Mac or Linux operating system	These are the operating systems that are supported by Python.
Python with OpenCV, Numpy and Tkinter	The program will run on Python with OpenCV, which uses Numpy. Tkinter will be used for the user interface.

17

Computer Science Coursework – Hand Tracking Application

Success Criteria

Criteria	How to evidence
Main window that shows webcam feed	Screen shot of main window that shows the webcam feed
Webcam feed shows what the computer 'sees'	Screen shot of the camera feed with image processing overlays
Simple, lightweight design.	Screen shot of the window with large buttons, with the main window not too cluttered. A large and easy to navigate to calibration button
A simple, non-technical calibration	Screen shot showing the calibration with simple instructions and 'next' buttons
A large stop button with colour showing if the program is running	Screen shot of the button with different colours depending on the state of the mouse
The option to press a button to stop the program	Showing the code that facilitates this, along with testing evidence.
Text that shows which button this is	Screen shot of the text on the main window
A settings menu in a separate window	Screen shot of settings button and it's window
The option to change the push to stop button	Screen shot of this option, and code and testing showing that it works
The option to change the mouse speed	Screen shot of this option, and code and testing showing that it works
A low processing power mode for lower-end computers	Screen shot of this option and show the CPU load reduction when this mode is switched on
The option to set gestures	Screen shot of this option and code and testing showing that it works
An information menu in a separate window	Screen shot of information button and its window
Text explain how to use the software and a link to calibration instructions	Screen shot of this text and button
Clear, picture based calibration instructions	Screen shot of these instructions

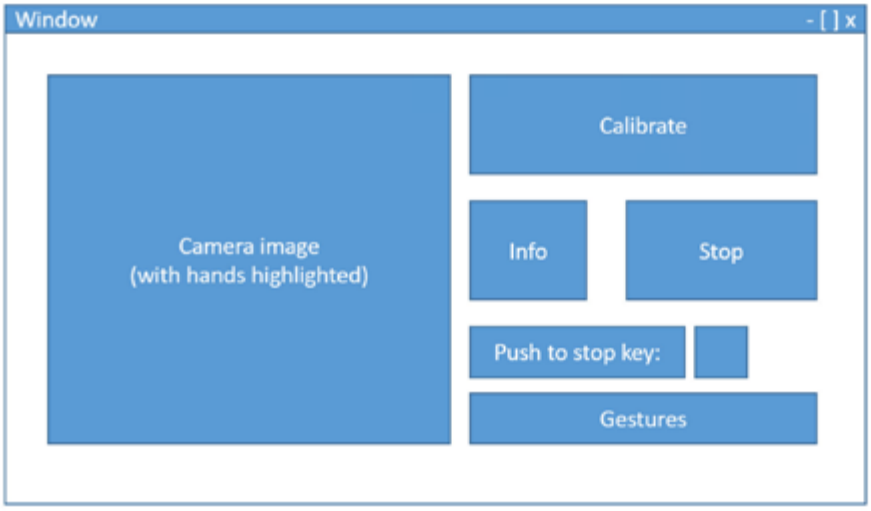
18

Computer Science Coursework – Hand Tracking Application

Instructions easy to find	Screen shot of large buttons that navigate to these instructions, no more than two clicks from the main menu
Moving the hand causes the mouse to move	Evidence of testing that this main feature works

Design

User interface design



This is the main design of the program. This allows the user to see what their camera sees, calibrate the program to recognise their hands, and change the settings. The most important setting, 'Push to stop key' is easily accessible so that the user knows which key stops the hand tracking if they need to re gain the control of the mouse.

Links to the success criteria:

- Main window that shows webcam feed
- Simple, lightweight design.

Camera Image

The camera image box will contain the image that the camera can see, with some image analysis overlaid so that the user can see what the program is doing. The overlaid information would be the outline of the hands detected so the user can easily see if the program needs calibrating or if it is working as intended.

Links to the success criteria:

- Webcam feed shows what the computer 'sees'

Calibrate button

The calibrate button will lead the user to the calibration screen that will guide them through the calibration process that will train the program to recognise the colour of their hands.

Info button

The info button will bring up a screen containing some text about the program. This text will outline how the program works, and how to use it.

Link to success criteria:

- An information menu in a separate window

Stop button

The stop button will stop the program from moving the mouse cursor. This is important so there is always a clear way to stop the automatic cursor movement.

Link to success criteria:

- A large stop button with colour showing if the program is running
- The option to press a button to stop the program

Push to stop key box

This is a text box that assigns the key that will stop the program, it also allows the user to clearly see the push to stop key if they need to. This would be used if the program is not calibrated properly and the user cannot control the mouse to stop the program. In this scenario, the user could hold down a stop key to stop the mouse from being controlled so that they can navigate to the program and stop it.

Link to success criteria:

- Text that shows which button this is

Gestures button

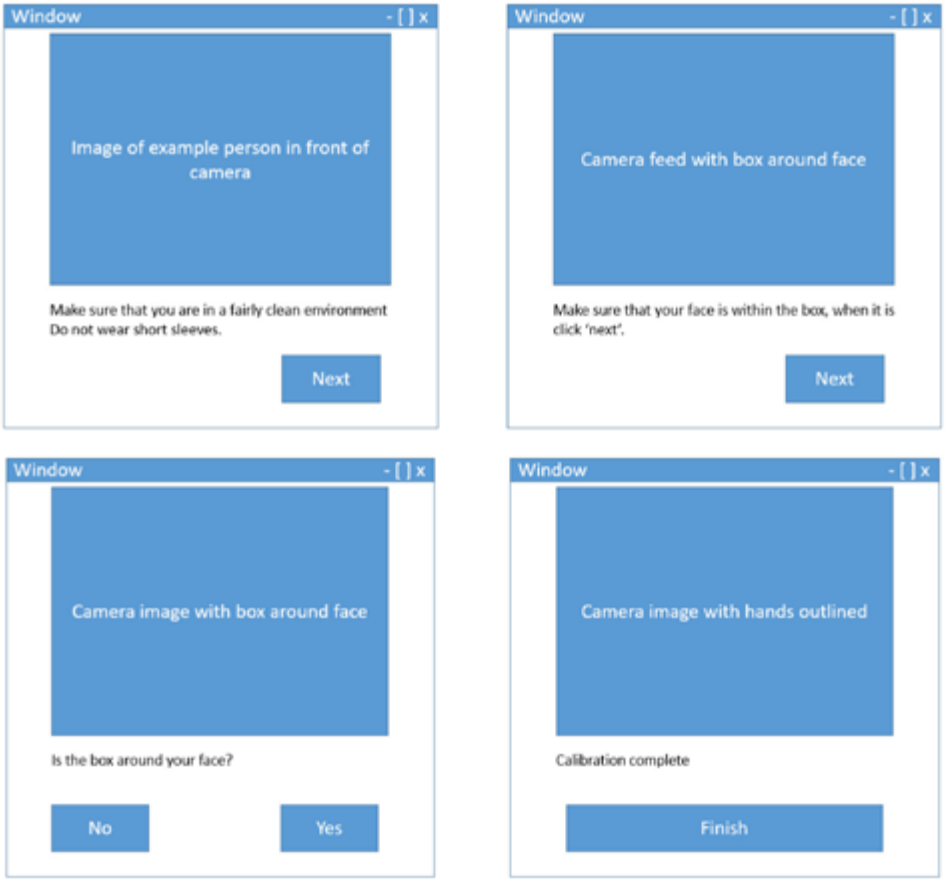
This button will bring up a settings menu that will show what different gestures are assigned to.

Link to success criteria:

- The option to set gestures
- A settings menu in a separate window

Computer Science Coursework – Hand Tracking Application

Calibration



These are designs for the windows that will guide the user through calibration.

The first screen lets them know that they need to be in a clean environment with a clear background, and that they cannot wear short sleeves (this is because the program uses skin colour to determine the hands, and arms would confuse this process). This will be accompanied by an illustration of how their camera should be aligned.

The second screen shows them the face detection. It does this by showing the video feed from the camera with a box around their face. When the box is suitably around their face they will select next and it will take a photo.

The third screen checks that the box placement is correct, if it is they can continue, if not they can repeat the previous screen.

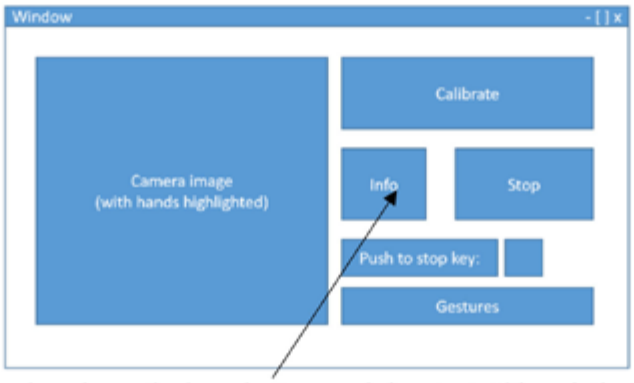
The fourth screen shows them that the calibration is complete by way of showing the camera feed with the hands outlined.

22

Usability features

The usability features that I have considered make sure that the program is easy to use for any type of user. On the screen designs that I have made the buttons are all large to make it easy to see then and click then, especially considering that some of the time the mouse will be controlled by hands (by the program) so aiming may be more difficult.

Any text on the main window needs to be large and easily readable for people with vision difficulties. The main window will also have colour indicators to further show if the program is running (red and green start/stop button). Some people may use screen readers with a text to speech facility, so for this I need to make sure that the text on the program is actual text and not just images so that screen readers are likely to pick up on it (although if the text was in an image OCR, optical character recognition, readers would still be able to read it so long as it is clear).



This design of the window shows the large buttons and clear text. Although the final version may have a different colour scheme, the text will still be visible.

The usage of the program needs to be easy. This involves things like the menu structure not being too complicated. Each menu has its own button so avoid the possibility of a user getting lost in the menus, and if this were to occur they could close the menu (using the red X in Windows) and still keep the main program open.

The calibration process is the biggest focus here, as it has the possibility of being very complicated. I will overcome this by automating it as much as possible and having clear calibration instructions as per the success criteria. The instructions need to be easy to find, which can be quantified by having the instruction no more than two clicks away from the main window.

To try to communicate to the user the processing that is occurring in the background the webcam image will have an overlay that shows what the computer is 'thinking'. This will include an outline to what the computer perceives as the user's hands.

Link to the success criteria

- A simple, non-technical calibration

Stakeholder input

At this point I have a good idea of what the final program will look like and how it will function. I have had a further meeting with the stakeholders to make sure it is what they required and to see if they had any further input.

I sent them all the following email:

"Hi,

Please find attached the diagrams of how the software will could look. I've made sure that the buttons are large and everything is very accessible. The calibration will not be technical, it will show you the webcam image with information such as boxes around the face, and you will have to click next when prompted (for example when there is a box around your face). The computer will handle everything else! I also would like any thoughts on the settings menu. At the moment, I plan on having some settings there, and I'm not sure how complex that would need to be?

Thanks,

David"

These are the replies that I received:

Ryan:

"These designs look good. It seems very simple, which is good. The buttons are very large but I don't really mind as this window will be minimised most of the time. For the settings, as long as there are enough settings for me to get the software to work it's fine!"

Ollie:

"The designs are great, I like the large labels. The calibration look simple enough for me. The settings shouldn't be too complicated, but as long as it works without me having to do anything difficult they should be okay."

Sarah:

"The calibration is perfect, it should be simple enough to do with a child. The design is well labelled, and I'm not worried about the settings, I probably won't use them anyway but if there is anything you think that belongs there I don't mind you adding it."

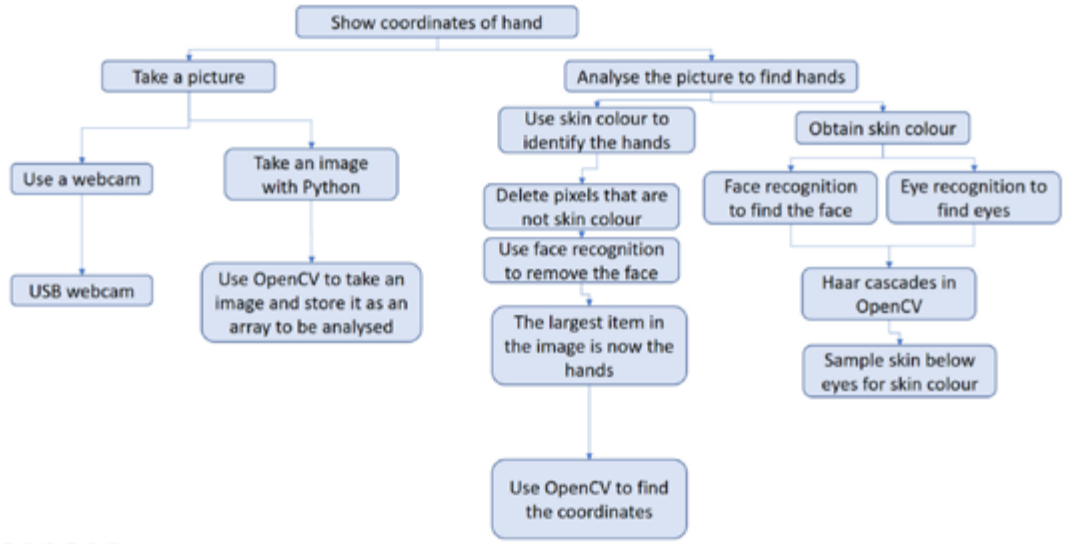
Overall, they are happy with the design and its simplicity. There were no strong feelings about the settings, so I think I will add the necessary settings as I go, provided that they are not complicated.

Algorithms

Apart from the user interface parts of this solution, the main challenge is in the image processing technique using OpenCV. The overall objective for this is:

Take image of the user → Return the coordinates of their hands and gesture

This can be broken down into smaller problems that illustrate how this process will work:



The process can be separated into two sections, the calibration and the hand detection. The calibration only needs to be done once, but the hand detection would be repeated for each frame that the camera captures. Breaking this down into separate subroutines shows the following process:

Calibration:

- Capture an image
- Convert it to HSV colour space and to grayscale
- Use face detection on the grayscale
- Sample the HSV face for skin colour

Hand detection:

- Capture an image
- Convert it to HSV colour space
- Create a mask of the correct areas (using the samples from calibration) and combine them
- Identify the large areas on the mask (the skin) and exclude the head
- Return the location of the hand
- Use contouring to determine the hand gesture
- Return the gesture

Using this data, the program will then move the mouse and perform any keypresses needed.

Each step in more detail:

Computer Science Coursework – Hand Tracking Application

Calibration:

Capture an image:

OpenCV can set the webcam as a capture device and capture an image. The image will be the size of the camera resolution, and is in the RGB colour space. The RGB colour space means that each pixel has three values the amount of red, the amount of green, and the amount of blue (Red Green Blue, hence RGB).

Link to success criteria:

- Main window that shows webcam feed

Convert it to HSV colour space and to grayscale:

Converting it to HSV is necessary for object detection. RGB uses colour values (amount of red, green and blue per pixel) whereas HSV uses Hue, Saturation, and Value. The hue is the colour, saturation is the saturation (high saturation being white, low saturation being the hue colour) and value is the darkness. Because it separates the colour of the pixel into a separate channel (hue), colour detection is more reliable using HSV as opposed to RGB.
Converting it to grayscale is necessary for face detection because of the method used to detect faces and facial features.

Use face detection on the grayscale:

The method that I am using for face detection is using Haar cascades. You provide the cascade with a file that described the object you are looking for (in this case a Haar cascade that describes a face). The image (in grayscale) is then scanned using a function that is built into OpenCV and parts of the image that match this file are recorded, and their coordinates are returned. In the context of this program I will provide it with a face Haar cascade and then it will return the coordinates of any faces in the frame, and their size.

Sample the HSV face for skin colour:

Using the location of the face, sample the face for the skin colour. The nose is important as it has surfaces facing in different directions. This is needed so that the hands can still be detected if they are facing different directions.
The samples need to be taken, and then an upper and lower bound created for each colour channel. This means that for a sample taken that has value [55, 49, 115] with a tolerance of +-10, the upper bound would be [65, 59, 125] and the lower bound would be [45, 39, 105]. Anything that is within this range would be added to the mask when the masks are created.

Hand detection:

Capture an image and convert it to HSV:

This uses the same method mentioned in calibration.

Create a mask of the correct areas and combine them:

Using the image taken from the camera and the samples, a set of masks are created and combined. Any pixel that is within the range of any of the samples is added to the mask. The mask then contains a black and white (not grayscale, literally black or white pixels) image of where the skin in the frame.

Identify the large areas on the mask (the skin) and exclude the head:
Using OpenCV contouring the large 'blobs' on the mask can be detected and put in order of size. The largest contour would be chosen, and then checked to see if it is the head. This is done by comparing its size and coordinates with that of the head (from the face detection). If the contour is not the head, then it can be assumed that it is the hand.

Return the location of the hand:
The centre of the hand can be found as a coordinate value and then returned. This can then be used to move the mouse.

Link to success criteria:

- Moving the hand causes the mouse to move

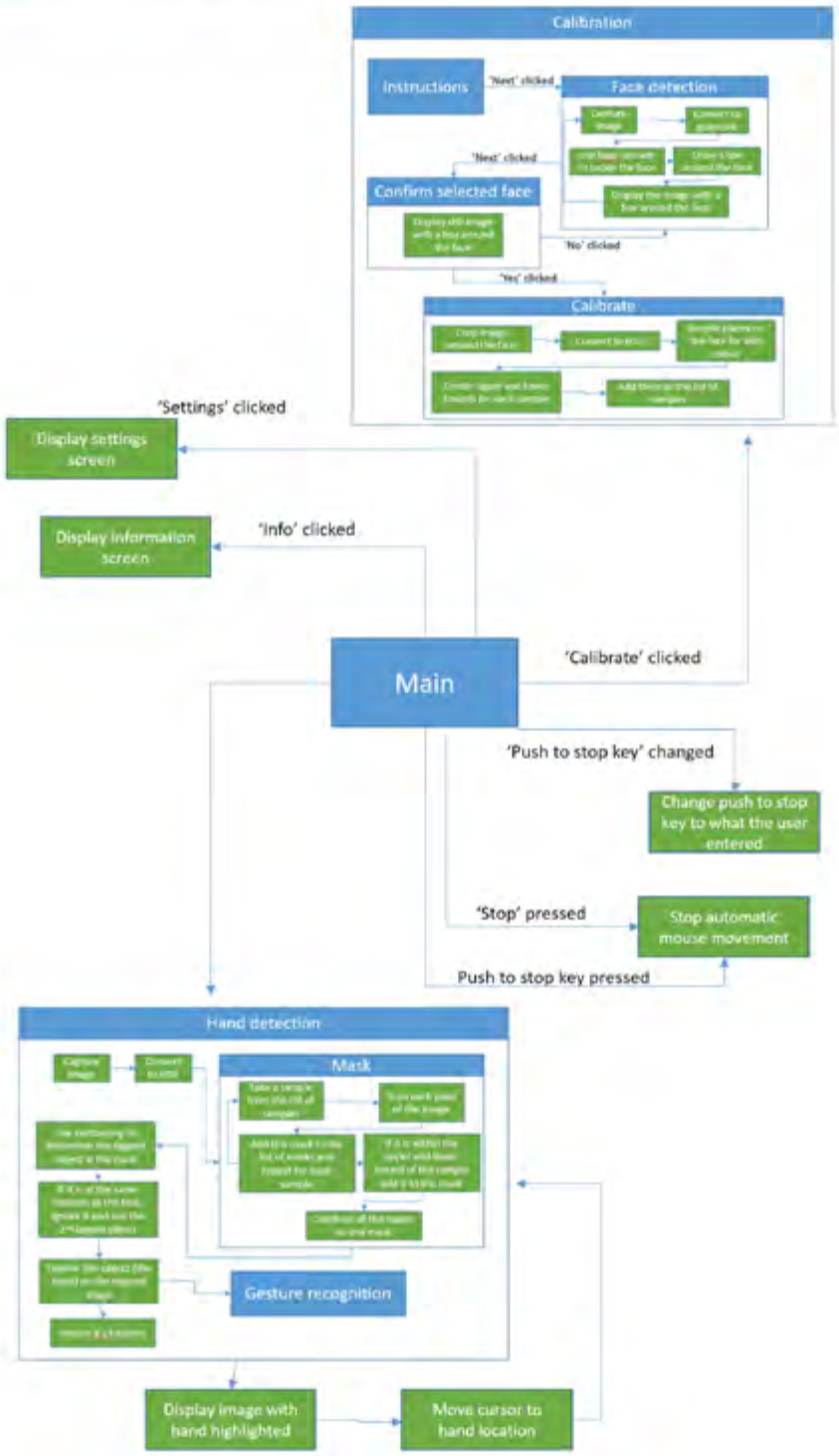
Use contouring to determine the hand gesture:
OpenCV can analyse a contour to determine its concave and convex features. Using this an outline of the hand can be reconstructed and, depending on its shape, a gesture can be identified.

Return the gesture:

Depending on what gesture is detected, return it for the rest of the software to perform the correct action.

Computer Science Coursework – Hand Tracking Application

Diagram showing how the subroutines link

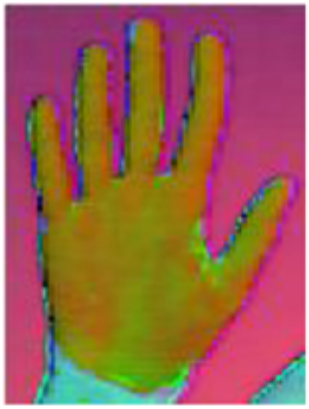


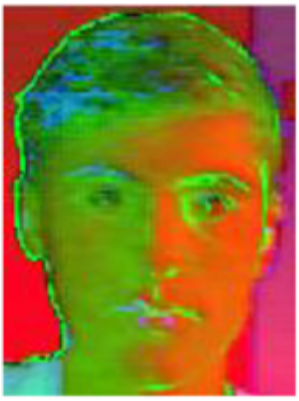
28

Sampling investigation

Part of the hand detection algorithm involves face detection to find the location of the face and sample it for the skin colour so that the program can find the colour of the hands. The face is used because it is a very recognisable feature that, unlike hands, does not change in shape as it is moved. There are many haar cascades available for face detection, that are more robust than ones used for detecting hands. Once the face is detected skin samples need to be taken from areas around the face that show the skin colour at different angles to the light source. We can see which are most useful by examining a HSV image of a person's face and hands to see where the best places to sample are.

On the left is an image of me with my face and hand in frame, notice the light source is from the side, so there are shadows on my hand and face when they are facing different angles. On the right is that same image converted into the HSV colour space, then expressed as if it were RGB so we can see the difference. Looking at it in HSV is important as it is how the program will see it.





The objective is to find a set of places to sample on the face on the right that will give the colours that the hand is made from. From inspecting this it is clear that the cheeks are a good place, and also somewhere across the middle of the face where the shadow changes. Haar cascades can be used to identify facial features such as the eyes and nose, here is the face with those features recognised:

Using these features a set of samples can be located. These are the locations of the samples that the program will collect:

The purple and white dots are where the samples are, and the grey lines show which facial feature they correspond to.

Using these samples a mask can be created that includes all pixels that are roughly the same as the samples, resulting in a mask that looks like this:

The two main features are the face and hand, with some anomalies around the side. Using face recognition, the face can be identified and removed, and then the largest object remaining is the hand.

Subroutines

Now that I know how the hand detection will work I can plan the subroutines that it will use. The program will be made into two files, one that has all of the programming for the image analysis and the other will have all of the user interface programming. This will keep the project more organised. If the user interface needs to interact with the image analysis program then it can call functions in the other program.

Image analysis

This program is split into two sections, calibration and hand detection. The program needs to be calibrated to learn what the skin colour of the user is, and then the hand detection part can find out where the hands are in the image.

Calibration:

FeatureDetect

The program will need to do face and eye detection using haar cascades. This will be a function that when given a grayscale image it returns the location of the face and eyes.

Pseudocode algorithm

```
FUNCTION FeatureDetect(grayscaleImg): #Takes a grasyscale image as an input
        faceCoordinates = OpenCV.FaceHaarCascade(grayscaleImg) #Finds the face in the image
        IF faceCoordinates != 0:#If there is a face...
                faceImg = grayscaleImg(faceCoordinates) #Crops the face
                eyeCoordinates = OpenCV.EyeHaarCascade(faceImg) #Uses the face image to find eyes
                RETURN (faceCoordinates, eyeCoordinates) #Returns the location of face and eyes
        ELSE:
                RETURN (0)#If there isn't a face, return 0
```

CalibrateFaceDetection

This will be the functions that handles the calibration. First it will take an image from the camera, then create a grayscale image, then a HSV image. It will call *FeatureDetect* to find the location of a face and eyes. It will then draw boxes around the face and eyes so the user can see what the program is seeing.

Pseudocode algorithm

```
FUNCTION CalibrateFaceDetection():
        cameraImage = OpenCV.Webcam.TakePicture() #Takes a picture with the webcam
        grayscaleImg = cameraImage.OpenCVGrayscale() #Converts to grayscale
        HSVImg = cameraImage.OpenCVHSV() #Converts to HSV
        faceCoordinates = FeatureDetect(grayscaleImg) #Calls FeatureDetect to find the face
        if faceCoordinates != 0: #If there is a face...
                cameraImage.OpenCV.DrawBox(faceCoordinates) #Draw a box on the original image
```

31

CalibrateSamples

When the user has aligned their face properly after *CalibrateFaceDetection* this function will be called. It will use the face and eye coordinates to take some samples from the cheek of the user's skin colour. Later in the program a pixel will be checked to see if it is the same colour as these samples, but because it is unlikely that it will be exactly the same colour there needs to be a tolerance, so the pixel must be greater than one sample, but less than another. Because of this, upper and lower bounds will be made from the samples in this function.

The program is now calibrated to the user's skin colour.

Pseudocode algorithm

```
FUNCTION CalibrateSamples():

        Samples = [] #Creates samples as a list

        FOR i IN [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]: #A for loop, iterating 'i'

                Samples.ADD(HSVImg(eyeCoords + i)) #Adds to samples the pixel in HSVImg + i
```

Each time the for loop iterates, the value of 'i' increases from 1 to 10. This means that each time a sample is made it is made from a different location on the face, as 'i' increases.

Hand detection:

CreateMask

This function will take an image from the camera and compare it to the samples. For each pixel, if it is within the upper and lower bounds for a sample it will become white, and if not then it will become black. This will create an image that will be black, apart from areas of skin which will be white.

Pseudocode algorithm

```
FUNCTION CreateMask():

        Mask = OpenCV.Image(BLACK) #Creates a black image

        FOR pixel IN HSVImg: #For every pixel in the HSVImage

                FOR sample in Samples: #The pixel is compared to each sample

                        IF pixel = sample: #And if they match,

                                Mask(pixel)==WHITE #The pixel on the mask becomes white
```

This shows how nested FOR loops can compare each pixel to each sample. The result of this is that wherever a pixel matches a sample there is a white pixel on the mask, so the mask is white where skin is present (the hands and face of the user).

FindContours

This function will take the mask image created earlier and find out where the large areas of white are. This will use OpenCV's contour function that will return coordinates of the white areas in order of size. The largest ones will be the hands.

Pseudocode algorithm

This will mainly use the OpenCV function to find the contours of an image, and then use a python function to find the largest of those, and this will be returned.

Computer Science Coursework – Hand Tracking Application

```
FUNCTION FindContours(Mask): #Takes the mask as an input
        Contour = OpenCV.Contours(Mask) #Uses OpenCV to find the contours
        LargestContour = Largest(Contour) #This python command will find the largest contour
        return LargestContour #This contour in then returned
```

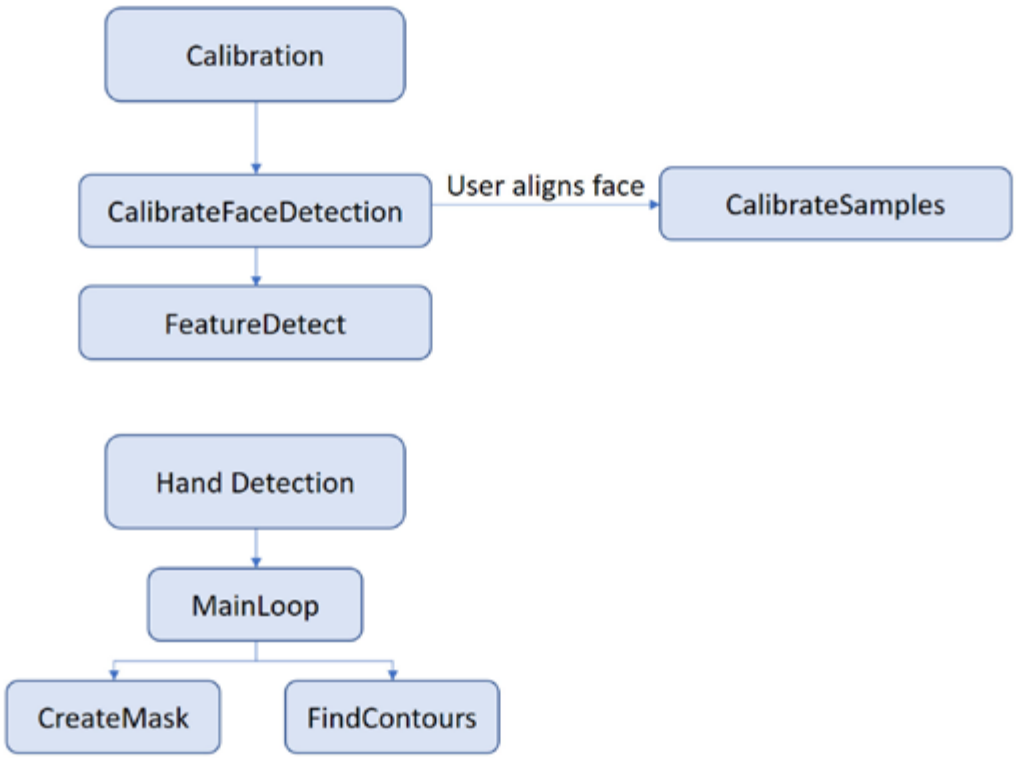
MainLoop

This will be the main loop that will run when the hand tracking is running. It will take an image form the camera, create the necessary grayscale and HSV images from that, and call *CreateMask* then *FindContours*. It will then move the mouse to the location of the largest contour, the hand (depending on the complexity of moving the mouse this section may be moved to a separate function called *MoveMouse*).

Pseudocode algorithm

```
FUNCTION MainLoop():
        cameraImage = OpenCV.Webcam.TakePicture() #Takes a picture from the webcam
        HSVImg = cameraImage.OpenCVHSV() #Converts to HSV
        CreateMask() #Calls the CreateMask function to make a mask
        contourCoordinates = FindContours(Mask) #Calls teh FindContours to find the coordinates of hands
        MoveMouse(contourCoordinates) #Moves the mouse accordingly
```

Here is a diagram showing how these functions link:



33

Testing these algorithms and trace tables

These functions are very different to normal programming as the data and variables that they store are images, and are not data such as integers that lend themselves to trace tables in pseudocode. Because of this testing their functionality will have to be done during development. This will mean there will be a higher testing workload during the development process but this is necessary as the nature of using webcams means that there will need to be small changes and tweaks to the algorithms to make them work correctly. The testing process that I will use is described further in this section.

Explanation and justification of this process

The initial task seems large and complicated, but the way that I have broken it down into separate parts will make development far easier and will aid the maintenance of the code later on as it will be modular.

This is a clarification to how I have broken this problem down; the program will be split into two halves. One half will be the image recognition algorithms, and the other half will be the user interface that handles user inputs and displays the webcam image. The image recognition will be that code that takes pictures from the webcam and analyses them to eventually find the hands in the image.

The image recognition part of the program will have a calibration section that will use face recognition to find the users face and then take some samples of their skin colour, and use this to identify any other skin in the frame, which will be their hands. When the calibration is complete the main part of the image recognition takes place, where it uses these collected samples to continuously find the users hands in the frame and then use this information to move the mouse.

The diagram above can be used to explain the process. *CalibrateFaceDetection* (which uses *FeatureDetect*) is repeatedly called. It finds the location of the face and eyes and highlights them in the image that a user will see. When it is all aligned perfectly the user will give some form of input (in the user interface) and *CalibrateSamples* will be called, which creates the samples. Then the program can start hand detection. *MainLoop* is looped repeatedly which finds the location of the hands and moves the mouse accordingly.

Breaking it down like this ensures that the program is in separate modules that can all communicate to each other. If there is an issue that I need to fix or a change to implement it is easy to find the function responsible and change it without affecting and possible breaking the rest of the program.

This code will be very modular which will help with development and any changes that I want to make later. I will achieve this by following this design and keeping each part of the code in different functions, and splitting the processing and user interface into two separate programs. If another developer were to change the code, this design would make it very easy to make amendments. Because this program will need to communicate with another program (the user interface program) I will use global variables. This will make the code a lot simpler, as each function will not need to parse and return variables, and then the user interface program will not need to call functions to get images, it will just need to call the images.

The largest benefit is the computational thinking that governs this process, as thinking about how to break down the problem ensures that I have thought about each algorithm and how they all work together so that the end program is functional.

Links to the success criteria:

- Webcam feed shows what the computer 'sees'
- A simple, non-technical calibration
- Moving the hand causes the mouse to move

Computer Science Coursework – Hand Tracking Application

Inputs and outputs

Input	Process	Output
Image from webcam	The hand recognition algorithm	Coordinates of the hands in the frame, and some image overlay on the image from the camera
Start button	It will trigger the program to use the windows library in python to change the location of the mouse	The mouse will move according to the hand coordinates
Menu buttons (Info, gestures, calibration)	Tkinter in Python will open a new window with the corresponding menu on	The menus will open on top of the main window
Hand gestures	The gesture recognition algorithm will identify the gesture	A button or action will be performed
The stop button or stop key pressed	The mouse will stop being controlled by the program	The mouse tracking will stop

36

Key Variables

These are the main variables that the program will use for the image analysis:

Name	Data Type	How it is used
ImageBGR	A large 2D array that will store the colour image data.	To store the normal colour image from the camera that the program is working on at any one time. When a new image is taken, this is overwritten. BGR stands for Blue, Green, Red, which is the order that the colours are stored in it.
ImageGRAY	A large 2D array that will store the grayscale image data.	The face detection algorithms only work on a grayscale image. This variable will be used to find the location of the face and eyes.
ImageHSV	A large 2D array that will store the HSV image data.	This will store the HSV colour data for the image, which is used to find all the pixels that are skin colour, then the hands can be found.
FaceCoords	An array with four numbers representing the coordinates of the two opposite corners of the face.	To store the location of the face in the image so that it can be removed, leaving only the hands in the image.
EyeCoords	An array with which stores two more arrays with four numbers representing the coordinates of the two opposite corners of the eyes.	To store the location of the eyes, which is used to sample the face to find the skin colour of the user.
HandCoords	An array storing the x and y coordinates of the hand in the frame.	To store the location of the hand in the frame so that the mouse can be moved to these coordinates.
Samples	An array containing the samples of skin colour.	These samples are taken from the users face and these are used by the program to identify which pixels in the image are skin.
Tolerance	An array with three number.	Each number corresponds to a colour space (like red, green, blue). If a pixel is close to the colour of a sample value, and within this tolerance then it is identified as skin.

Validation

To make sure the program is robust all the user inputs must be validated to make sure they are inputting the correct data. These are the types of data that the user can input and how they will be validated:

Buttons

The buttons that I will use are part of Tkinter, so they are already robust. By the nature of how these work, if they are pressed when they are not linked to anything, nothing will happen. When they are pressed that can link to a function, but they cannot pass any variables so there is no risk of sending incorrect data.

Sliders

Again, these are managed by Tkinter. They have the option to set a range of numbers that the user can slider between. I will have to ensure that the full range on numbers is acceptable for the program and will not cause errors. When this is done, the sliders are completely robust and will not be able to break the program.

Text boxes

Text boxes are different as the user can enter any text they want. The text boxes in this program will be for entering the push to stop key in the settings. The validation that I will have to perform would be to test that the input is only one character, and that it is a valid key on the keyboard (for example numbers 0 to 9, letters a to z). I would also have to make sure it is the correct case, or if it is not I would have to change it to the correct case. By checking the input before the program uses it I can ensure that it will not cause issues.

If the user puts in incorrect data I will need to make sure that there is feedback so that they know what they entered is wrong and they can try again.

Webcam image

The only way that the webcam image could be incorrect is if there was no face in it during the calibration. The program might expect to find one and then crash. To prevent this the image will need to be validated by making sure there is a face in the frame before trying any calibration.

Testing method

Throughout development each function should be tested as it is created to make sure that it works. This means that the program will function when complete, and then any further testing will highlight bugs or feature improvements.

The testing during the development should include the data input to the program (for example the images used) and the result, and if the result is what was intended. Show the program console output if there are any crashes, and for debug purposes print any variables and analyse their content.

After the development is complete a final test should be carried out by me that will include destructive testing and ensuring that the features are there.

Destructive testing will be entering incorrect data and seeing how the program handles it. For user text input I will try a variety of incorrect entries to see what happens. For the images, I will see what happens if I say there is a face when there is not a face in the frame, for example.

I will record any data that I input, and for any video based webcam input I will show screenshots of the input and its result.

After this I will show it to the stakeholders to ensure that it meet's their requirements and to see if they have anything they would like added or amended.

Iterative development

I will be using iterative development to complete this program. This means that I will be breaking the program down into smaller solvable problems (which have already been explained in design) and then solving them in an iterative process. This means that after each stage is completed it will be tested to gradually create a prototype and this process repeats until the program is complete.

This testing will help with this process as it shows how I will test the program as it is being developed and how I will complete a final testing.

Computer Science Coursework – Hand Tracking Application

Visual inputs

For the main part of the program the input will be from the webcam. Due to the nature of this there is no set of exact data that I can prepare for testing, but I can show the type of thing to do to test the program.

For the face recognition and feature recognition (eyes and nose) I could use the webcam pointed at my face and take screenshots showing that my face is highlighted in the correct places. However, for more consistency I could find a picture of a generic face and point the webcam at the image. Here is a generic face image that I found from google:

I could use this image as a direct input to the program, but this would create very consistent and predictable results, whereas with a real face the angle and lighting may affect the program. To accommodate this I will point the webcam at this image or my face, which allows me to change the angle and where the face is in the frame. This is how the face looks when I point the webcam at it:

For a majority of the hand recognition algorithm troubleshooting will be done by using my hands in front of the webcam, and I will document the actions that I performed and some screenshots showing the program and the webcam image.

40

Testing checklist

This is the checklist that I will use to make sure that I have tested all of the functionality of the program:

Action to test	Working?
Check the text for formatting errors	
'Settings' button opens settings menu	
'Info button' opens information menu	
'Calibration instructions' button opens calibration instructions	
'Next' buttons show the next instruction then close the window	
'Calibrate' button opens the calibration window	
'Next' and 'Yes'/'No' buttons work in the calibration sequence	
'Calibration Needed' image shows when not calibrated	
Calibration recognises face and eyes	
If there are errors the 'No' can be used to retry	
Tolerance slider changes the amount of white in the image	
Calibration completes and allows the program to function	
'Calibration Needed' image is replaced by webcam image	
Overlay showing the hand outline and the 'Active Region'	
'Start' button stars the program and turns into a red 'Stop'	
'Stop' button stops the program and turns into a green 'Start'	
The push to stop key stops the program	
Moving the hand causes the mouse to move	
The mouse cannot move when the hand is outside the 'Active Region'	
The horizontal and vertical sliders can be adjusted	
The horizontal and vertical sliders adjust the 'Active Region' correctly	
Changing the push to stop key changes the push to stop key	
Low processing mode stops face recognition	

When this comprehensive list is all complete I will be sure that all the functionality of the program is met and that it has no errors and can be passed onto stakeholders for their final input. At this stage all the criteria on the success criteria will be met, and development complete.

Interface inputs and test tables

Sliders

To change the speed of the mouse there will be sliders that the user can adjust in the settings. To test these sliders, I will change their position and make sure that it creates the desired output. The positions that I will test should include the extreme values of the sliders:

Slider 1	Slider 2	Result
Minimum	Minimum	
Maximum	Maximum	
Minimum	Maximum	
Maximum	Minimum	
Arbitrary mid-range value	Arbitrary mid-range value	

The parts of the success criteria that this testing will satisfy are:

- The option to change the mouse speed

Text boxes

There will be a text entry box to let the user type in an F-key that will become the stop button for the program. Because it will only be an F-key there will only be a certain type of correct inputs (such as F1, f2, F9 etc.) and only F-keys 1 to 9. To test the inputs, I will prepare a series of correct inputs and incorrect inputs to make sure that the program validates properly:

Input	Valid/Invalid	Result
F1, F2, F3, F4, F5, F6, F7, F8, F9	Valid	
f1, f2, f3, f4, f5, f6, f7, f8, f9	Valid	
Test	Invalid	
F10	Invalid	
F1 (space followed by 'F1')	Invalid	
F 5	Invalid	
46	Invalid	

The parts of the success criteria that this testing will satisfy are:

- The option to press a button to stop the program
- The option to change the push to stop button

Buttons

The functionality of the buttons is managed by Tkinter, so they should work without crashing. However, the action they perform is coded by me so I will need to check that they do the correct thing. This will be part of the final testing and it will involve pressing each button in the program and ensuring it does what it is intended to do.

The parts of the success criteria that this testing will satisfy are:

- The option to press a button to stop the program
- A large stop button with colour showing if the program is running

Development and testing

Stage 1, calibration algorithm

The program will be split into two different files. One will contain the user interface (GUI) and the other will handle the image processing. I will start with the image processing.

With OpenCV displaying a camera video is fairly simple to do:

```
import cv2           #imports OpenCV version 2
import numpy as np  #and numpy, which is needed for OpenCV
cap = cv2.VideoCapture(0)    #Assigns the variable 'cap' to the camera
                             #The number notates the port that the camera is
using,
                             #which is 0 by default
while True:                   #Infinite loop to create video
    _, image = cap.read()      #Captures an image from the camera
    cv2.imshow("Image", image)  #Displays the image
    k = cv2.waitKey(30) & 0xff  #Assigns k to a key press
    if k == 27:                  #Stop condition, if esc is pressed (if k is 27)
        break                     #then the loop ends
cv2.destroyAllWindows() #Closes the window
cap.release()           #Releases the webcam so that it can be used by other
programs
```

Calling cap.read() returns the image information and the image itself from the camera. Because I only need the image I have left the image data unassigned by putting in an underscore.

To adapt this for my project I will put this algorithm into a class.

```
import cv2
import numpy as np
class detection():
    def __init__(self):
        self.cap = cv2.VideoCapture(0)
    def showImage(self):
        _, image = self.cap.read()
        cv2.imshow("Image", image)
cameraFeed = detection()
while True:
    cameraFeed.showImage()
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break
cv2.destroyAllWindows()
cameraFeed.cap.realease()
```

When the frame has been converted to grayscale and to HSV different functions in the class will need to read them and change them. To make this easier I have assigned them variables which have scope of the entire class so that I am not always passing arguments.

Computer Science Coursework – Hand Tracking Application

```
import cv2

import numpy as np

class detection():
    def __init__(self):
        self.cap = cv2.VideoCapture(0)#Camera
        self.imageBGR = []#Colour image
        self.imageGRAY = None#Grayscale image
        self.imageHSV = None#HSV image
        self.faceCoords = None#Coordinates of face
        self.eyeCoords = None#Coordinates of eyes
        self.face_cascade =
cv2.CascadeClassifier('haarcascade_frontalface_default.xml')#Face cascade file
        self.eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')#Eye cascade
file
    def faceDetect(self, frame):#input: grayscale frame
        faces = self.face_cascade.detectMultiScale(frame, 1.3, 5)#Haar cascade with
parameters
        if len(faces) > 0:#If it found a face
            largest = 0
            index = []#Largest algorithm
            for i in faces:
                if i[2] * i[3] > largest:#i[2]*i[3] = width*height of face
                    largest = i[2] * i[3]
                    index = i
            self.faceCoords = index#output: assigns self.faceCoords to coordinates
[x, y, w, h] of largest face in frame
        else:
            self.faceCoords = []
    def featureDetect(self, frame):#input: grayscale frame
        eyes = self.eye_cascade.detectMultiScale(frame)#Haar cascade with no
parameters
        self.eyeCoords = eyes#Output: assigns self.eyeCoords to the eye coordinates
in the form [x, y, w, h]
    def calibrateFaceDetection(self):#Takes an image, returns image of user with
face, eyes highlighted
        _, self.imageBGR = self.cap.read()
        self.imageGRAY = cv2.cvtColor(self.imageBGR, cv2.COLOR_BGR2GRAY)
        self.imageHSV = cv2.cvtColor(self.imageBGR, cv2.COLOR_BGR2HSV)
        self.imageHSV = cv2.blur(self.imageHSV, (5, 5))
        self.faceDetect(self.imageGRAY)
        if len(self.faceCoords) > 0:#If there is a face
            [x, y, w, h] = self.faceCoords
            cv2.rectangle(self.imageBGR, (x, y), (x+w, y+h), (255, 0, 0), 2)#Draw's
a rectangle aroudn face
            self.featureDetect(self.imageGRAY[y:y+h, x:x+w])#calls featureDetect
for eye detection
            for [ex, ey, ew, eh] in self.eyeCoords:
                cv2.rectangle(self.imageBGR, (ex + x, ey + y), (ex + x + ew, ey + y
+ eh), (0, 255, 0), 2)#Draws a rectangle aroudn the eyes
            if len(self.eyeCoords) != 2:#If there are not two eyes, prints and
error message
                print "Eye count error"
        return self.imageBGR#returns the image
```

This is the complete first stage of the calibration. The changes made here are the main function calibrateFaceDetect has been expanded, and the face and eye detection have been separated into separate functions (faceDetect and featureDetect). The featureDetect function finds the eyes within the face, and is called feature detect because there is the possibility that I will add other facial features for it to detect (like the nose), but at the moment I think that the eyes are enough.

Some class variables have also been declared. The image variables are imageBGR, imageGRAY and imageHSV. The coordinate variables for the location of the face and eyes are faceCoords and eyeCoords. The haar cascades also have their own variables. These are found within the __init__

44

function of the class, the function that is run when the class is initialised so that these variables are assigned upon the class creation.

When calibrateFaceDetect is called it will return an image from the camera with the face and eyes highlighted (with a coloured rectangle around them). It first takes a picture from the webcam (imageBGR) and converts it to grayscale (imageGRAY) and HSV (imageHSV). The HSV image is then blurred (this is for later on in the program when imageHSV will be used to sample the face, blurring makes more accurate samples due to less anomalies). faceDetect is then called with imageGRAY as the argument.

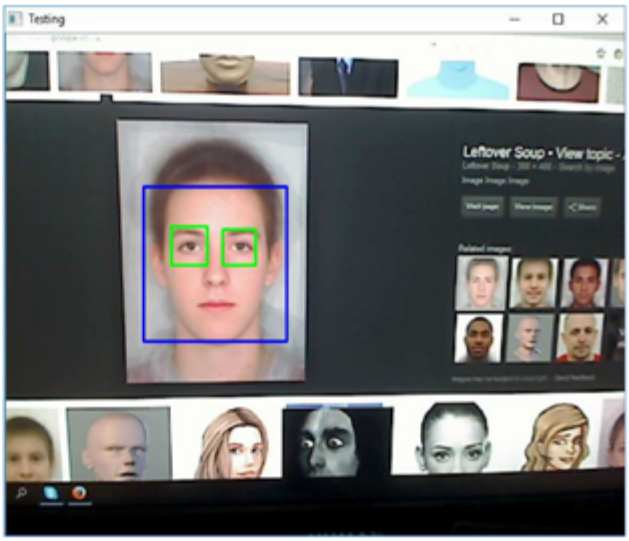
faceDetect completes a haar cascade of the image using the face haar cascade. If it finds a face it then sorts through them to find the largest, and assigns the coordinates of the largest face to faceCoords.

The main function, calibrateFaceDetect, then places a rectangle around the face using the face coordinates (faceCoords). It then crops the image around the face and calls featureDetect, which looks for any eyes. It crops the image so reduce the amount of processing it has to do. Any eyes in the frame are going to be on the face, so giving featureDetect just the face to search is more efficient than making it search the whole frame. The eyes are then highlighted, and the image is returned.

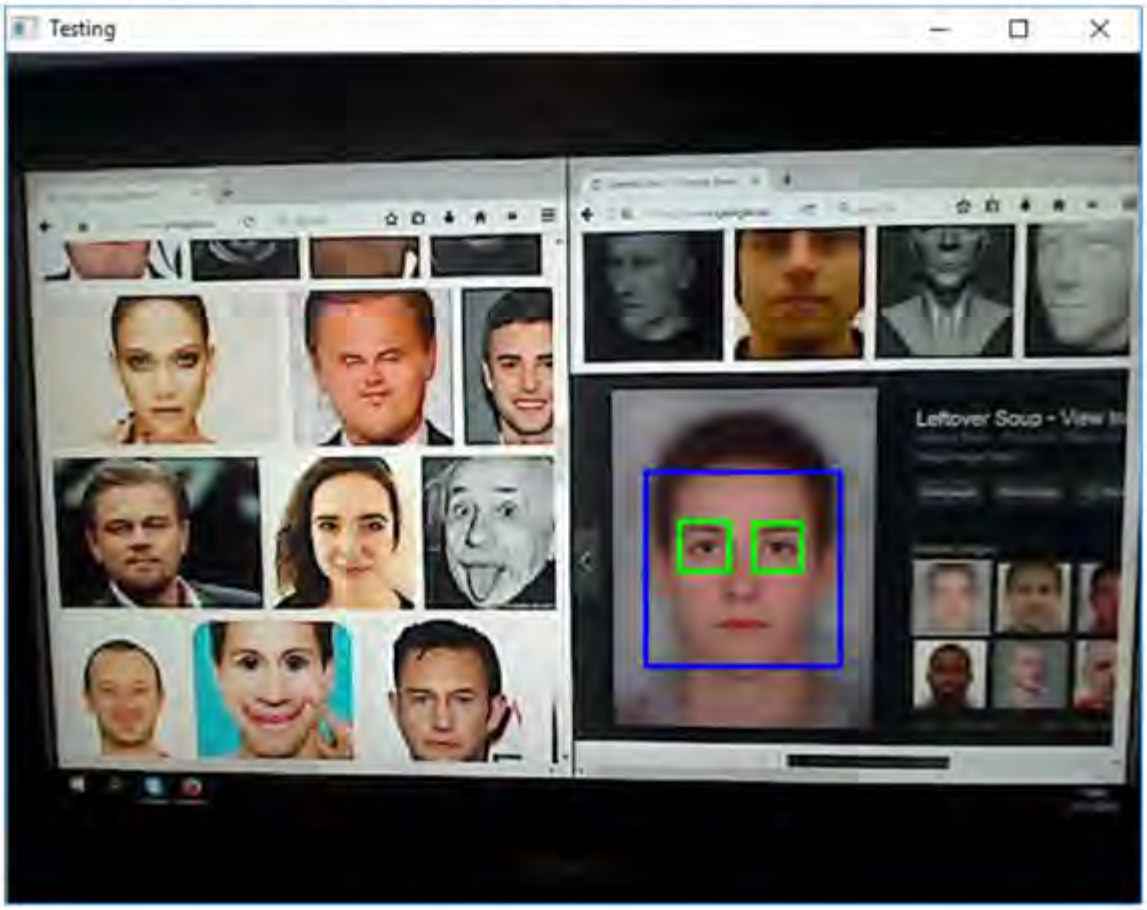
To test this I added a small loop at the end of the program that runs this code. As a face I searched for images online of generic faces. These are the results:

```
test = detection()

for i in range(100):
    cv2.imshow("Testing", test.calibrateFaceDetection())
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break
cv2.destroyAllWindows()
test.cap.release()
```

This part of the code functions exactly as it is meant to as shown by this image. The face is found and it's eyes are also highlighted. When handling multiple faces in one frame it should only highlight the largest face, and only that face's eyes.



This is how it handles multiple faces (only uses the largest face in the frame).



The next step is searching the face for samples of the skin colour. To do this I created a new function within the class called calibrateSamples, this is the second stage of the calibration process. For this I needed to add some more class variables, rawSamples, samples, and tolerance. When the program picks a spot on the image it samples it by looking at it's colour value (in HSV). This will be an array with three numbers representing Hue, Saturation, and Value respectively (can also be explained as Colour, Darkness, Lightness). This is stored in rawSamples as it is the raw image data taken from the face. When the software wants to find skin in the image it will want to find pixels that are almost the same colour as this, but not exact. This is the tolerance. For example, if it is looking for a pixel with H value 100, but anywhere between 90 and 110 is okay then the tolerance is +- 10. Each sample will be split into it's upper and lower bounds. Any pixel has to have values that are higher than the lower bound but lower than the higher bound for them to be interpreted as skin. They are stored this way because of how OpenCV's mask function works, it needs to be provided with an upper and lower bound, and calculating those now with the other samples makes more sense than to leave it until later on in the program.

Computer Science Coursework – Hand Tracking Application

```
def __init__(self):

    self.cap = cv2.VideoCapture(0)#Camera
    self.imageBGR = []#Colour image
    self.imageGRAY = None#Grayscale image
    self.imageHSV = None#HSV image
    self.faceCoords = None#Coordinates of face
    self.eyeCoords = None#Coordinates of eyes
    self.rawSamples = []#Samples collected from image [?]
    self.samples = []#Samples corrected with the tolerance
    #                  V   S   H
    #                  L   D   C
    self.tolerance = [30, 30, 30]#Tolerance of upper and lower sample bounds
    self.face_cascade =
cv2.CascadeClassifier('haarcascade_frontalface_default.xml')#Face cascade file
    self.eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')#Eye cascade
file
```

This is the revised __init__ function. The tolerance is labelled clearly because it's values will be tweaked later on in the development.

```
def calibrateSamples(self):

    [x, y, w, h] = self.faceCoords
    imageFace = self.imageHSV[y:y + h, x:x + w]
    by = 0  # biggest ey
    sx = 10000  # smallest ex, arbitrarily large number
    bx = 0  # biggest ex
    for [ex, ey, ew, eh] in self.eyeCoords:
        # cv2.rectangle(imageFace, (ex, ey), (ex+ew, ey+eh), (0, 0, 0))
        if ey + eh > by:
            by = ey + eh
        if ex < sx:
            sx = ex
        if ex + ew > bx:
            bx = ex + ew
    for i in range(1, (bx - sx) // 10 + 1):
        self.rawSamples.append(imageFace[by][sx + i * 10])
        upper, lower = [], []
        for x in range(len(self.tolerance)):
            upper.append(imageFace[by][sx + i * 10][x] + self.tolerance[x])
            lower.append(imageFace[by][sx + i * 10][x] - self.tolerance[x])
        for y in range(len(upper)):  # so that 0<upper<255
            if upper[y] < 0:
                upper[y] = 0
            if upper[y] > 255:
                upper[y] = 255
        for y in range(len(lower)):  # so that 0<lower<255
            if lower[y] < 0:
                lower[y] = 0
            if lower[y] > 255:
                lower[y] = 255
        lower = np.array(lower, dtype=np.uint8)
        upper = np.array(upper, dtype=np.uint8)
        self.samples.append([lower, upper])
```

This is the calibrateSamples function. It takes the coordinates of the face and creates a cropped image that is just the face. This new variable does not have a 'self.' infront of it because it does not need to have scope of the entire class as it is just being used for this function. It then has an algorithm to find the largest numbers in the eye coordinates. Specifically, it is finding the lowest point across both eyes (ey + eh means the y coordinate plus the height of the eye, and since the origin for the image coordinate system is in the top left a large y is lower down in the frame). 'sx' is

the leftmost x coordinate. 'bx' is the rightmost x coordinate. These values are important as these are what I will be using to collect the samples without using the nose. Using the image taken earlier as part of the testing, this is how I am collecting samples:

The pink dotted line is the line that the samples will be taken from. The box that surrounds the eyes is larger than I originally thought, and because it goes about halfway down the nose the nose does not need to be used to find samples, as this dotted line spans the face perfectly and catches a variety of skin shade and shadowing.

The code then cycles through ten x coordinate values equally spaced out along that line. This is done through the for loop:

```
for i in range(1, (bx - sx) // 10 + 1):
```

For each sample, the sample is added to rawSamples and then the upper and lower bounds are created. The raw sample is stored so that it can be printed for testing purposes. The tolerance is then added to form the upper bound, and subtracted to form the lower bound. Because these are not integers, but three value arrays another loop is used.

```
upper, lower = [], []

for x in range(len(self.tolerance)):
    upper.append(imageFace[by][sx + i * 10][x] + self.tolerance[x])
    lower.append(imageFace[by][sx + i * 10][x] - self.tolerance[x])
```

An algorithm is then used to make sure that each value is still within 0 to 255 (the only values allowed in OpenCV image data). The upper and lower bounds are then converted to 'uint8' datatype so that they will work with OpenCV masking later on. They are then added to the list of samples:

```
for y in range(len(upper)):   # so that 0<upper<255

    if upper[y] < 0:
        upper[y] = 0
    if upper[y] > 255:
        upper[y] = 255
for y in range(len(lower)):   # so that 0<lower<255
    if lower[y] < 0:
        lower[y] = 0
    if lower[y] > 255:
        lower[y] = 255
lower = np.array(lower, dtype=np.uint8)
upper = np.array(upper, dtype=np.uint8)
```

Computer Science Coursework – Hand Tracking Application

```
self.samples.append([lower, upper])
```

Now this function is complete, it should scan the HSV image in the class for samples, create the lists of samples and not return anything. To test if it is working I had it print the values of samples and rawSamples:

```
test = detection()

for i in range(1000):
    cv2.imshow("Testing", test.calibrateFaceDetection())
    test.calibrateSamples()
    print test.rawSamples
    print test.samples
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break
cv2.destroyAllWindows()
test.cap.release()
```

This is part of the data that was the result:

```
[array([ 30,  28, 170], dtype=uint8), array([ 33,  24, 169], dtype=uint8), array([ 33,  26, 167], dtype=uint8), array([ 24,  27, 171], dtype=uint8), array([ 24,  35, 165], dtype=uint8),
array([ 20,  91, 132], dtype=uint8), array([ 16, 122, 106], dtype=uint8), array([ 14,  90, 133], dtype=uint8), array([ 16,  89, 135], dtype=uint8), array([ 23,  35, 177], dtype=uint8),
array([ 23,  31, 174], dtype=uint8), array([ 23,  34, 171], dtype=uint8), array([ 21,  35, 177], dtype=uint8), array([ 18,  45, 173], dtype=uint8), array([ 14,  97, 130], dtype=uint8),
array([ 15,  90, 139], dtype=uint8), array([ 15,  91, 139], dtype=uint8), array([ 15,  95, 133], dtype=uint8), array([ 17,  90, 141], dtype=uint8), array([ 14, 118, 108], dtype=uint8),
array([ 22,  33, 177], dtype=uint8), array([ 25,  28, 175], dtype=uint8), array([ 24,  32, 174], dtype=uint8), array([ 21,  35, 178], dtype=uint8), array([ 18,  45, 175], dtype=uint8),
array([ 23,  31, 177], dtype=uint8), array([ 23,  31, 173], dtype=uint8), array([ 21,  37, 166], dtype=uint8), array([ 23,  33, 178], dtype=uint8), array([ 19,  46, 173], dtype=uint8),
array([ 14, 107, 122], dtype=uint8), array([ 16,  82, 142], dtype=uint8), array([ 22,  45, 149], dtype=uint8)
```

Extracting the values for the first few samples:

Raw Sample: [30, 28, 170]

Upper: [60, 58, 200]

Lower: [0, 0, 140]

The tolerance for this was set to 30. This shows that the function works, and it also showed that the algorithm that keeps the upper and lower bound greater than 0 and less than 255 works, as 28 – 30 is -2, but the value stayed at 0.

The calibration part of the program is complete. The next stage is to create a part of the program that can apply this calibration to find the hands in the image.

A function is needed that will take an image taken from the camera, and look at the samples and return a black and white (note that this is in the literal sense, not grayscale) mask showing the areas of the image that contains skin. Because most of the work with calculating and converting the samples was done as part of the calibration, the mask is fairly simple to create.

```
def createMask(self):

    self.mask = cv2.inRange(self.imageHSV, self.samples[0][0], self.samples[0][1])
#just to create mask of correct size
    for i in self.samples:
        tempMask = cv2.inRange(self.imageHSV, i[0], i[1])
        self.mask = cv2.bitwise_or(self.mask, tempMask)
```

The mask variable has been added to the __init__ function.

Computer Science Coursework – Hand Tracking Application

```
def __init__(self):

    self.cap = cv2.VideoCapture(0)#Camera
    self.imageBGR = []#Colour image
    self.imageGRAY = None#Grayscale image
    self.imageHSV = None#HSV image
    self.faceCoords = None#Coordinates of face
    self.eyeCoords = None#Coordinates of eyes
    self.rawSamples = []#Samples collected from image [?]
    self.samples = []#Samples corrected with the tolerance
    #                  V   S   H
    #                  L   D   C
    self.tolerance = [30, 30, 30]#Tolerance of upper and lower sample bounds
    self.mask = None#Mask of pixels that are within the range of the samples
    self.face_cascade =
cv2.CascadeClassifier('haarcascade_frontalface_default.xml')#Face cascade file
    self.eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')#Eye cascade
file
```

It starts off by using the OpenCV inRange function. This is a function that takes three arguments, the image, and the upper and lower bounds. It returns a black and white mask of all the pixels in the original image that are within those bounds. The code starts off by creating a mask with the image and the first of the samples. It then loops through the rest of the samples create a mask for each sample, and adds it to the first one as it goes (the bitwise_or function). This function takes each pixel from the two images you supply it as performs a bitwise or on them and returns a combination of the two. A truth table for how it compares each pixel looks like this:

Image 1	**Image 2**	**Output**
White	White	White
White	**Black**	White
Black	White	White
Black	**Black**	**Black**

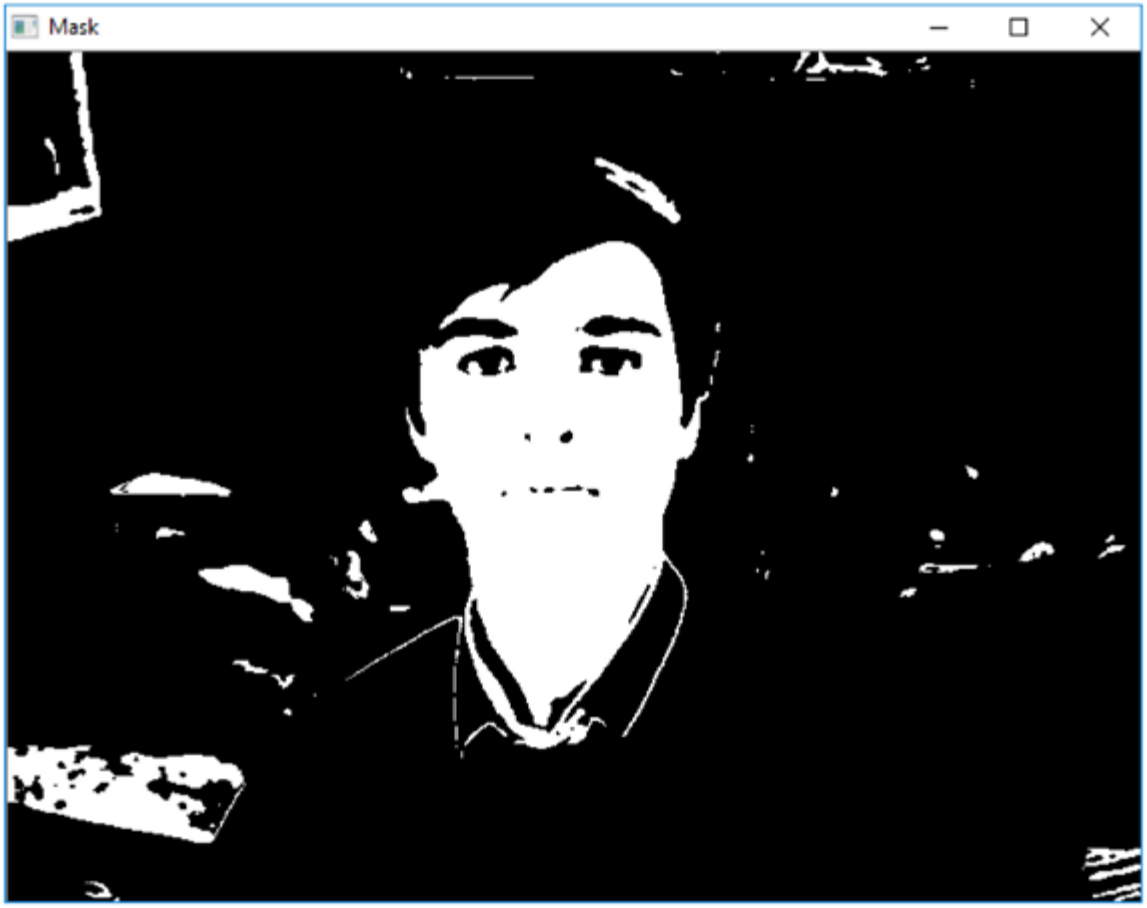
When this function finished the mask of the image will be stored in the mask variable.

This is the code that I used to test this function:

```
test = detection()

for i in range(100):
    cv2.imshow("Testing", test.calibrateFaceDetection())
    test.calibrateSamples()
    test.createMask()
    cv2.imshow("Mask", test.mask)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break
cv2.destroyAllWindows()
test.cap.release()
```

50

And this was the result:



This works perfectly, and the amount of items in the background that get picked up can be adjusted by changing the tolerance variable. This can be done through trial and error.

When testing I came across this error which caused the program to crash:

```
"C:\Users\David Robson\Anaconda2\python.exe" "C:/Users/David Robson/Documents/Work/Computing/OpenCV stuff/Programs/Coursework/tEST.py"
Traceback (most recent call last):
  File "C:/Users/David Robson/Documents/Work/Computing/OpenCV stuff/Programs/Coursework/tEST.py", line 98, in <module>
    test.calibrateSamples()
  File "C:/Users/David Robson/Documents/Work/Computing/OpenCV stuff/Programs/Coursework/tEST.py", line 55, in calibrateSamples
    [x, y, w, h] = self.faceCoords
ValueError: need more than 0 values to unpack

Process finished with exit code 1
```

This is an error that occurs when calibrateSamples is called. The error occurs when there is not face in the frame. The program is trying to look for face coordinates when there are none. To fix this I have added an if statement to the loop of code that runs the class. I did not add the if statement into the function itself because then I would have had to do that for every function that is called after it, and it is more efficient to place the if statement in the loop of code at the end.

```
test = detection()

for i in range(100):
    cv2.imshow("Testing", test.calibrateFaceDetection())
    if len(test.faceCoords) != 0:
        test.calibrateSamples()
        test.createMask()
        cv2.imshow("Mask", test.mask)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break
cv2.destroyAllWindows()
test.cap.release()
```

Now that the mask works I need to write a section that will analyse the mask to find the hands.

The findContours function will make use of OpenCV's findContours function. The function looks at a black and white image and looks for contours within it (boundaries between white and black areas, in this case the outline of the hands). It forms a list of all the contours in the image and returns it. It takes two arguments which refer to the process that it will use to find the contours. The process That I have chosen uses cv2.CHAIN_APPROX_SIMPLE which finds the contours and uses less processing power than some alternatives.

```
def findContours(self):

    _, contours, _ = cv2.findContours(self.mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    if len(contours) != 0:
        largest = 0
        contour = None
        for cnt in contours:
            x, y, w, h = cv2.boundingRect(cnt)
            if w*h > largest:
                largest = w*h
                contour = cnt
        cv2.drawContours(self.imageBGR, [contour], 0, (0, 255, 0), 2)
        hx, hy, hw, hh = cv2.boundingRect(contour)
        self.handCoords = [hx + hw//2, hy + hh//2, hw*hh]
```

This is the findContours function. Using OpenCV's findContours function it has a list of the contours. Making the assumption that the hands are going to be the largest contour in the frame I have used an algorithm to find the contour with the largest area (w*h, or width multiplied by height). It then draws a line around this largest contour. Finally, it finds the centre of this contour and assigns the variable handCoords to these coordinates, these are the coordinates of the hand in the frame.

This function relies on the image having already been taken from the camera, meaning that I will need to expand the loop of code used for testing. Instead, I will make a function within the class that will tie together all these functions to make testing easier.

```
def temploop(self):

    _, self.imageBGR = self.cap.read()
    self.imageHSV = cv2.cvtColor(self.imageBGR, cv2.COLOR_BGR2HSV)
    self.imageHSV = cv2.blur(self.imageHSV, (5, 5))
    self.createMask()
    self.findContours()
```

This function then turns the testing loop from this:

Computer Science Coursework – Hand Tracking Application

```
test = detection()

for i in range(100):
    cv2.imshow("Testing", test.calibrateFaceDetection())
    if len(test.faceCoords) != 0:
        test.calibrateSamples()
        _, test.imageBGR = test.cap.read()
        test.imageHSV = cv2.cvtColor(test.imageBGR, cv2.COLOR_BGR2HSV)
        test.imageHSV = cv2.blur(test.imageHSV, (5, 5))
        test.createMask()
        cv2.imshow("Mask", test.mask)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break
cv2.destroyAllWindows()
test.cap.release()
```

Into this:

```
test = detection()
for i in range(100):
    cv2.imshow("Testing", test.calibrateFaceDetection())
    if len(test.faceCoords) != 0:
        test.calibrateSamples()
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break
while True:
    test.temploop()
    cv2.imshow("Image", test.imageBGR)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break
cv2.destroyAllWindows()
test.cap.release()
```

Initially this looks more complicated, but it is a lot more future proof and organised.

When testing I noticed a large decrease in frame rate after the face had been calibrated. This was not a small effect of the image processing as it was a significant change compared to the code before adding the 'temploop' function. The frame rate was around 1 FPS, which is completely unusable for this application.

From tracing the code I could not see any unnecessary functions being executed. The only part of the code that could have a large load is the function that creates the image mask. I edited the code to print how many samples that it had taken of the skin colour and tested it with a picture of a face on a white background. This was the result:

The code:

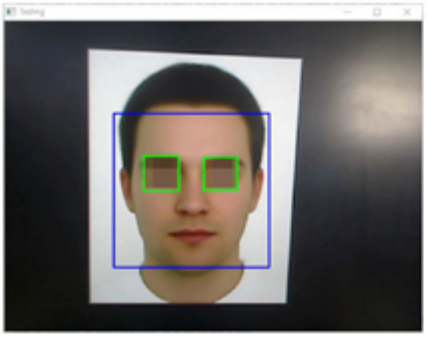
```
def createMask(self):
    self.mask = cv2.inRange(self.imageHSV, self.samples[0][0], self.samples[0][1])
#just to create mask of correct size
    print (len(self.samples))
    for i in self.samples:
        tempMask = cv2.inRange(self.imageHSV, i[0], i[1])
        self.mask = cv2.bitwise_or(self.mask, tempMask)
```

53

Computer Science Coursework – Hand Tracking Application

The third line of code (print (len(self.samples))) is the line that I added for troubleshooting.

The face:



The face and eyes have been selected properly, but the troubleshooting output that printed the number of samples that had been taken printed '1292', which is far too many. It seemed like the program was constantly adding to the list of samples.

I found the error in the code at the end:

```
test = detection()
for i in range(100):
    cv2.imshow("Testing", test.calibrateFaceDetection())
    if len(test.faceCoords) != 0:
        test.calibrateSamples()
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break
while True:
    test.temploop()
    cv2.imshow("Image", test.imageBGR)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break
cv2.destroyAllWindows()
test.cap.release()
```

The first loop that loops 100 times that is used to find the face is adding to the list of samples every time, when it should only do it at the end of the loop. The consequence of this is that later on in the program when the mask is being created each frame the program is creating 1200 different masks and combining them, which was causing the reduced frame rate.

Fixing the code changed the last part to be the following:

```
test = detection()

for i in range(100):
    cv2.imshow("Testing", test.calibrateFaceDetection())
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break
if len(test.faceCoords) != 0:
    test.calibrateSamples()

while True:
    test.temploop()
    cv2.imshow("Image", test.imageBGR)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break
cv2.destroyAllWindows()
test.cap.release()
```

54

This reduced the amount of samples to 12, and therefore restored the frame rate to around 30 FPS. I then removed the troubleshooting print statement.

The code now works fine apart from one issue. When testing I noticed that the code cannot differentiate between my hands and my face, so I need to either program it to ignore the face or change the camera angle so that my face is not in the frame.

To get the program to ignore my face I decided to use the face detection to determine the location of my face in the frame, and then put a black square over that location in the mask image (the image that the program uses to find the hands). This would keep the image preview from looking normal, but would prevent the program from locating my face and thinking that it is a hand (the program only looks for skin coloured objects in the frame, hence why it thinks that a face is a hand).

I have added this code to the start of the findContours function:

```
def findContours(self):

    self.imageGRAY = cv2.cvtColor(self.imageBGR, cv2.COLOR_BGR2GRAY)
    self.faceDetect(self.imageGRAY)
    if len(self.faceCoords) != 0:
        [fx, fy, fw, fh] = self.faceCoords
        cv2.rectangle(self.mask, (fx, fy), (fx+fw, fy+fh), (0, 0, 0), -1)
```

This code creates a grayscale image which is used with the faceDetect function created earlier, which updates the faceCoords variable with the location of the face.

An if statement then checks to see that a face has been found, and if it has a black rectangle is dawn over the face that has the same width and height of the face. The function then continues as normal

This is the mask after the face has been removed. There is still a lot of background objects being picked up, but this can be adjusted later by changing the tolerance values. The rectangle may need to extend lower to cover the neck if the neck gets mistaken for a hand.

An issue with this method is that when the hand passes over the face either the hand will not be detected, or the face recognition will not identify the face and the face will be mistaken for a hand. Using the face detection on every frame can also be very intensive to process. If this becomes an issue later on it can be disabled and instead the user will be told to keep their head out of the camera frame.

With the image processing almost finished I can now start working on a user interface for it.

Stage 1 Review

What has been done

The image processing program has been very nearly completed. The program can successfully take an image from a webcam and mask it to only show areas of skin (excluding the face). It achieves this by using face recognition to find the face in the image, and then eye recognition to find the eyes and then take some samples of the colour of the users face. Then, any pixels in the image that are close to this colour are assumed to be skin and are highlighted in white, and the rest are black. So that the program does not see the face, a black rectangle is drawn over the face.

The result is a mask that shows all the areas in an image apart from the face that are skin, and can be assumed to be a hand. This is that start of the hand recognition.

How it has been tested

Each function that I wrote was tested as I wrote it to make sure that it worked. For the webcam this involved just checking that I could see the webcam image on the screen, and then when I added the face recognition I made the program draw a rectangle around any faces or eyes and tested that the faces were identified correctly by pointing the camera at myself and different pictures of people.

To make sure that it only showed the largest face I found a picture of a large face and put it next to a variety of smaller faces to see which one it detected.

For non-graphical parts of the program I tested it by printing different values to see how the program was behaving. The program was running slowly after I added the part that takes colour samples of the face. To test this I tried to work out what was wrong and determined it could be the amount of samples the program was taking. I added a line of code that printed the amount of samples the program had (which should have been around the scale of 5 to 30) and found that there were over 1000, which was a problem.

For the mask I made the program show it so that I could see if it had identified areas of skin, which led me to use face recognition to remove the face.

How it meets the success criteria and user expectations

The webcam is used to take an image of the user, which is a requirement. This image should then be used to find the hands, which is mostly complete. Face detection is used to calibrate it, which is non technical. There is an overlay showing what the computer 'sees', in this case the boxes around the face and eyes.

Criteria met/being met:

- Webcam feed shows what the computer 'sees'
- A simple, non-technical calibration
- Main window that shows webcam feed

Changes in the design that have resulted from this section

So far, the code is following the pre-planned algorithms. The window design at this stage is temporary, and when the user interface is completed it should look as it does in the design.

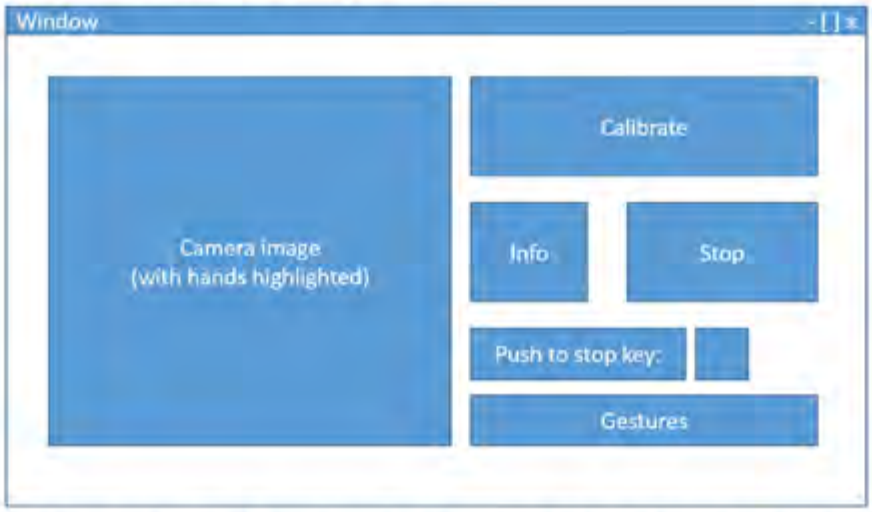
Summary of the whole project as a prototype at this stage

Starting the program causes a window to open with just the webcam feed in it with faces and eyes highlighted (no buttons or user interface). After a certain amount of time the program takes some samples of the face and then shows a mask image (black and white) showing areas that are the same

colour as skin, and there is a box around the face (removing it). It is not in a final or usable state as it is very early in development.

Stage 2, main window user interface

This is the design that I created in the design section:



I will be using Tkinter to handle the user interface. I have made a basic Tkinter class that has buttons and an image that fit's the layout above.

```
import Tkinter as tk

from Tkinter import PhotoImage
class mainClass:
    def __init__(self, master):
        self.master = master
        self.master.title("main")
        photo = PhotoImage(file = 'CalibrationNeeded.gif')
        self.w = tk.Label(master, image = photo)
        self.w.grid(row = 0, column = 0, rowspan = 5)
        self.calibrateButton = tk.Button(master, text = "Calibrate", height = 5,
width = 50)
        self.calibrateButton.grid(row = 0, column = 1, columnspan = 2)
        self.infoButton = tk.Button(master, text = "Info", height = 5, width = 25)
        self.infoButton.grid(row = 1, column = 1)
        self.stopButton = tk.Button(master, text = "Start", height = 5, width =
25)
        self.stopButton.grid(row = 1, column = 2)
        self.stopLabel = tk.Label(master, text = "Push to stop key:",
font=("Helvetica", 16))
        self.stopLabel.grid(row = 2, column = 1)
        self.stopKeyLabel = tk.Label(master, text = "S",  font=("Helvetica", 16))
        self.stopKeyLabel.grid(row = 2, column = 2)
        self.settingsButton = tk.Button(master, text = "Settings", height = 5,
width = 50)
        self.settingsButton.grid(row = 3, column = 1, columnspan = 2)
        self.runningLabel = tk.Label(master, text = "Not Running",
font=("Helvetica", 16))
        self.runningLabel.grid(row = 4, column = 1, columnspan = 2)
def main():
    root = tk.Tk()
    app = mainClass(root)
    root.mainloop()
if __name__ == '__main__':
    main()
```

This code creates this window:

Computer Science Coursework – Hand Tracking Application

It first imports the necessary libraries, so far only Tkinter and PhotoImage for the images. I created a class for the main window called mainClass, and in the initialise function the layout is defined. Each item in the window has two lines (except for the image), one that defines what it is (for example self.infoButton = tk.Button(master, text = "Info", height = 5, width = 25)), the second line states where it will go in the window. I have used the grid layout system which means each item is given a row number and a column number, and in cases where an item takes up multiple rows or columns a rowspan and columnspan. When executed the window looks like the following:

The layout is the same as in the design, except I added some text at the bottom to let the user know the current state of the program (it shows 'Not Running' to start with). The large blank space is supposed to be a place holder image (CalibrationNeeded.gif) that says calibration is needed, which is not working.

This is the image that is meant to fill the blank space:

This is not working; however, I am not going to fix it because the method I am trying to use to display it is for static images like this, and this space will eventually be occupied by a video feed which will need completely different code. If I were to fix this I would only remove it later on, and it shows the size of the window which is all I need it to do at this stage.

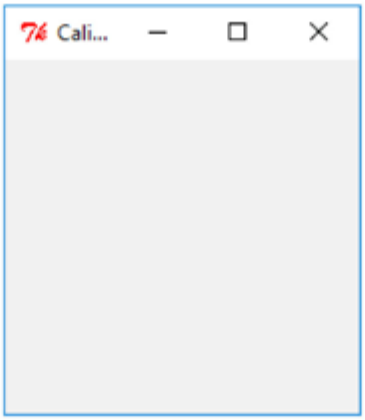
I will now create the pop-up windows needed for the calibration windows and the other buttons. To do this I need a new class for each window, and a function in the main class that will open them.

```
class calibration:

    def __init__(self, master):
        self.master = master
        self.master.title("Calibration")
```

This is the blank class that I've created, and it makes this interface:

59

Computer Science Coursework – Hand Tracking Application

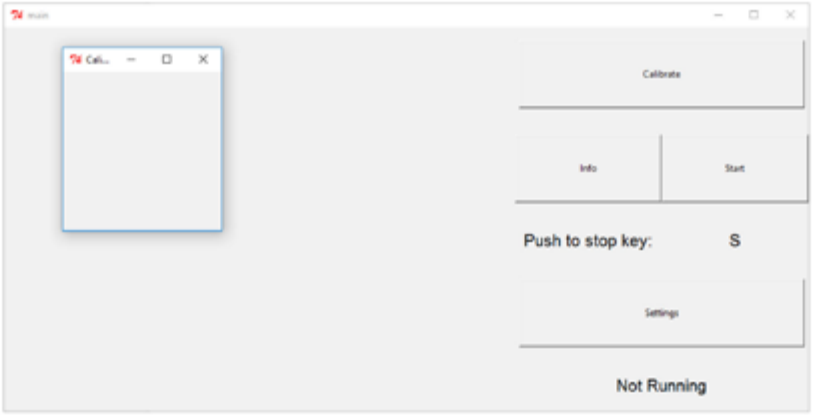


In the mainClass I have added this function named calibrateWindow:

```
def calibrateWindow(self):

    self.newWindow = tk.Toplevel(self.master)
    self.app = calibration(self.newWindow)
```

I have also amended the calibrate button line to link it to this function (added command = self.calibrateWindow):

```
self.calibrateButton = tk.Button(master, text = "Calibrate", command =
self.calibrateWindow, height = 5, width = 50)
```



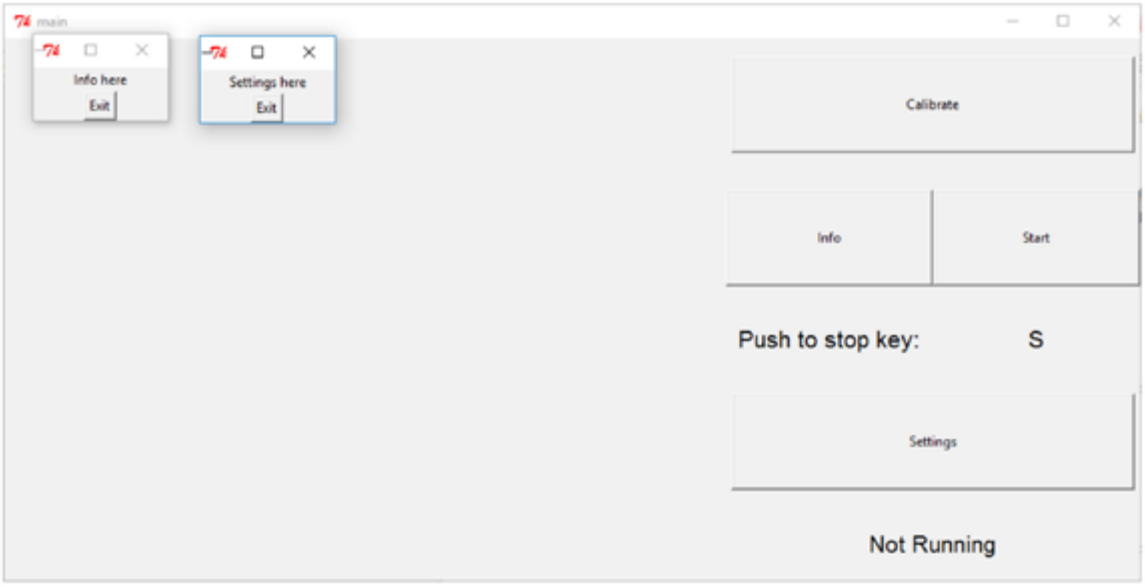
The result is this pop up window in front of the main window. Using this method of creating windows means that I can close the pop up window with the main window staying open, and if I close the main window the pop up window also closes with it.

Before adding content to the calibration window I have added placeholder windows for each of the other two buttons (information and settings). These are the classes for those windows:

Computer Science Coursework – Hand Tracking Application

```
class info():

    def __init__(self, master):
        self.master = master
        self.infoText = tk.Label(master, text = "Info here")
        self.infoText.pack()
        self.exitButton = tk.Button(master, text = "Exit", command =
self.close_windows)
        self.exitButton.pack()

    def close_windows(self):
        self.master.destroy()
class settings():
    def __init__(self, master):
        self.master = master
        self.infoText = tk.Label(master, text = "Settings here")
        self.infoText.pack()
        self.exitButton = tk.Button(master, text = "Exit", command =
self.close_windows)
        self.exitButton.pack()

    def close_windows(self):
        self.master.destroy()
```

They each have text identifying what they will be later on, and they have a button that links to a function that closes them. I have also added a command to their buttons so that they can be opened that links to their respective functions. The mainClass now looks like this:



61

Computer Science Coursework – Hand Tracking Application

```
class mainClass:

    def __init__(self, master):
        self.master = master
        self.master.title("main")
        photo = PhotoImage(file = 'CalibrationNeeded.gif')
        self.w = tk.Label(master, image = photo)
        self.w.grid(row = 0, column = 0, rowspan = 5)
        self.calibrateButton = tk.Button(master, text = "Calibrate", command =
self.calibrateWindow, height = 5, width = 50)
        self.calibrateButton.grid(row = 0, column = 1, columnspan = 2)
        self.infoButton = tk.Button(master, text = "Info", command =
self.infoWindow, height = 5, width = 25)
        self.infoButton.grid(row = 1, column = 1)
        self.stopButton = tk.Button(master, text = "Start", height = 5, width =
25)
        self.stopButton.grid(row = 1, column = 2)
        self.stopLabel = tk.Label(master, text = "Push to stop key:",
font=("Helvetica", 16))
        self.stopLabel.grid(row = 2, column = 1)
        self.stopKeyLabel = tk.Label(master, text = "S",  font=("Helvetica", 16))
        self.stopKeyLabel.grid(row = 2, column = 2)
        self.settingsButton = tk.Button(master, text = "Settings", command =
self.settingsWindow, height = 5, width = 50)
        self.settingsButton.grid(row = 3, column = 1, columnspan = 2)
        self.runningLabel = tk.Label(master, text = "Not Running",
font=("Helvetica", 16))
        self.runningLabel.grid(row = 4, column = 1, columnspan = 2)
    def calibrateWindow(self):
        self.newWindow = tk.Toplevel(self.master)
        self.app = calibration(self.newWindow)
    def infoWindow(self):
        self.newWindow = tk.Toplevel(self.master)
        self.app = info(self.newWindow)
    def settingsWindow(self):
        self.newWindow = tk.Toplevel(self.master)
    self.app = settings(self.newWindow)
```

62

Stage 2 Review

What has been done

Using Tkinter a separate program file has been created for the user interface. The main menu is populated with buttons and text as per the design, but they are not linked to any functions yet. The settings and information menus have been created, but they are empty.

How it has been tested

Each time I added a part of the interface I ran the program to see if it worked, and if it was aligned correctly. The main window was meant to have an image that said 'Calibration needed' but I didn't get it to show in the window. I decided not to fix this as the method I was using will be changed later on, so if I fixed it I would only remove it later on in development.

How it meets the success criteria and user expectations

The main window has a simple design and has room for the webcam feed. It is the start to meeting the following criteria:

- Main window that shows webcam feed
- Simple, lightweight design.
- A large stop button with colour showing if the program is running
- The option to press a button to stop the program
- Text that shows which button this is
- A settings menu in a separate window
- An information menu in a separate window

Changes in the design that have resulted from this section

This section has followed the design so far.

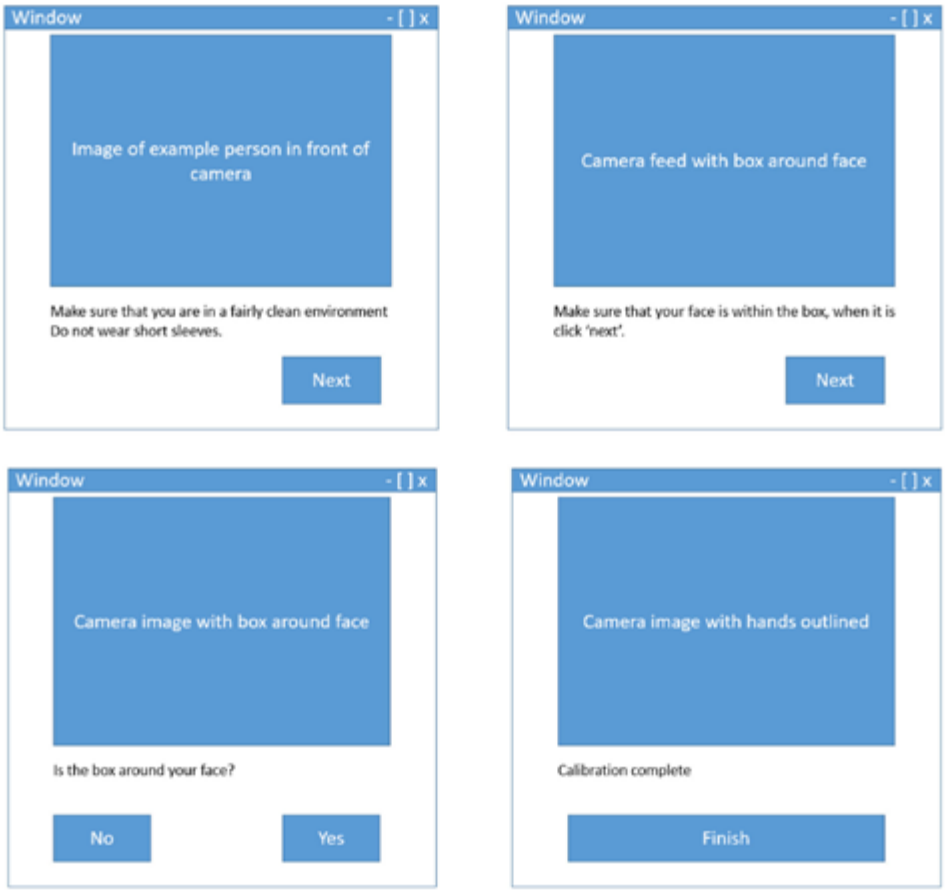
Summary of the whole project as a prototype at this stage

The user interface made here and the program from section 1 are completely separate at the moment. This program opens the user interface, but no buttons are linked to functions. Soon they can be linked and a working prototype can be made.

Computer Science Coursework – Hand Tracking Application

Stage 3, calibration window user interface

The calibration window will have multiple stages which I have designed earlier.



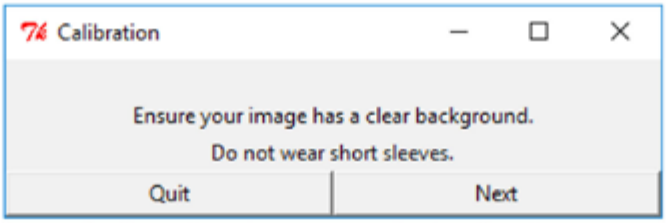
I have added the buttons and text into the calibrate window, but without the image. The quit buttons closes the window using the same method for the information and settings window, and the next button currently does not do anything, but it will link onto the next stage of calibration.

64

This is the new calibration class:

```
class calibration:

    def __init__(self, master):
        self.master = master
        self.master.title("Calibration")
        self.imageLabel = tk.Label(self.master)
        self.imageLabel.grid(row=0, column=0, columnspan=2)
        self.quitButton = tk.Button(self.master, text='Quit', width=25,
command=self.close_windows)
        self.quitButton.grid(row=3, column=0)
        self.nextButton = tk.Button(self.master, text='Next', width=25)
        self.nextButton.grid(row=3, column=1)
        self.instructions = tk.Label(master, text="Ensure your image has a clear
background.")
        self.instructions2 = tk.Label(master, text="Do not wear short sleeves.")
        self.instructions.grid(row=1, column=0, columnspan=2)
        self.instructions2.grid(row=2, column=0, columnspan=2)
    def close_windows(self):
        self.master.destroy()
```

Which creates this window:



To add the camera images I will use the other program that handles the image processing. The program is called MainDevelopment.py, so I will import it to the user interface program and assign the object cameraFeed to the main class in MainDevelopment.py.

I have also added openCV (cv2) and some image libraries (PIL, Image, ImageTk) to the imports. The imports in the user interface program are now:

```
import Tkinter as tk

from Tkinter import PhotoImage
from PIL import Image, ImageTk
import cv2
import MainDevelopment
```

And the code at the end that runs it is now:

```
cameraFeed = MainDevelopment.detection()

def main():
    root = tk.Tk()
    app = mainClass(root)
    root.mainloop()
if __name__ == '__main__':
    main()
```

When running this program two windows opened, the user interface and the MainDevelopment code. This is because at the end of the MainDevelopment code there was still the code that tested each function:

Computer Science Coursework – Hand Tracking Application

```
test = detection()

for i in range(100):
    cv2.imshow("Testing", test.calibrateFaceDetection())
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break
if len(test.faceCoords) != 0:
    test.calibrateSamples()
while True:
    test.temploop()
    cv2.imshow("Image", test.imageBGR)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break
cv2.destroyAllWindows()
test.cap.release()
```

I have commented this code out so that it does not run (I have not deleted it because it can still be used later on if I need to troubleshoot, it allows me to test the algorithm without using the user interface).

I've added a function that uses the MainDevelopment program and takes an image from the webcam and displays it. There is a loop at the end that creates the video feed. This is the function which is called showFrame:

```
def showFrame(self):

    _, image = cameraFeed.cap.read()#Captures an image
    cv2image = cv2.cvtColor(image, cv2.COLOR_BGR2RGBA)#Converts the format of the
image
    img = Image.fromarray(cv2image)#Converts it again to a format that tkinter
accepts
    imgtk = ImageTk.PhotoImage(image=img)
    self.imageLabel.imgtk = imgtk
    self.imageLabel.configure(image=imgtk)#Sets the image
    self.imageLabel.after(10, self.showFrame)#Loops
```

This structure can be used for the rest of the user interface as it needs an image to be returned from the MainDevelopment program (in this case the variable image), and each image needed for the calibration uses this method.

The webcam image now shows in the calibration window:

66

When next is clicked the frame should change to have new instructions and a different video feed (the one that highlights where the face and eyes are in the frame). The function that will do this destroys everything in the frame and rebuilds it. This function is called stage2:

```
def stage2(self):

    self.instructions.destroy()
    self.instructions2.destroy()
    self.nextButton.destroy()
    self.quitButton.destroy()
    self.imageLabel.destroy()
    self.imageLabel = tk.Label(self.master)
    self.imageLabel.grid(row=0, column=0, columnspan=2)
    self.showFace()
    self.instructions = tk.Label(self.master, text = "Ensure your face is within
the blue box,")
    self.instructions2 = tk.Label(self.master, text = "When it is, click 'Next'")
    self.instructions.grid(row = 1, column = 0, columnspan = 2)
    self.instructions2.grid(row = 2, column = 0, columnspan = 2)
    self.quitButton = tk.Button(self.master, text = 'Quit', width = 25, command =
self.close_windows)
    self.quitButton.grid(row = 3, column = 0)
    self.nextButton = tk.Button(self.master, text = 'Next', width = 25)
    self.nextButton.grid(row = 3, column = 1)
```

It links to the showFace function that currently doesn't exist, which will be show the video feed with the face highlighted.

This is the showFace function:

```
def showFace(self):

    image = cameraFeed.calibrateFaceDetection()
    cv2image = cv2.cvtColor(image, cv2.COLOR_BGR2RGBA)
    img = Image.fromarray(cv2image)
    imgtk = ImageTk.PhotoImage(image=img)
    self.imageLabel.imgtk = imgtk
    self.imageLabel.configure(image=imgtk)
    self.imageLabel.after(10, self.showFace)
```

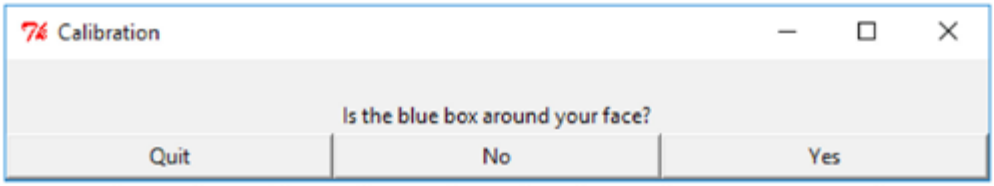
It is almost identical to the showFrame function but it calls a different part of the MainDevelopment file.

With the next button linked to the stage2 function and testing it with a sample image shows that it works:

I then created stage3 using the same method:

```
def stage3(self):

    self.instructions.destroy()
    self.instructions2.destroy()
    self.nextButton.destroy()
    self.quitButton.destroy()
    self.imageLabel.destroy()
    self.imageLabel = tk.Label(self.master)
    self.imageLabel.grid(row=0, column=0, columnspan=3)
    self.instructions = tk.Label(self.master, text="Is the blue box around your
face?")
    self.instructions.grid(row=1, column=0, columnspan=3)
    self.yesButton = tk.Button(self.master, text='Yes', width=25)
    self.noButton = tk.Button(self.master, text='No', width=25)
    self.yesButton.grid(row=2, column=2)
    self.noButton.grid(row=2, column=1)
    self.quitButton = tk.Button(self.master, text='Quit', width=25,
command=self.close_windows)
    self.quitButton.grid(row=2, column=0)
```

This creates the following window:



The image in this window will be a still image from when the 'next' button was pressed in stage2. Because of this I will not need to have a video feed, but only a still image. I will not need to use a function or a loop for this, but can just run this code once:

```
image = cameraFeed.imageBGR

cv2image = cv2.cvtColor(image, cv2.COLOR_BGR2RGBA)
img = Image.fromarray(cv2image)
imgtk = ImageTk.PhotoImage(image=img)
self.imageLabel.imgtk = imgtk
self.imageLabel.configure(image=imgtk)
```

This code is the same as the image loops earlier. Testing this resulted in the video stopping when the 'next' button was pressed, which means that it is working. The user then has a choice of whether they would like to try again with taking a picture or accept the one they have taken. If they choose to go back to stage2 to try again, the yes and no button will not be destroyed, resulting in a glitch where the yes and no buttons will not disappear. To resolve this I will create a function called gotoStage2 which will just destroy the two buttons, then go to stage two.

```
def gotostage2(self):

    self.yesButton.destroy()
    self.noButton.destroy()
    self.stage2()
```

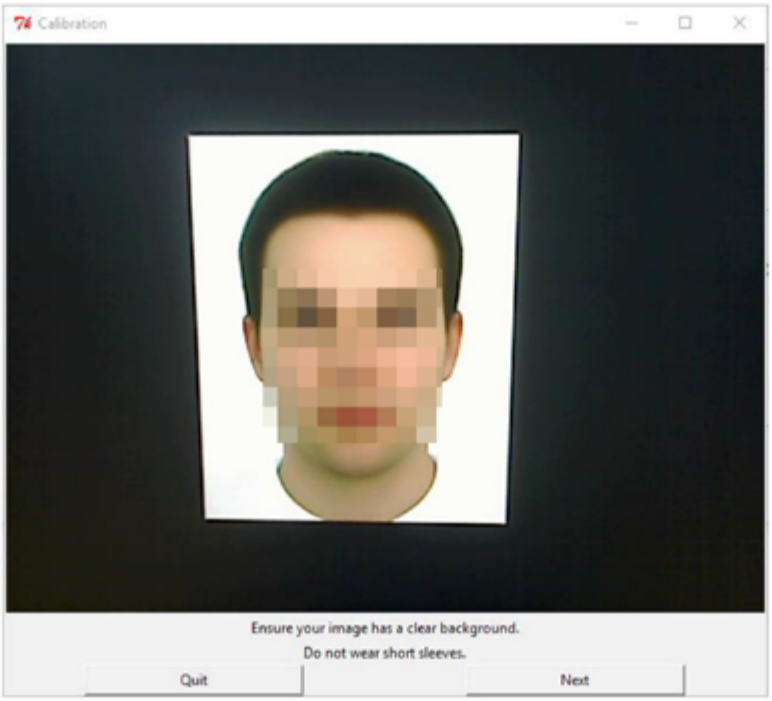
The other button will continue to the next stage of calibration. The next stage is just a screen saying that the calibration is complete with a button to close the pop up window. This stage will call the calibrateSamples function from the MainDevelopment file which will generate the samples.

Computer Science Coursework – Hand Tracking Application

To test this calibration system I ran through each step to ensure that it worked:

From the main window I click calibrate to start the calibration process.

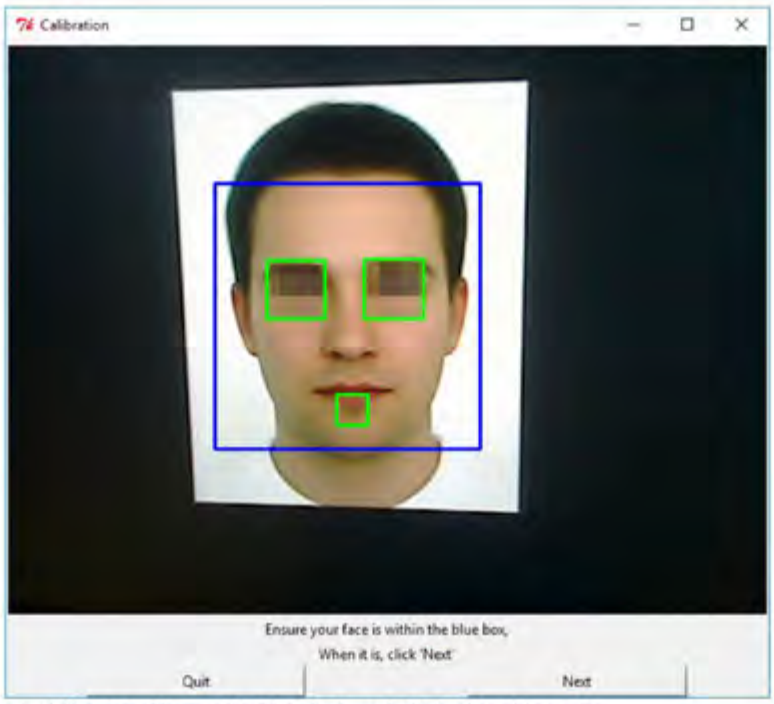
Then, with the camera pointed at an image of a generic face, I have the option of quitting the calibration or continuing, so I click next.



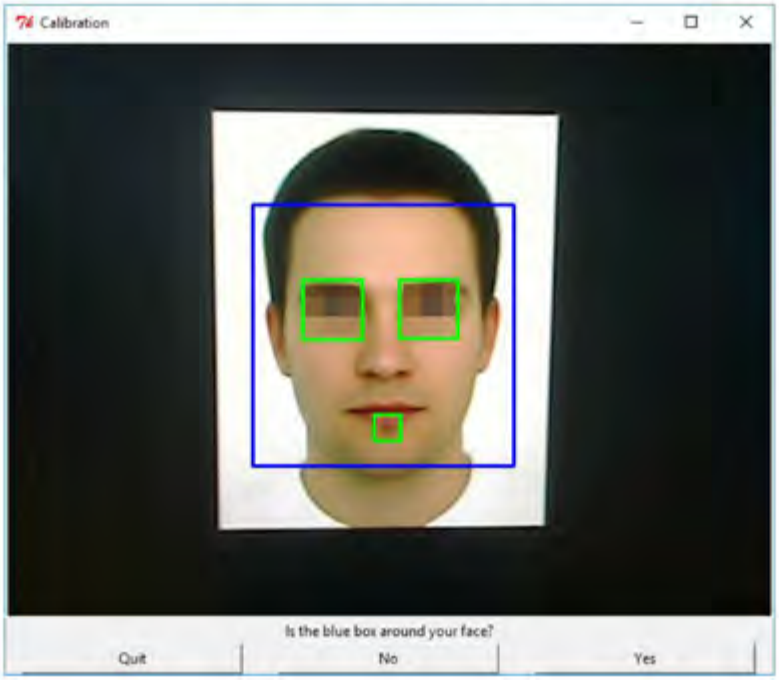
I then need to make sure the boxes are in the correct place with the blue box around the face and the green boxes around both eyes. This highlights a problem with the program so far, the eye detection can sometimes pick up facial features that are not the eyes. I clicked next, trying to not have the green box around the mouth.

69

Computer Science Coursework – Hand Tracking Application



I took a picture but the mouth was highlighted, so I clicked 'no' to retry.

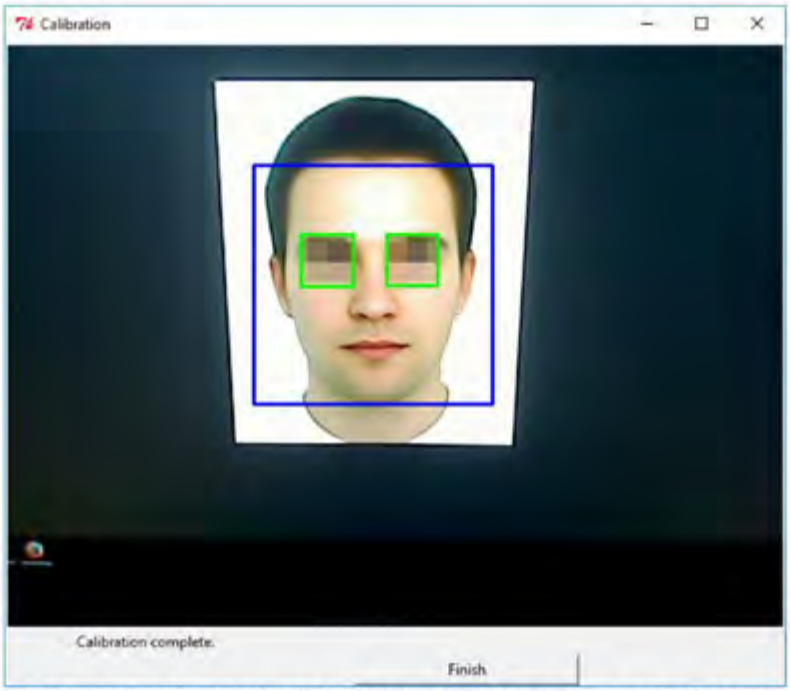


The second time I got it right, so I clicked 'yes'.

70

Computer Science Coursework – Hand Tracking Application

Then the calibration was complete.



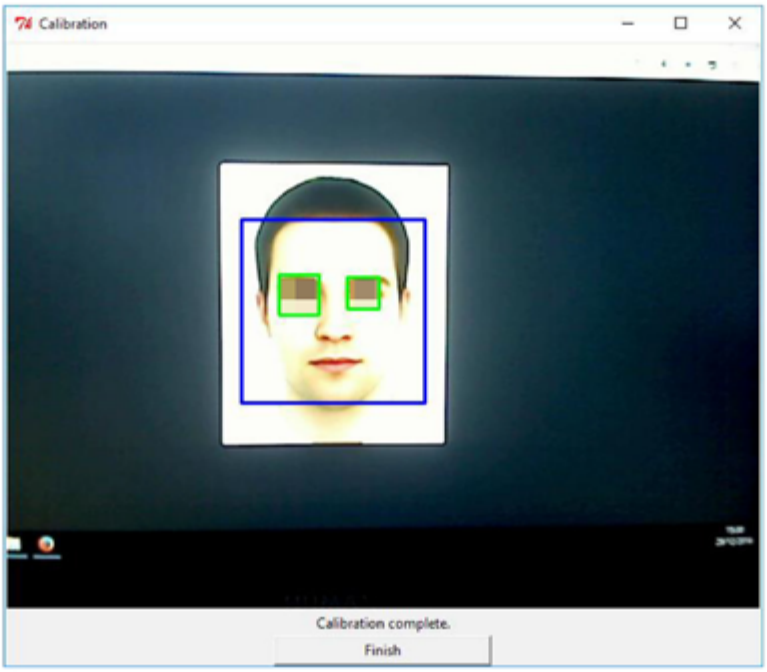
The calibration is then complete. I noticed that the text and the button were off centre, so I fixed this by changing their location:

```
self.instructions = tk.Label(self.master, text = "Calibration complete.")

self.instructions.grid(row = 1, column = 0, columnspan = 3)
self.finishButton = tk.Button(self.master, text = "Finish", width = 25, command =
self.close_windows)
```

71

Computer Science Coursework – Hand Tracking Application

```
self.finishButton.grid(row = 2, column = 0, columnspan = 3)
```



This is the result.

If the calibration is done incorrectly it may result in the program crashing, for example if there is no face in the frame, or if there are too many eyes detected. This is something that I will fix later on after I have added the video feed to the main window.

This is the error that results from clicking 'yes' in the calibration when there is not a face:

```
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\David Robson\Anaconda2\lib\lib-tk\Tkinter.py", line 1537, in __call__
    return self.func(*args)
  File "C:/Users/David Robson/Documents/Work/Computing/OpenCV stuff/Programs/Coursework/GUI Development.py", line 151, in stage4
    cameraFeed.calibrateSamples()
  File "C:\Users\David Robson\Documents\Work\Computing\OpenCV stuff\Programs\Coursework\MainDevelopment.py", line 56, in calibrateSamples
    [x, y, w, h] = self.faceCoords
ValueError: need more than 0 values to unpack
```

For the main window I will need two options for what to display, one being the video feed after it can been calibrated and another being an image informing the user that calibration is required. I already have the 'calibration needed' image from earlier which I will use.

72

The program will need a way on knowing if calibration has been complete. For this I created a global variable called 'calibrated'. It has to be a global variable so that the different classes (mainClass and calibration) can both use it. The variable is declared at the start of the program:

```
calibrated = 0
```

And anywhere where the variable needs to be changed this line will need to be added:

```
global calibrated
```

The variable is 0 by default, meaning that the program is not calibrated. When the calibration is complete the variable is set to 1, meaning that it has been calibrated.

To show the 'calibration needed' image or the camera image I have created a function called imageLoop that will change the image accordingly. The image is inside of a label called imageLabel:

```
self.img = ImageTk.PhotoImage(Image.open("CalibrationNeeded.png"))#The calibration
image

self.imageLabel = tk.Label(master, image = self.img)#Sets the frame to show the
calibration image by default
self.imageLabel.grid(row = 0, column = 0, rowspan = 5)#Location of the image
self.imageLoop()
```

By default, it sets the image to the 'calibration needed' image. It then goes to the imageLoop function:

```
def imageLoop(self):

    global calibrated
    if calibrated:#checks the calibrated variable
        cameraFeed.temploop()#If it has been calibrated, it shows the camera image
        image = cameraFeed.imageBGR
        cv2image = cv2.cvtColor(image, cv2.COLOR_BGR2RGBA)
        img = Image.fromarray(image)
        imgtk = ImageTk.PhotoImage(image=img)
        self.imageLabel.imgtk = imgtk
        self.imageLabel.configure(image=imgtk)
    else:
        self.imageLabel.configure(image=self.img)#Else, it shows the 'calibration
needed' image
    self.imageLabel.after(10, self.imageLoop)#Loops
```

It first checks if the program has been calibrated. If it has it uses the image loop created earlier for the calibration windows.

If it has not been calibrated then it sets the image to the calibration needed image again.

Stage 3 Review

What has been done

The calibration menu user interface has been created and the calibration from stage 1 has been moved into these windows.

How it has been tested

After each change to the code the program was run to make sure that it worked. After the section was finished I did a run through of the calibration process and in this I pressed each button to make sure that it worked properly. Testing revealed that if 'yes' is clicked when there is no face in the frame then the program crashes, which will be addressed later when the user interface is complete.

How it meets the success criteria and user expectations

It is a simple calibration that is not technical because all the user has to do is to sit in front of the camera and click the 'next' buttons accordingly.

It meets the following criteria

A simple, non-technical calibration

Changes in the design that have resulted from this section

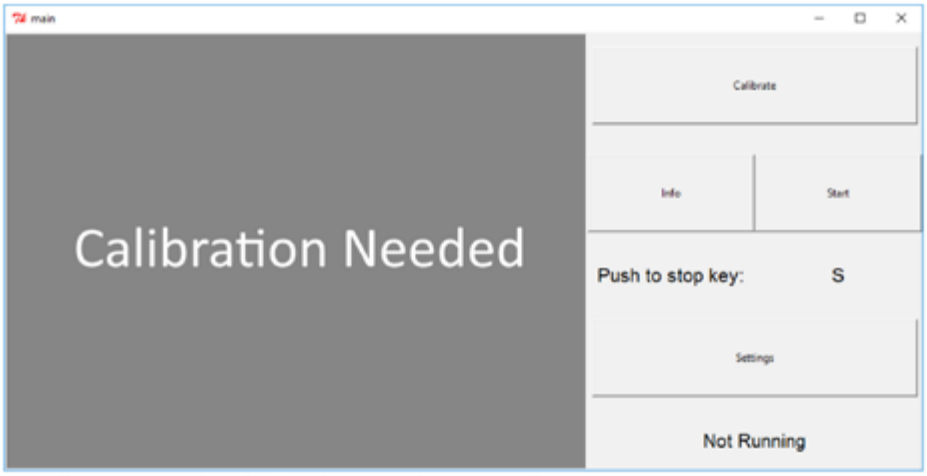
A 'quit' button has been added to each window in the calibration.

The 'Calibration needed' image has been added to the main window for when the program has not been calibrated.

Summary of the whole project as a prototype at this stage

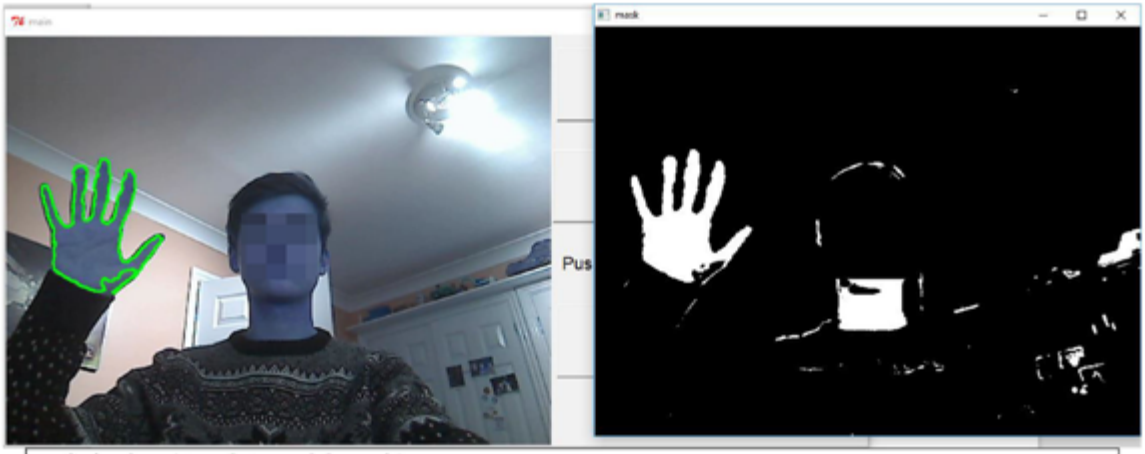
The main window shows that the program has not been calibrated. It can be calibrated by clicking on the calibration button which brings up the calibration windows. After the calibration instructions are followed then the program is calibrated, but at this stage it does not do anything with this information.

Stage 4, refining the main window and calibration



This is how the main window looks when it is first started.

The left window is showing the video from the camera with the hand highlighted, but there is also a pop up window showing the mask image. This is from the other program from when I was troubleshooting. There is a line of code that shows the mask, I will comment it out because I may use it later for changing the tolerance levels of the samples (to reduce the white spots in the background but still have the hand highlighted).




```
cv2.imshow("mask", self.mask)
```

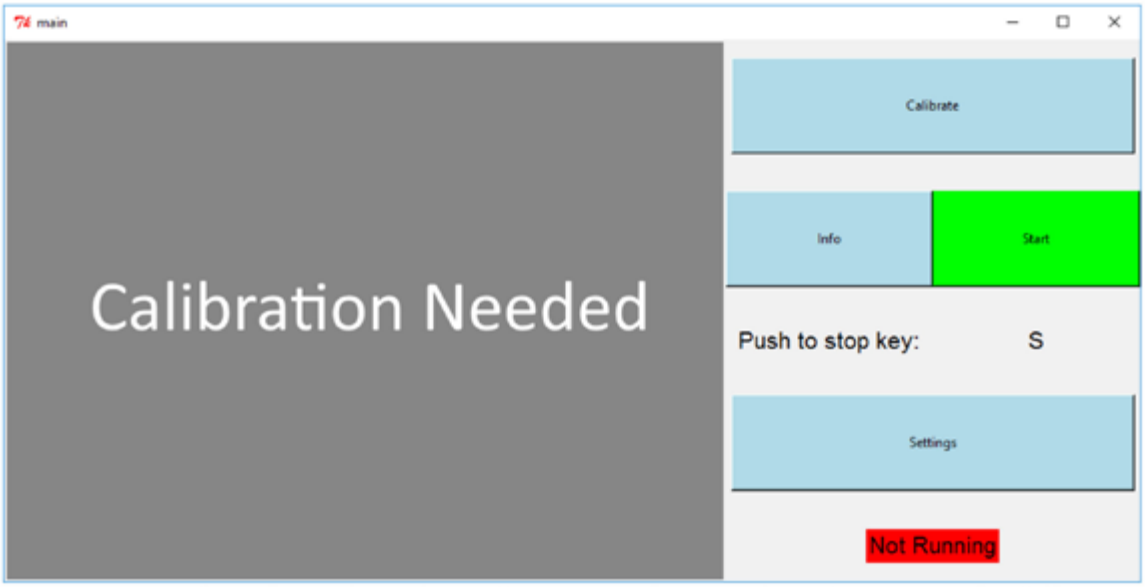
That is the line that I commented out.

The start button needs to be a green colour so that it is obvious that that is the button to press to start the mouse movements. I also decided to colour code the 'Not Running' text at the bottom of the window. While I was at it I also changed the other buttons colours to be a light blue so that they stand out more.

Computer Science Coursework – Hand Tracking Application

This is the revised code with the colours included:

```
self.calibrateButton = tk.Button(master, text = "Calibrate", command =
self.calibrateWindow, height = 5, width = 50, bg = 'LIGHT BLUE')

self.calibrateButton.grid(row = 0, column = 1, columnspan = 2)
self.infoButton = tk.Button(master, text = "Info", command = self.infoWindow,
height = 5, width = 25, bg = 'LIGHT BLUE')
self.infoButton.grid(row = 1, column = 1)
self.stopButton = tk.Button(master, text = "Start", height = 5, width = 25, bg =
'GREEN')
self.stopButton.grid(row = 1, column = 2)
self.stopLabel = tk.Label(master, text = "Push to stop key:",  font=("Helvetica",
16))
self.stopLabel.grid(row = 2, column = 1)
self.stopKeyLabel = tk.Label(master, text = "S",  font=("Helvetica", 16))
self.stopKeyLabel.grid(row = 2, column = 2)
self.settingsButton = tk.Button(master, text = "Settings", command =
self.settingsWindow, height = 5, width = 50, bg = 'LIGHT BLUE')
self.settingsButton.grid(row = 3, column = 1, columnspan = 2)
self.runningLabel = tk.Label(master, text = "Not Running", font=("Helvetica", 16),
bg = 'RED')
self.runningLabel.grid(row = 4, column = 1, columnspan = 2)
```



To make the colours change depending on if the program is running I added another local variable called self.running.

```
self.running = 0
```

I also made a function that changes that variables state when the start button is pressed. It also changes the colour of the button and its text, and does the same for the text at the bottom of the window.

76

```
def startStop(self):

    if calibrated:
        self.running = not(self.running)
        if self.running:
            self.stopButton.configure(bg = 'RED', text = "Stop")
            self.runningLabel.configure(bg = 'GREEN', text = "Running")
        else:
            self.stopButton.configure(bg = 'GREEN', text = "Start")
            self.runningLabel.configure(bg='RED', text="Not Running")
```

It only does this if the program is calibrated to prevent an error (the mouse can only move automatically when the program knows where the hand is, so it has to be calibrated).

To stop the program from thinking the neck is a hand I have extended the rectangle that covers the face downwards to cover the neck:

This is the old code:

```
cv2.rectangle(self.mask, (fx, fy), (fx+fw, fy+fh), (0, 0, 0), -1)
```

And this is the revised code:

```
cv2.rectangle(self.mask, (fx, fy), (fx+fw, fy+int(fh*1.5)), (0, 0, 0), -1)
```

Old

New

'fh' is the height of the rectangle, and I multiplied it by 1.5 to make it longer. This is to prevent this issue of it detecting the neck and thinking it is a hand, like in this image:

The face is clearly blocked out by the rectangle, but the neck isn't. This is now fixed.

After using this it is clear that the tolerance levels need to be able to change via the software (at the moment I am changing them in the program code). I thought that to change them there could be a slider inside the settings menu, but I think that it would be best to place it at the end of the calibration window. Instead of seeing the webcam image you would see the mask image (like in the image above) and would have a slider to change the tolerance.

To start this I changed the final stage of calibration to have the mask image inside of it. I created another function for the video feed called showMask:

Computer Science Coursework – Hand Tracking Application

```
def showMask(self):

    cameraFeed.temploop()
    image = cameraFeed.mask
    img = Image.fromarray(image)
    imgtk = ImageTk.PhotoImage(image=img)
    self.imageLabel.imgtk = imgtk
    self.imageLabel.configure(image=imgtk)
    self.imageLabel.after(10, self.showMask)
```

This is the same as the other functions like showFace, except it calls the temploop function so that the mask is generated. I have not included the line that converts the colour of the image as it is black and white. This is the result:

This is not what I wanted as it is showing the contours (the edges) of the shapes rather than the mask. This is because temploop is called, and part of temploop includes finding the contours, so I have adapted the showMask function to not call temploop, but to instead do everything temploop does apart from finding the contours. This will help because finding the contours changes the mask to what the image shows above.

For reference, this is the temploop function in the MainDevelopment file:

```
def temploop(self):

    _, self.imageBGR = self.cap.read()
    self.imageHSV = cv2.cvtColor(self.imageBGR, cv2.COLOR_BGR2HSV)
    self.imageHSV = cv2.blur(self.imageHSV, (5, 5))
    self.createMask()
    self.findContours()
```

This is the revised showMask function in the GUI file (notice that self has been replaced with cameraFeed):

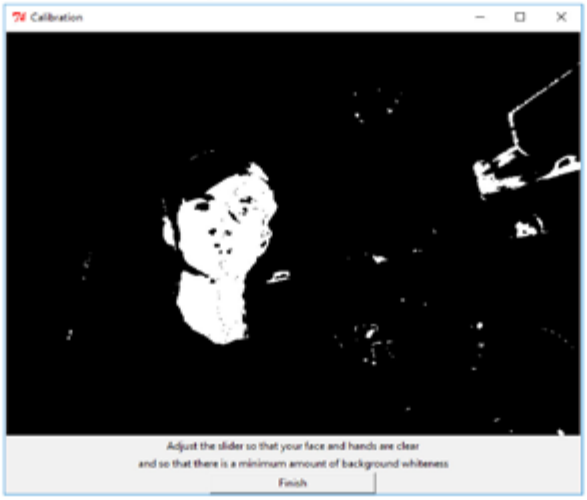
78

Computer Science Coursework – Hand Tracking Application

```
def showMask(self):

    _, cameraFeed.imageBGR = cameraFeed.cap.read()
    cameraFeed.imageHSV = cv2.cvtColor(cameraFeed.imageBGR, cv2.COLOR_BGR2HSV)
    cameraFeed.imageHSV = cv2.blur(cameraFeed.imageHSV, (5, 5))
    cameraFeed.createMask()

    image = cameraFeed.mask
    img = Image.fromarray(image)
    imgtk = ImageTk.PhotoImage(image=img)
    self.imageLabel.imgtk = imgtk
    self.imageLabel.configure(image=imgtk)
    self.imageLabel.after(10, self.showMask)
```

This is the result; the window is showing the correct mask. The text and buttons now need to be changed.

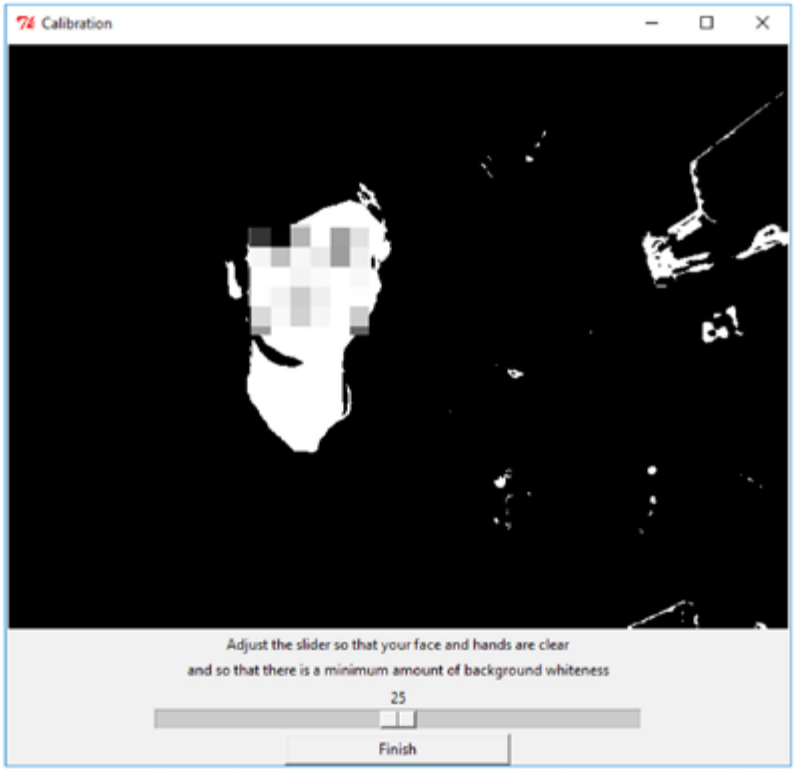
This is the changed window (the text may change later on):



79

Computer Science Coursework – Hand Tracking Application

Then I was able to add a slider that will be used to change the tolerance:



The finish button needs to change the value of the 'calibrated' variable as well as closing the window, so I made a function that does this called 'finish'. This is the new code for the final stage of calibration:

```
def stage4(self):

    self.instructions.destroy()
    self.yesButton.destroy()
    self.noButton.destroy()
    self.quitButton.destroy()
    self.imageLabel.destroy()
    self.imageLabel = tk.Label(self.master)
    self.imageLabel.grid(row=0, column=0, columnspan=3)
    cameraFeed.calibrateSamples()
    self.showMask()
    self.instructions = tk.Label(self.master, text = "Adjust the slider so that
your face and hands are clear")
    self.instructions2 = tk.Label(self.master, text="and so that there is a minimum
amount of background whiteness")
    self.instructions.grid(row = 1, column = 0, columnspan = 3)
    self.instructions2.grid(row=2, column=0, columnspan=3)
    self.toleranceSlider = tk.Scale(self.master, from_ = 0, to = 50, orient =
tk.HORIZONTAL, length = 400)
    self.toleranceSlider.grid(row = 3, column = 0, columnspan = 3)
    self.finishButton = tk.Button(self.master, text = "Finish", width = 25, command
= self.finish)
    self.finishButton.grid(row = 4, column = 0, columnspan = 3)
def finish(self):
    global calibrated
    calibrated = 1
    self.close_windows()
```

80

There needs to be a way of changing the tolerance variable, then to change the samples upper and lower bounds. For this I wrote a function in the MainDevelopment file called 'changeTolerance':

```
def changeTolerance(self, value):

    self.tolerance = [value, value, value]
    self.samples = []
    for i in self.rawSamples:
        upper, lower = [], []
        upper.append(i[0] + self.tolerance[0])
        upper.append(i[1] + self.tolerance[1])
        upper.append(i[2] + self.tolerance[2])
        lower.append(i[0] - self.tolerance[0])
        lower.append(i[1] - self.tolerance[1])
        lower.append(i[2] - self.tolerance[2])
        for y in range(len(upper)):  # so that 0<upper<255
            if upper[y] < 0:
                upper[y] = 0
            if upper[y] > 255:
                upper[y] = 255
        for y in range(len(lower)):  # so that 0<lower<255
            if lower[y] < 0:
                lower[y] = 0
            if lower[y] > 255:
                lower[y] = 255
        lower = np.array(lower, dtype=np.uint8)
        upper = np.array(upper, dtype=np.uint8)
        self.samples.append([lower, upper])
```

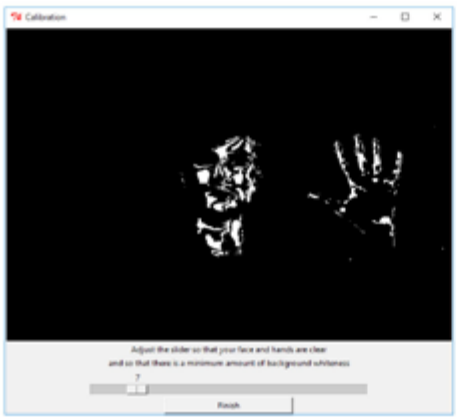
It first sets the tolerance variable to the value provided by the slider, then resets the samples (the samples contain the upper and lower bounds, the rawSamples contain the original colour data).

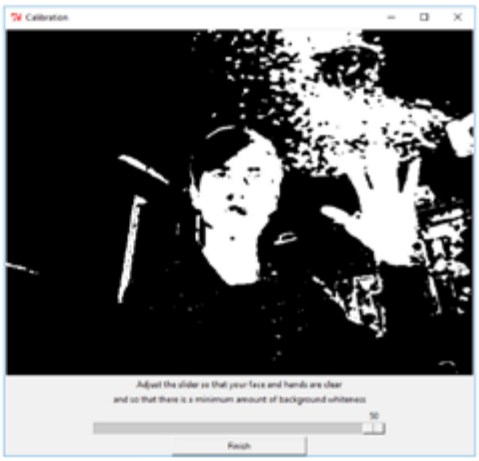
A for loop then loops through each sample in rawSamples and creates the upper and lower bounds. The two for loops following that have been copied from earlier on in the program (they prevent negative colour values, or values that are too large). They are then converted to the correct data type and added to the new list of samples.

In the user interface file the showMask function has been changed to include this line first:

```
cameraFeed.changeTolerance(self.toleranceSlider.get())
```

It calls the changeTolerance function, giving it the value from the slider. The result of this is that the tolerance can be changed in the calibration screen:





These images show the difference that the tolerance values make.

The aim of the program is to be able to move the mouse with your hands, so a function that turns the hand movements into mouse movements is needed.

The variable self.handCoords has three items within it, the x and y coordinates of the hands, and their size (the area of a box that is drawn around them). To move the mouse this command can be used:

```
win32api.SetCursorPos((x, y))
```

so long as win32api has been imported (win32api is a windows library for python). The function moveMouse converts the hand coordinates into a location on the screen then moves the mouse:

```
def moveMouse(self):

    x = self.handCoords[0] / 640.0
    y = self.handCoords[1] / 480.0
    x = 1920 - int(x*1920)
    y = int(y*1200)
    win32api.SetCursorPos((x, y))
```

The hand coordinates are relative to the 640 by 480 camera frame, so they need to be converted to fit onto a larger screen (in my case a 1920 by 1200 screen). The x value is then inverted (the 1900 – part) because otherwise the x axis movement would be backwards (this is because the camera is facing away from the computer screen). They are converted to integers and then the mouse is moved to their location.

In the imageLoop function (the function that puts the video feed onto the main window when it has been calibrated) I added an if statement that checks if the movement is running, and if it is it calls the move mouse function:

```
if self.running:

    cameraFeed.moveMouse()
```

Now when the start button is pressed the mouse movement starts, and can be stopped by the same button used to start it. To stop it there is meant to be the stop button, as well as a key that can be pressed.

After doing some research I found that this can be achieved by using the f-keys (F1, F2, F3 etc.) as inputs by using the following code:

```
import win32api

import win32con
while True:
    if win32api.GetAsyncKeyState(win32con.VK_F2) != 0:
        print("Pressed")
    else:
        print("Not pressed")
```

It imports the necessary libraries then has an infinite loop. In this loop it checks if the F2 key has been pressed (when it has the value of win32api.GetAsyncKeyState(win32apicon.VK_F2) is negative, and not 0). If it is pressed, it prints a message.

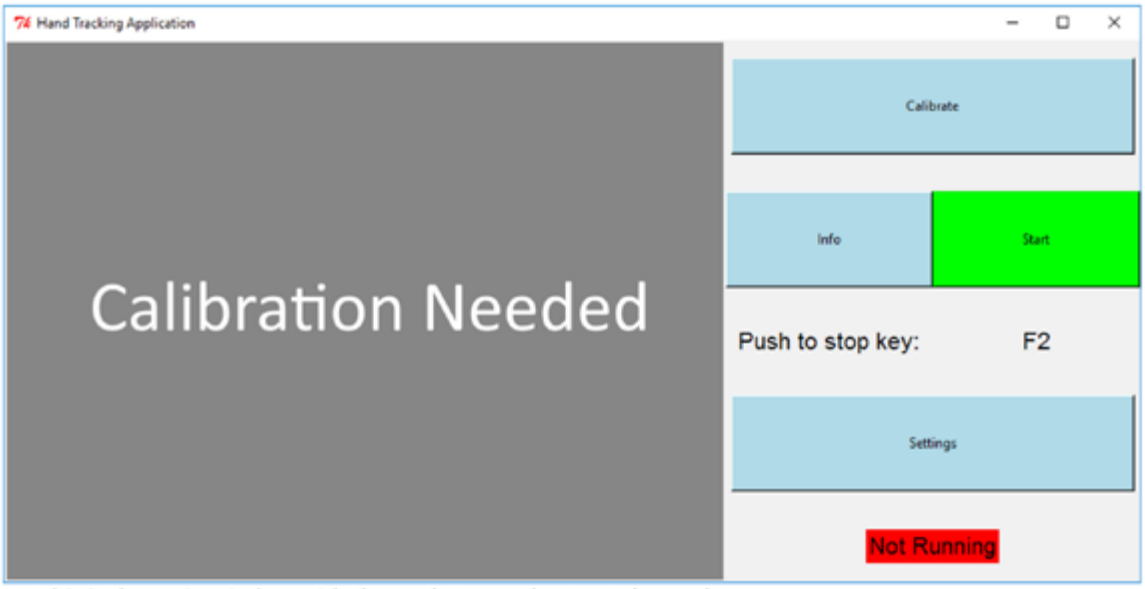
Using this I have added the following line to the imageLoop function:

Computer Science Coursework – Hand Tracking Application

```
if self.running:

    cameraFeed.moveMouse()
    if win32api.GetAsyncKeyState(win32con.VK_F2) != 0:
        self.startStop()
```

So now the whole imageLoop function is:

```
def imageLoop(self):

    global calibrated
    if calibrated:#checks the calibrated variable
        cameraFeed.temploop()#If it has been calibrated, it shows the camera image
        image = cameraFeed.imageBGR
        cv2image = cv2.cvtColor(image, cv2.COLOR_BGR2RGBA)
        img = Image.fromarray(cv2image)
        imgtk = ImageTk.PhotoImage(image=img)
        self.imageLabel.imgtk = imgtk
        self.imageLabel.configure(image=imgtk)
        if self.running:
            cameraFeed.moveMouse()
            if win32api.GetAsyncKeyState(win32con.VK_F2) != 0:
                self.startStop()
    else:
        self.imageLabel.configure(image=self.img)#Else, it shows the 'calibration
needed' image
 self.imageLabel.after(10, self.imageLoop)#Loops
```



This is the main window with the push to stop key text changed.

If the mouse movement is running then it checks to see if F2 has been pressed, if it has it stops the mouse movement by calling the stop function 'startStop', and then stops checking. This also means that the stop key cannot be any key as previously thought, it will have to be an F-key. I have changed the text on the main window to reflect this. I have also changed the title of the window to 'Hand Tracking Application' until the program gets a proper name.

Stage 4 Review

What has been done:

The main window has been fixed so that it now shows the 'Calibration needed' image, and it shows the webcam feed when the program has been calibrated. The calibration has been changed to include a new phase where the user can change the tolerance that the masks use to eliminate background noise.

The main window now has colours to show if the program is running.

The mouse movement now works, so you can use your hands to move the mouse.

How it has been tested:

When each item that I added had been completed I tried to use the program to see if there were any errors, or to see if it was working properly. I made the mask image pop up in a different window to check that it was working as intended despite that the user will not see this in the finished version. The mask was used to make sure the tolerance values were correct (to eliminate background noise, like the white spots)

How it meets the success criteria and user expectations:

The main menu is now clearer as to which button to press to stop the program, and if the program is running. The mouse can now be moved with the users hands.

These are the criteria that this section meets:

- Webcam feed shows what the computer 'sees'
- A simple, non-technical calibration
- A large stop button with colour showing if the program is running
- Moving the hand causes the mouse to move

Changes in the design that have resulted from this section:

The calibration section now has an extra stage to change the tolerance value. This was originally going to be pre-set, but depending on the lighting of the room you are in this value needs to be adjusted, so I let the user set this in the calibration.

Summary of the whole project as a prototype at this stage:

The main menu now shows the webcam feed after it has been calibrated, and when it has not been calibrated it shows the 'Calibration needed' image. The calibration section works and the menu has colours that change depending on weather the hand tracking is running. The hand tracking works (hands can be used to move the mouse once the program is calibrated), but the sensitivity cannot be adjusted.

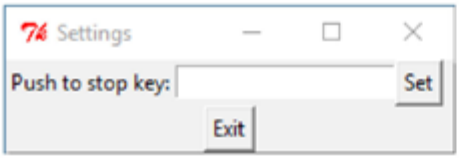
Stage 5, settings menu

The F-key that will be used could be changed in the settings menu in case it is already being used as a key for another program. The settings menu could also have the option to not have face detection to remove the face as this takes up a lot of computing power and removing it would increase performance.

This is the changed settings code that includes the option to change the stop button:

```
class settings():

    def __init__(self, master):
        self.master = master
        self.master.title("Settings")

        self.topKeyText = tk.Label(self.master, text="Push to stop key:")
        self.topKeyText.grid(row = 0, column = 0)

        self.topKeyText2 = tk.Label(self.master, text="")
        self.topKeyText2.grid(row = 0, column = 3)

        self.stopKeyVar = tk.StringVar()
        self.stopKeyEntry = tk.Entry(self.master, textvariable = self.stopKeyVar)
        self.stopKeyEntry.grid(row = 0, column = 1)

        self.stopKeyButton = tk.Button(self.master, text = "Set")
        self.stopKeyButton.grid(row = 0, column = 2)
        self.exitButton = tk.Button(master, text="Exit",
command=self.close_windows)
        self.exitButton.grid(row = 2, column = 0, columnspan = 4)
    def close_windows(self):
        self.master.destroy()
```

This is how it looks:



The text that is entered into the box is stored in the variable stopKeyVar.

```
self.stopKeyVar = tk.StringVar()

self.stopKeyEntry = tk.Entry(self.master, textvariable = self.stopKeyVar)
self.stopKeyEntry.grid(row = 0, column = 1)
self.stopKeyButton = tk.Button(self.master, text = "Set", command =
self.stopKeyGet)
self.stopKeyButton.grid(row = 0, column = 2)
```

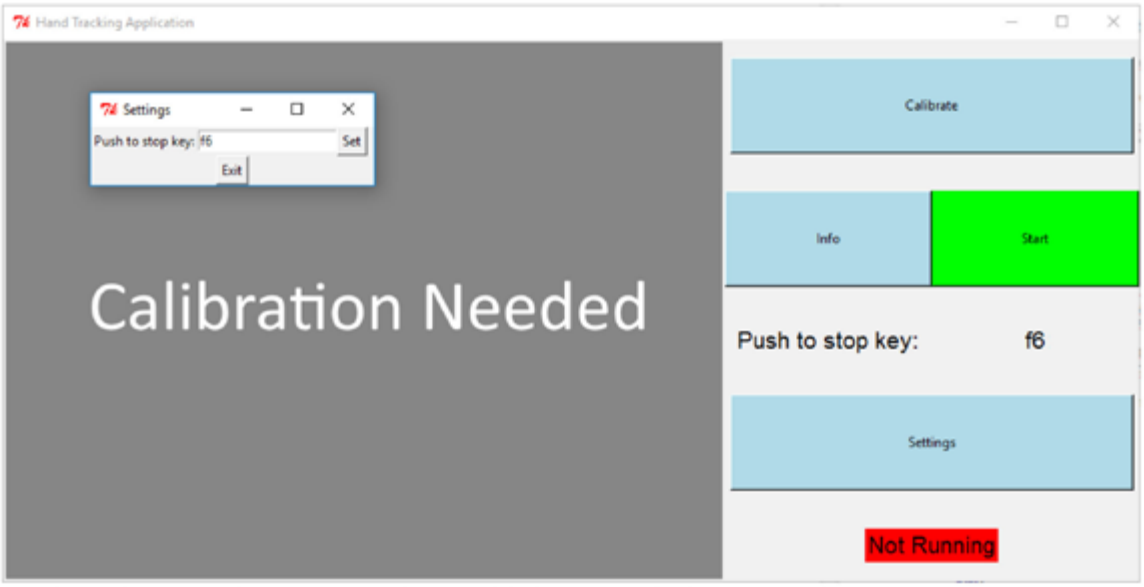
This is the code that assigns the text entered into the box to the variable stopKeyVar. To retrieve the text and to check that the user has entered it in correctly I have made this function:

Computer Science Coursework – Hand Tracking Application

```
def stopKeyGet(self):

    if len(self.stopKeyEntry.get()) == 2:
        if self.stopKeyEntry.get()[0].upper() == "F" and
int(self.stopKeyEntry.get()[1]) >= 1 and int(self.stopKeyEntry.get()[1]) <= 9:
            self.stopKeyVar = self.stopKeyEntry.get()
            print(self.stopKeyVar)
            mainApp.stopKeyTextUpdate()
        else:
            print("Input error")
    else:
        print("Input error")
```

This function checks a number of things to make sure that the input is correct. First it checks the length of the text to make sure that it is two characters (the input should be in the form "f1", or "F8" etc. and between F1 and F9, so only two characters are needed). That is followed by and if statement that checks that the first letter is an F, and the second character is one number from 1 to 9. If it passes the check then the stopKeyVar is changed, which will change the key used to stop the program. The text on the main menu now needs to be updated.

```
def stopKeyTextUpdate(self):

    try:
        self.stopKeyLabel = tk.Label(self.master, text = self.app.stopKeyVar,
font=("Helvetica", 16))
        self.stopKeyLabel.grid(row = 2, column = 2)
    except AttributeError:
     self.stopKeyLabel = tk.Label(self.master, text="F2", font=("Helvetica", 16))
        self.stopKeyLabel.grid(row=2, column=2)
```



This code updates the text. It tries to make the text equal the text in stopKeyVar, but if this has not been set I gave an AttributeError when running. To fix this I added a try and except statement that tries to change the text, and if it doesn't work it assumes that the default setting of 'F2' has not been changed, and so displays 'F2'. I made sure that the text is displays is always uppercase even if the user input a lowercase F.

The next step is to make sure that pressing the new button will actually work and stop the program.

86

To do this I made a dictionary that contains all the F-Keys and their values in win32con. When I pass an F-Key it will return the value associated with that key, which is used to check if it has been pressed.

```
def stopKeyTextUpdate(self):

    try:
        self.stopKeyLabel = tk.Label(self.master, text =
self.app.stopKeyVar.upper(),  font=("Helvetica", 16))
        self.stopKeyLabel.grid(row = 2, column = 2)
        stopKeys = {
            "F1":win32con.VK_F1,
            "F2":win32con.VK_F2,
            "F3":win32con.VK_F3,
            "F4":win32con.VK_F4,
            "F5":win32con.VK_F5,
            "F6":win32con.VK_F6,
            "F7":win32con.VK_F7,
            "F8":win32con.VK_F8,
            "F9":win32con.VK_F9,
            }
        self.stopKey = stopKeys[self.app.stopKeyVar.upper()]
        print (self.stopKey)
    except AttributeError:
        self.stopKeyLabel = tk.Label(self.master, text="F2", font=("Helvetica",
16))
        self.stopKeyLabel.grid(row=2, column=2)
        self.stopKey = win32con.VK_F2
```

Now I need to change the line of code that checks for a key press to stop looking for F2 and instead look for the F-Key that the user chose:

```
if win32api.GetAsyncKeyState(win32con.VK_F2) != 0:
```

Was changed to:

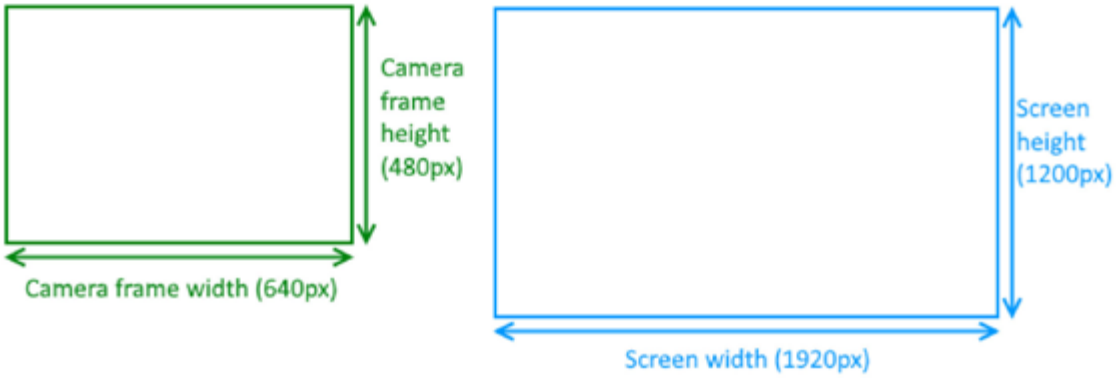
```
if win32api.GetAsyncKeyState(self.stopKey) != 0:
```

After testing this with a variety of allowed and not allowed inputs I concluded that it worked correctly.

The next feature to add into the settings is the option to change the speed of the cursor. This was harder than I thought and required some visualisation.

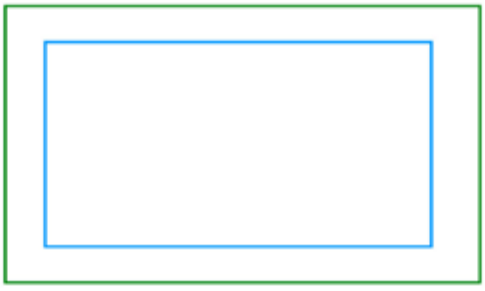
Computer Science Coursework – Hand Tracking Application

Here are two boxes representing the resolution of the camera and the resolution of the screen. At the moment, there is a one-to-one mapping of pixels on the camera frame to pixels on the screen when I move my hands. This means that if I place my hand in a corner of the camera frame, the mouse will move to the corresponding corner on the screen. If I were to speed up the mouse cursor, then to reach the corner of the screen I would not have to move my hand to the corner of the frame, because the speed has increased.



Here are the two boxes overlapped, showing the default speed and a one-to-one mapping.

Here are the two rectangles illustrating an increase in speed. Imagine my hand in the corner of the screen (blue rectangle), then you can see that it doesn't need to be in the corner of the green rectangle. The hand only moves the mouse when it is inside the blue box, and so the blue box is called the 'Active Region' (the area where hand movements can move the mouse). Note that the aspect ratio of both rectangles doesn't have to remain the same, this means that the speed of the horizontal movement can be adjusted independently of the vertical movement.



88

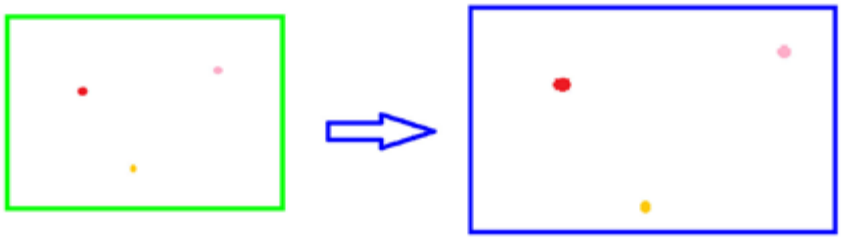
Computer Science Coursework – Hand Tracking Application

To do this I have worked out a set of steps. First, apply the location of the hand on the camera frame onto the screen using a one-to-one mapping. Do this separately for the x and y axis. Take the x location of the hand on the camera frame and divide it by the width of the camera frame, then multiply this value by the screen width. Do the same for the y axis (but with camera frame height and screen height). This is the code:

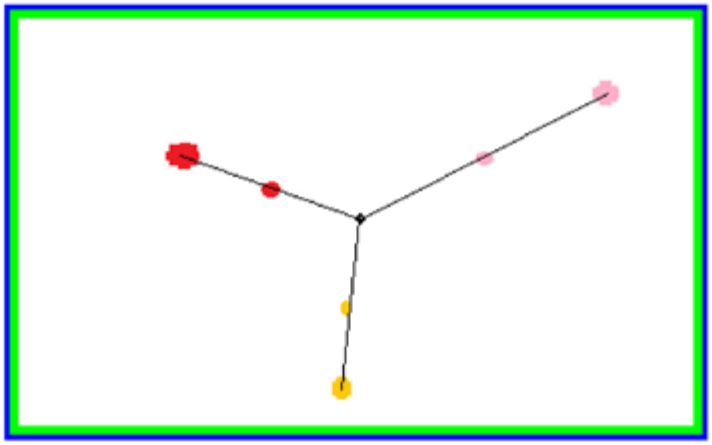
```
def newMoveMouse(self):
    camHeight = 480.0
    camWidth = 640.0
    screenHeight = 1200.0
    screenWidth = 1920.0
    x = self.handCoords[0]#Camera frame x
    y = self.handCoords[1]#Camera frame y
    x = (x/camWidth)*screenWidth#One to one mapping onto the screen resolution
    y = (y/camHeight)*screenHeight
```

The first few lines are declaring the needed variables, and the last two lines are the one-to-one mapping described earlier.

To apply the change in speed, imagine the centre of the image is in the middle of the frame (as opposed to the corner, which it is by default). To make the mouse move faster for each pixel moved by the hand, the mouse needs to move more pixels. Image the point being stretched away from the centre of the image, this would change the speed. For example, with a few different points:



If these were resized to be the same size and then overlapped:



Notice how the 'new' points are just moved away from the old ones directly away from the centre.

89

Computer Science Coursework – Hand Tracking Application

By default the centre of the image is in the corner (the location with coordinates (0, 0)). To apply this stretch the centre needs to be changed to the middle of the image. For the x coordinate of the hands just take away half of the screen width, and for the y coordinate just take away half of the screen height. This means that pixels on the left side of the image will have a negative x coordinate.

```
    x = x - (screenWidth/2)#Moves the origin (point at coordinates (0, 0) to the
middle of the frame
    y = y - (screenHeight/2)
```

Next the stretch needs to be applied by multiplying both coordinates by their respective stretch scale factors (the larger the scale factor, the faster the speed).

```
    xSF = 2 #x stretch scale factor (speed)
    ySF = 2 #y stretch scale factor
    x = x * xSF#Applies the stretch
    y = y * ySF
```

Because the camera is facing the user, the x axis needs to be flipped. This can be done by swapping the sign on the x coordinate. Then the x and y coordinate can be turned into integer form, and then the origin replaced to the corner.

```
    x = -x#Flips  the x axis
    x = x + (screenWidth/2)#Replaces the origin to the corner
    y = y + (screenHeight/2)
    x = int(x)#Puts into integer form
    y = int(y)
```

The issue with this is that some coordinates may be in locations that are beyond the screen, so some loops are needed to make sure that they are greater than or equal to 0, but smaller than or equal to the screen resolution. The mouse can then be moved.

```
if x > screenWidth:#Checks the coordinates are on the screen
        x = int(screenWidth)
    elif x < 0:
        x = 0
    if y > screenHeight:
        y = int(screenHeight)
    elif y < 0:
        y = 0
    win32api.SetCursorPos((x, y))#Moves the cursor
```

Computer Science Coursework – Hand Tracking Application

```
def moveMouse(self):

    x = self.handCoords[0] / 640.0
    y = self.handCoords[1] / 480.0
    x = 1920 - int(x*1920)
    y = int(y*1200)
    print (self.handCoords)
    win32api.SetCursorPos((x, y))
```

This is the old code that only did one speed, and this is the new code:

```
def newMoveMouse(self):
    camHeight = 480.0
    camWidth = 640.0
    screenHeight = 1200.0
    screenWidth = 1920.0
    xSF = 2 #x stretch scale factor (speed)
    ySF = 2 #y stretch scale factor
    x = self.handCoords[0]#Camera frame x
    y = self.handCoords[1]#Camera frame y
    x = (x/camWidth)*screenWidth#One to one mapping onto the screen resolution
    y = (y/camHeight)*screenHeight
    x = x - (screenWidth/2)#Moves the origin (point at coordinates (0, 0) to the
middle of the frame
    y = y - (screenHeight/2)
    x = x * xSF#Applies the stretch
    y = y * ySF
    x = -x#Flips  the x axis
    x = x + (screenWidth/2)#Replaces the origin to the corner
    y = y + (screenHeight/2)
    x = int(x)#Puts into integer form
    y = int(y)
    if x > screenWidth:#Checks the coordinates are on the screen
        x = int(screenWidth)
    elif x < 0:
        x = 0
    if y > screenHeight:
        y = int(screenHeight)
    elif y < 0:
        y = 0
    win32api.SetCursorPos((x, y))#Moves the cursor
```

Now the variables can be placed into the __init__ function of the class so that they can be edited by the GUI (so the speed can change). This has been added to the __init__ function:

```
self.xSF = 1.2#x and y scale factors the adjust the speed of the cursor

self.ySF = 2
self.cameraHeight = 480.0
self.cameraWidth = 640.0
```

91

This is the code accounting for these new global variables:

```
def newMoveMouse(self):

    x = self.handCoords[0]#Camera frame x
    y = self.handCoords[1]#Camera frame y
    x = (x/self.cameraWidth)*self.screenWidth#One to one mapping onto the screen
resolution
    y = (y/self.cameraHeight)*self.screenHeight
    x = x - (self.screenWidth/2)#Moves the origin (point at coordinates (0, 0) to
the middle of the frame
    y = y - (self.screenHeight/2)
    x = x * self.xSF#Applies the stretch
    y = y * self.ySF
    x = -x#Flips  the x axis
    x = x + (self.screenWidth/2)#Replaces the origin to the corner
    y = y + (self.screenHeight/2)
    x = int(x)#Puts into integer form
    y = int(y)
    if x > self.screenWidth:#Checks the coordinates are on the screen
        x = int(self.screenWidth)
    elif x < 0:
        x = 0
    if y > self.screenHeight:
        y = int(self.screenHeight)
    elif y < 0:
        y = 0
    win32api.SetCursorPos((x, y))#Moves the cursor
```

To make it clear to the user that the speed has changed I decided to show where the active region is within the frame.

```
def temploop(self):

    _, self.imageBGR = self.cap.read()
    self.imageHSV = cv2.cvtColor(self.imageBGR, cv2.COLOR_BGR2HSV)
    self.imageHSV = cv2.blur(self.imageHSV, (5, 5))
    self.createMask()
    self.findContours()
    x1 = int((self.cameraWidth - self.cameraWidth/self.xSF)/2)
    x2 = int(self.cameraWidth - x1)
    y1 = int((self.cameraHeight - self.cameraHeight/self.ySF)/2)
    y2 = int(self.cameraHeight - y1)
    cv2.rectangle(self.imageBGR, (x1, y1), (x2, y2), (0, 0, 255))
    cv2.putText(self.imageBGR, "Active Region", (x1, y1 - 5),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255))
```

This new code works out the corners of the active region and draws a red rectangle on the image that the user will see. There is also some text added that shows that it is the active region.

Computer Science Coursework – Hand Tracking Application

This is what it looks like:

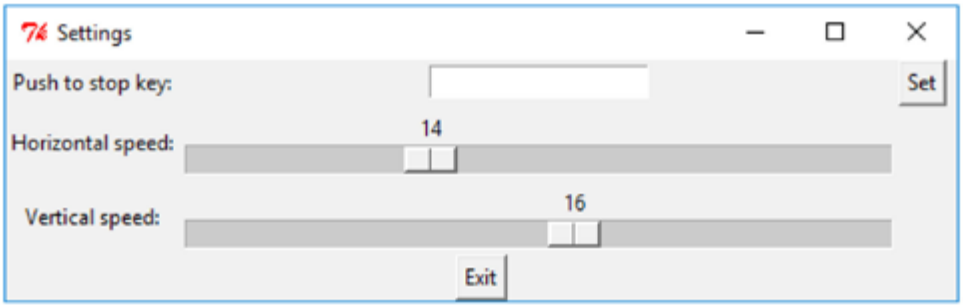
The user needs to be able to change the speed of the mouse in the settings. To do this I have added two sliders into the settings, one to change the vertical speed and the other to change the horizontal speed of the mouse. This code just creates the sliders and places them in the settings window, and gives them some text:

```
self.xSpeedText = tk.Label(self.master, text = "Horizontal speed:")

self.xSpeedText.grid(row = 1, column = 0)
self.xSpeedSlider = tk.Scale(self.master, from_ = 11, to = 20, orient =
tk.HORIZONTAL, length = 400)
self.xSpeedSlider.grid(row = 1, column = 1)
self.ySpeedText = tk.Label(self.master, text="Vertical speed:")
self.ySpeedText.grid(row=2, column=0)
self.ySpeedSlider = tk.Scale(self.master, from_=11, to=20, orient=tk.HORIZONTAL,
length=400)
self.ySpeedSlider.grid(row=2, column=1)
```

When the settings window is closed the position of these sliders will be read, and then the speed variables (xSF and ySF) will be changed. I have added this change into the close button in the settings.

```
def close_windows(self):

    cameraFeed.xSF = self.xSpeedSlider.get() / 10.0
    cameraFeed.ySF = self.ySpeedSlider.get() / 10.0
    self.master.destroy()
```

These are the new sliders and their text:



93

Testing this showed that it worked fine as both the active region changed and the mouse speed.

Another feature that is mentioned in the specification is the option to reduce processing power for computers that can not compute as much. The only way of achieving this is to cut out any unnecessary processes. The only one that can be removed and still have the program function is also one that takes up a lot of processing power, and this is the face detection. In the findContours function faceDetect is called so that the face is not mistaken for a hand. If this were removed then the program would still work so long as the user knows that their face cannot be seen by the camera, otherwise it will be detected as a hand.

To start this I have added the option for 'Low processing mode' in the settings:

```
self.lowModeText = tk.Label(self.master, text = "Low processing mode:")

self.lowModeVar = tk.IntVar()
self.lowModeCheckbutton = tk.Checkbutton(self.master, variable = self.lowModeVar,
onvalue = 1, offvalue = 0)
self.lowModeText.grid(row = 3, column = 0)
self.lowModeCheckbutton.grid(row = 3, column = 1)
```

This uses Tkinter's Checkbutton object, which is a small box that you can tick. You assign it a variable and define what the value of the variable is when the box is ticked and not ticked. The variable assigned to it has to be a Tkinter IntVar, so I made a variable lowModeVar that is an IntVar. This will interface with a variable in the MainDevelopment program (the one that does the image processing) called cameraFeed.lowMode.

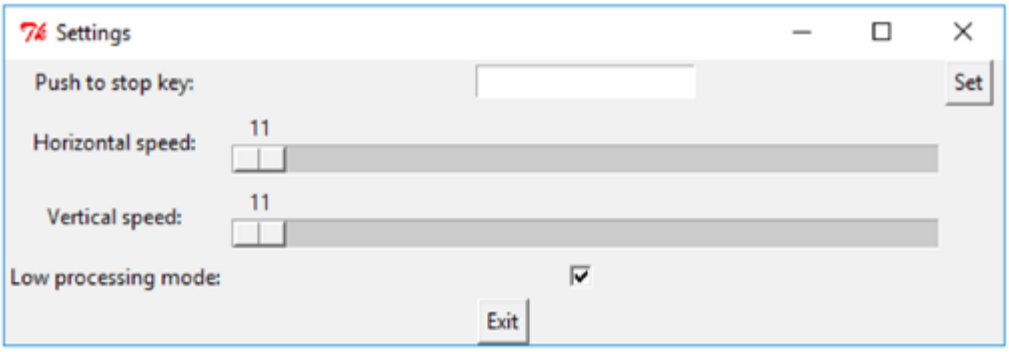
One problem is that no matter what the value of lowMode, the box is always unticked when the settings are opened. To solve this I have added this if statement that checks the state of lowMode, and if it is on the Checkbutton is toggled (turned on).

```
if cameraFeed.lowMode == 1:
    self.lowModeCheckbutton.toggle()
```

When the window is closed the lowMode variable needs to be updated, so I have added a new line to the code for when the settings are closed:

```
def close_windows(self):

    cameraFeed.xSF = self.xSpeedSlider.get() / 10.0
    cameraFeed.ySF = self.ySpeedSlider.get() / 10.0
    cameraFeed.lowMode = self.lowModeVar.get()
    self.master.destroy()
```

This is the settings so far with the sliders and the check box:



Computer Science Coursework – Hand Tracking Application

Now the low processing mode needs to implemented so that it works. This is the start of findContours where the face detect is located:

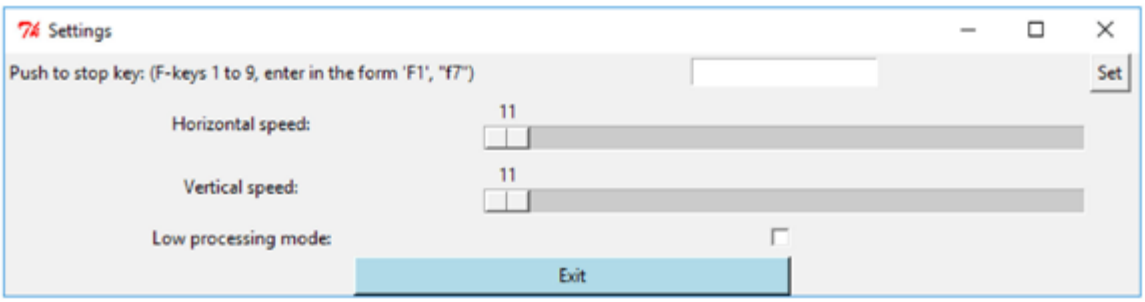
```
def findContours(self):

    self.imageGRAY = cv2.cvtColor(self.imageBGR, cv2.COLOR_BGR2GRAY)
    self.faceDetect(self.imageGRAY)
    if len(self.faceCoords) != 0:
        [fx, fy, fw, fh] = self.faceCoords
        cv2.rectangle(self.mask, (fx, fy), (fx+fw, fy+int(fh*1.5)), (0, 0, 0), -1)
```

To change this to have a low processing mode there needs to be an if statement that checks the state of the lowMode variable, and only if it is off it will call faceDetect:

```
def findContours(self):

    self.imageGRAY = cv2.cvtColor(self.imageBGR, cv2.COLOR_BGR2GRAY)
    if self.lowMode == 0:
        self.faceDetect(self.imageGRAY)
        if len(self.faceCoords) != 0:
            [fx, fy, fw, fh] = self.faceCoords
            cv2.rectangle(self.mask, (fx, fy), (fx+fw, fy+int(fh*1.5)), (0, 0, 0),
-1)
```

(This only shows the first part of findContours)

I also have made the exit button larger and gave it a light blue colour to match the theme of the main window. Here is the finished settings tab:



95

Stage 5 Review

What has been done

The settings menu has been populated with the option to change the push to stop button. There are now two sliders in the settings menu so that the horizontal and vertical speed of the mouse can be changed. This is then displayed on the camera feed as the 'Active region'. A low processing mode has been added that prevents face detection from running on every frame which lets the program run better on lower powered computers.

How it has been tested

The text entry box was tested with a variety of inputs. To make sure that only the correct input can be put in the box there are a series of 'if' statements to ensure the text is in the form "F4" or "f6" etc. When the text is entered correctly the text on the main window changes.

The other features in the settings were tested to make sure that they functioned as they were programmed. These features will be tested more throughout at the end of development during the destructive testing phase.

How it meets the success criteria and user expectations

The settings menu works, and you can change the mouse speed and change the push to stop button. There is a low processing mode.

This is the criteria that this section meets/contributes to

- Webcam feed shows what the computer 'sees'
- A settings menu in a separate window
- The option to change the push to stop button
- The option to change the mouse speed
- A low processing power mode for lower-end computers

Changes in the design that have resulted from this section

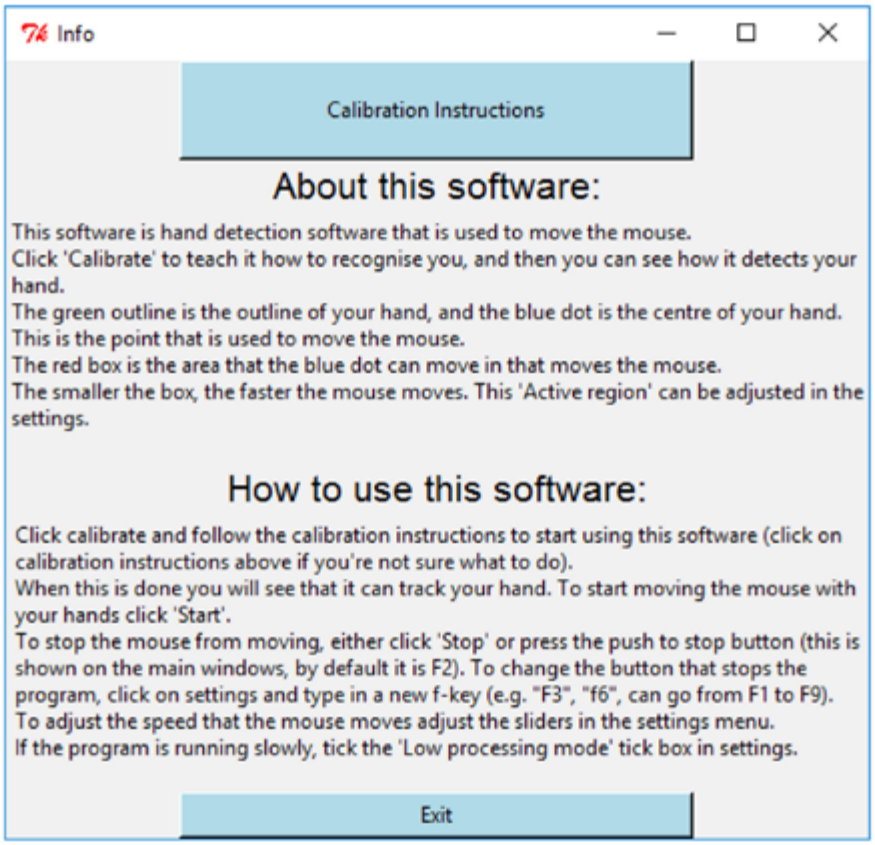
The camera feed now shows a red box that indicates the 'active region' (the region where hand movement is translated to mouse movement).

Summary of the whole project as a prototype at this stage

The only thing that this prototype is missing is the information window with instructions, and final testing. Apart from this all the features a functional and this is a working prototype.

Stage 6, information menu and calibration instructions

Currently the info window is blank. It should be filled with information on how to use the software and how to calibrate it. I have added some text to the info window, along with a button that will lead to the instructions on how to calibrate:



Computer Science Coursework – Hand Tracking Application

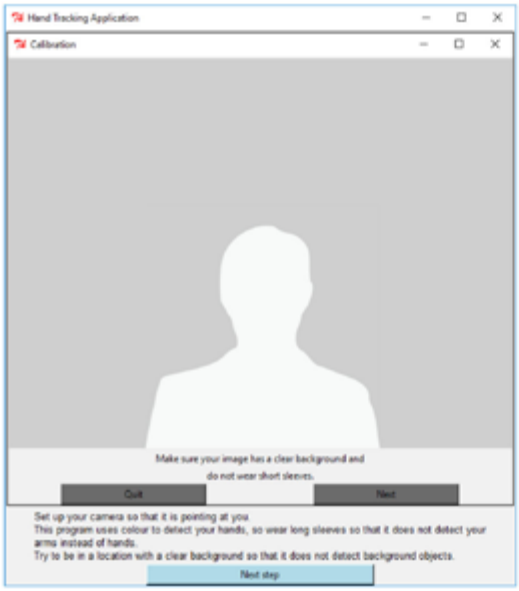
This is the code for the new information window:

```
class info():

    def __init__(self, master):
        self.master = master
        self.master.title("Info")
        self.calibrationInstructionsButton = tk.Button(self.master, text =
"Calibration Instructions", height = 3, width = 40, bg = 'LIGHT BLUE', command =
self.instructionsWindow)
        self.calibrationInstructionsButton.grid(row = 0, column = 0)
        self.infoText = tk.Label(self.master, text="About this software:", font =
("Helvetica", 16))
        self.infoText.grid(row = 1, column = 0)
        aboutThisSoftware = """This software is hand detection software that is
used to move the mouse.
Click 'Calibrate' to teach it how to recognise you, and then you can see how it
detects your hand.
The green outline is the outline of your hand, and the blue dot is the centre of
your hand.
This is the point that is used to move the mouse.
The red box is the area that the blue dot can move in that moves the mouse.
The smaller the box, the faster the mouse moves. This 'Active region' can be
adjusted in the settings.
        """
        self.aboutThisSoftwareLabel = tk.Label(self.master, text =
aboutThisSoftware, justify = tk.LEFT, wraplength = 500)
        self.aboutThisSoftwareLabel.grid(row = 2, column = 0)
        self.infoText2 = tk.Label(self.master, text="How to use this software:",
font = ("Helvetica", 16))
        self.infoText2.grid(row = 3, column = 0)
        howToUse = """Click calibrate and follow the calibration instructions to
start using this software (click on calibration instructions above if you're not
sure what to do).
When this is done you will see that it can track your hand. To start moving the
mouse with your hands click 'Start'.
To stop the mouse from moving, either click 'Stop' or press the push to stop button
(this is shown on the main windows, by default it is F2). To change the button that
stops the program, click on settings and type in a new f-key (e.g. "F3", "f6", can
go from F1 to F9).
To adjust the speed that the mouse moves adjust the sliders in the settings menu.
If the program is running slowly, tick the 'Low processing mode' tick box in
settings.
        """
        self.howToUseLabel = tk.Label(self.master, text = howToUse, justify =
tk.LEFT, wraplength = 500)
        self.howToUseLabel.grid(row = 4, column = 0)
        self.exitButton = tk.Button(master, text="Exit",
command=self.close_windows, width = 40, bg = 'LIGHT BLUE')
        self.exitButton.grid(row = 5, column = 0)
    def close_windows(self):
        self.master.destroy()
    def instructionsWindow(self):
        self.instrWindow = tk.Toplevel(self.master)
        self.app = instructions(self.instrWindow)
```

To make the calibration tutorial I took some screen shots of the calibration process and in an image editor added an example image for each step. I then created a series of windows that have these images explaining what to do, and some text and a button that leads to the next image. These are the instructions:

98

Computer Science Coursework – Hand Tracking Application



The code for each of these frames is very similar for each frame. It just removes what was on the previous frame and adds the new image and text:

```
class instructions():

    def __init__(self, master):
        self.master = master
        self.img = ImageTk.PhotoImage(Image.open("infoImg1.png"))
        self.imgLabel = tk.Label(self.master, image=self.img)
        self.imgLabel.grid(row=0, column=0)
        text = """Set up your camera so that it is pointing at you.
This program uses colour to detect your hands, so wear long sleeves so that it does
not detect your arms instead of hands.
Try to be in a location with a clear background so that it does not detect
background objects."""
        self.textLabel = tk.Label(self.master, text = text, justify = tk.LEFT,
wraplength = 600, font = ("Helvetica", 10))
        self.textLabel.grid(row = 1, column = 0)
```

99

Computer Science Coursework – Hand Tracking Application

```
        self.nextButton = tk.Button(self.master, text = "Next step", command =
self.step2, width = 40, bg = 'LIGHT BLUE')
        self.nextButton.grid(row = 2, column = 0)
    def step2(self):
        self.imgLabel.destroy()
        self.textLabel.destroy()
        self.nextButton.destroy()
        self.img = ImageTk.PhotoImage(Image.open("infoImg2.png"))
        self.imgLabel = tk.Label(self.master, image=self.img)
        self.imgLabel.grid(row=0, column=0)
        text = """The program will then try to find your face and eyes.
        When the blue box is around your face and there is a green box over each
eye (and nowhere else), click next."""
        self.textLabel = tk.Label(self.master, text=text, justify=tk.LEFT,
wraplength=600, font=("Helvetica", 10))
        self.textLabel.grid(row=1, column=0)
        self.nextButton = tk.Button(self.master, text="Next step",
command=self.step3, width = 40, bg = 'LIGHT BLUE')
        self.nextButton.grid(row=2, column=0)
    def step3(self):
        self.imgLabel.destroy()
        self.textLabel.destroy()
        self.nextButton.destroy()
        self.img = ImageTk.PhotoImage(Image.open("infoImg3.png"))
        self.imgLabel = tk.Label(self.master, image=self.img)
        self.imgLabel.grid(row=0, column=0)
        text = """You then have the option to confirm the image, or if the boxes
have moved you can re-take it."""
        self.textLabel = tk.Label(self.master, text=text, justify=tk.LEFT,
wraplength=600, font=("Helvetica", 10))
        self.textLabel.grid(row=1, column=0)
        self.nextButton = tk.Button(self.master, text="Next step",
command=self.step4, width = 40, bg = 'LIGHT BLUE')
        self.nextButton.grid(row=2, column=0)
    def step4(self):
        self.imgLabel.destroy()
        self.textLabel.destroy()
        self.nextButton.destroy()
        self.img = ImageTk.PhotoImage(Image.open("infoImg4.png"))
        self.imgLabel = tk.Label(self.master, image=self.img)
        self.imgLabel.grid(row=0, column=0)
        text = """This is what the program sees.
        Adjust the slider so that it can see your face clearly. Hold up a hand and
make sure it can see the hand clearly.
        Make sure that there is a little background noise as possible (as shown in
the top right in this image).
        Click finish, and the program is calibrated!"""
        self.textLabel = tk.Label(self.master, text=text, justify=tk.LEFT,
wraplength=600, font=("Helvetica", 10))
        self.textLabel.grid(row=1, column=0)
        self.nextButton = tk.Button(self.master, text="Finish",
command=self.close_windows, width = 40, bg = 'LIGHT BLUE')
        self.nextButton.grid(row=2, column=0)
    def close_windows(self):
        self.master.destroy()
```

Now the development is almost finished. The code is in a working state and all (apart from a few) points from the specification have been met. Now the code just needs full testing to make sure there are no bugs from a few different people, and the stakeholders need to check that it meets their requirements.

100

Stage 6 Review

What has been done:

The information menu has been filled with text containing information about the program and how to use it. There is a calibration instructions button that opens a window with clear picture based instructions of how to calibrate the program.

How it has been tested:

Most of the additions in this stage are just text, so the only functionality that needed testing was that the buttons that opened new windows worked. This was done just by using the program.

How it meets the success criteria and user expectations:

There is now an information window, and it links to clear picture based instructions.

These are the criteria it meets:

An information menu in a separate window

Text explain how to use the software and a link to calibration instructions

Clear, picture based calibration instructions

Instructions easy to find

Changes in the design that have resulted from this section:

The information menu was never designed fully as I was not sure what information would be needed, so everything in the information menu is an addition to the design.

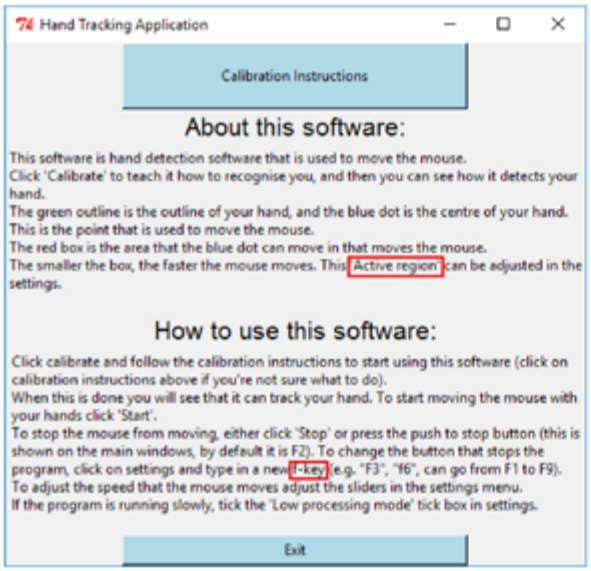
Summary of the whole project as a prototype at this stage:

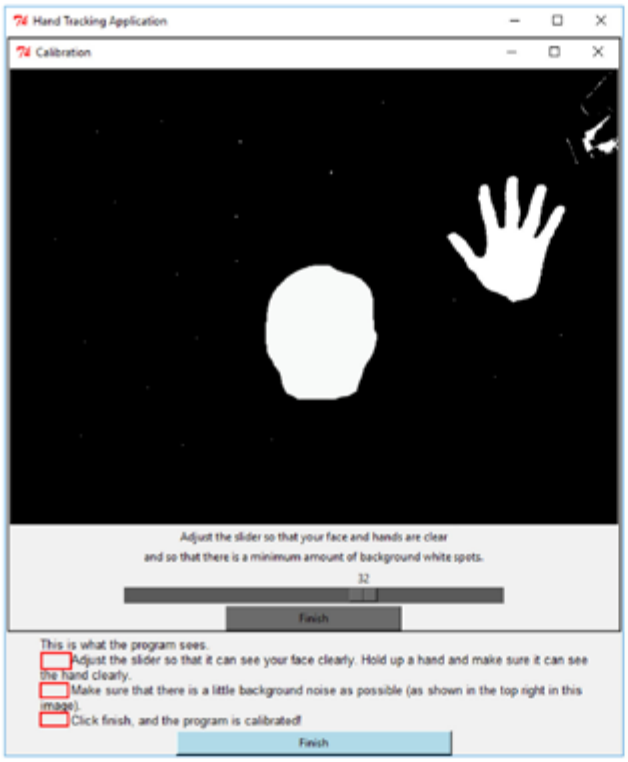
The project is now in a final state. It has full functionality and is usable. Final testing needs to be carried out to make sure there are no bugs in the code.

Computer Science Coursework – Hand Tracking Application

Stage 7, final testing

The first thing to test is the user interface to check for typos in the text and making sure that there are no usability errors.

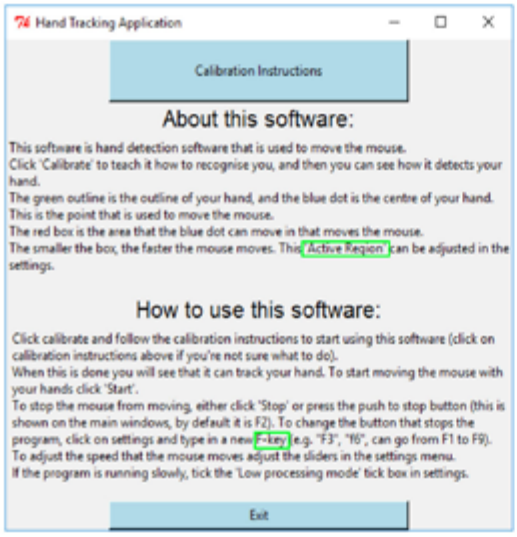




These are some text issues that I found (shown in red). The 'Active region' part should have region capitalised, to become 'Active Region'. There were some unintentional indents found in the tutorial windows. These are now fixed (shown in green).

102

Computer Science Coursework – Hand Tracking Application



A test of all the buttons showed that the user interface is working. The start button can be pressed when the program is not calibrated and no errors occur, the exit buttons all work and if a window is closed all windows higher than it also close (if the main window is closed, the settings and information menus also close).

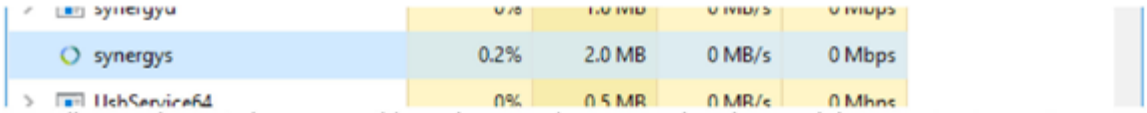
I then calibrated the program properly to test the settings one more time.

When pressing the push to stop key (F2 by default) the program crashed and I was given the following error:

```
error: (0, 'SetCursorPos', 'No error message is available')
```

I tried changing the F-key but this error still remained.

I found out that it is due to another program trying to control the mouse, in my case a program called Synergy. It had a process running in the background that was causing problems, which since it was closed the error has gone away.

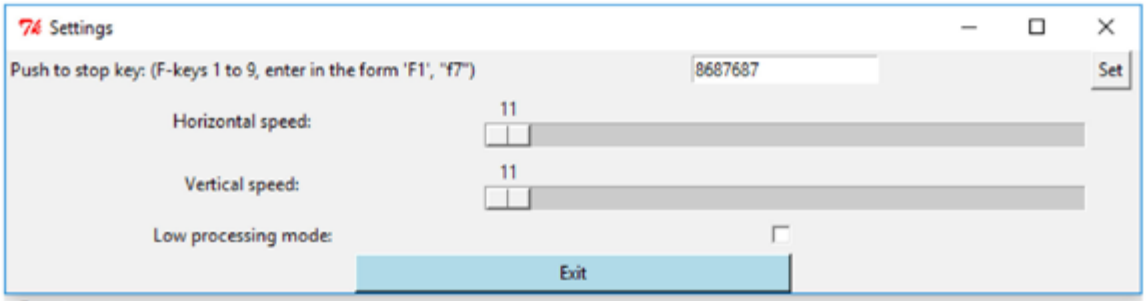


Following this I tried every possible push to stop key to test that they work (F1, F2, F3, F4, F5, F6, F7, F8 and F9) and none of them showed any errors and they worked fine.

When I change the push to stop key if I enter invalid data the key does not change, and an error is printed to the console ("Input error"). The user would likely want some feedback telling them that what they entered is incorrect.

103

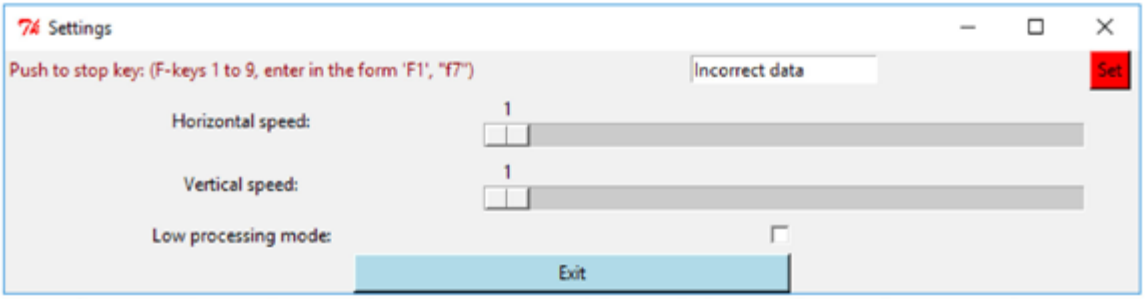
Computer Science Coursework – Hand Tracking Application



I have changed the code for the button to fix this:

```
def stopKeyGet(self):
    if len(self.stopKeyEntry.get()) == 2:
        if self.stopKeyEntry.get()[0].upper() == "F" and
int(self.stopKeyEntry.get()[1]) >= 1 and int(self.stopKeyEntry.get()[1]) <= 9:
            self.stopKeyVar = self.stopKeyEntry.get()
            print(self.stopKeyVar)
            mainApp.stopKeyTextUpdate()
            self.stopKeyButton.configure(bg="WHITE")
            self.topKeyText.configure(fg="BLACK")
        else:
            self.stopKeyInputError()
    else:
        self.stopKeyInputError()
def stopKeyInputError(self):
    self.stopKeyButton.configure(bg = "RED")
    self.topKeyText.configure(fg = "DARK RED")
```

Now when the data is incorrect the button and text turn red to show the user what they input is wrong. When then enter the correct data, this is reverted:



To destructively test this, I tried inputting a range of data including numbers and strings, and strings that are very long and anything other than the correct input data ('f1', 'F5' etc.) did not break the program and the text turned red to show me that the data I entered is not correct. This box is working as expected.

Here is the data that I prepared in the design section to test this text input box and the results:

Input	Valid/Invalid	Result
F1, F2, F3, F4, F5, F6, F7, F8, F9	Valid	Input works
f1, f2, f3, f4, f5, f6, f7, f8, f9	Valid	Input works
Test	Invalid	Red feedback, working as intended
F10	Invalid	Red feedback, working as intended

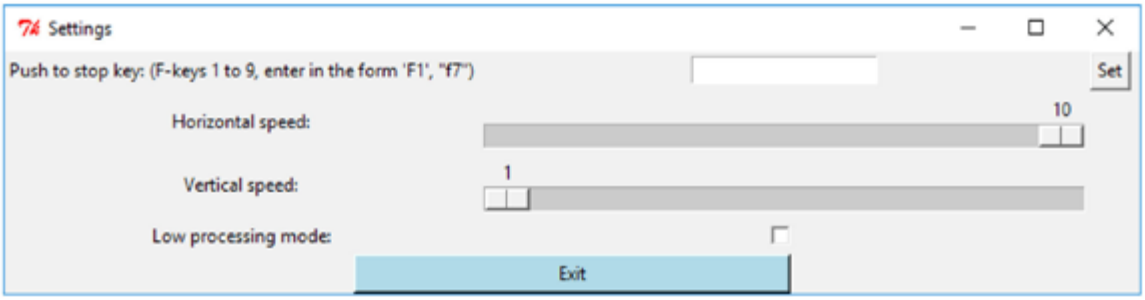
104

Computer Science Coursework – Hand Tracking Application

F1 (space followed by 'F1')	Invalid	Red feedback, working as intended
F 5	Invalid	Red feedback, working as intended
46	Invalid	Red feedback, working as intended

This shows that the validation on the inputs is working as intended.

The 'Low processing mode' check box functions well, and if it is checked and the menu is closed when the menu is reopened it remains checked. The sliders also work well, however if I change their value no matter what it is when the settings are opened they show '11'. It would also make more sense if they started from 1 and went to 10, instead of 11 to 20.



The sliders have now been changed to a scale of 1 to 10. This means that when I retrieve the value from the sliders, I have to add 10 to compensate.

Their position now stays when the menu is reopened by using the set() function, which sets the position of the sliders.

```
self.xSpeedSlider.set(cameraFeed.xSF*10 - 10)
```

The sliders position is set to the value of the speed it corresponds to, the numbers are converting it into a form that it suitable for the slider.

To test the functionality of the sliders I used this test table from the design section:

Slider 1	Slider 2	Result
Minimum	Minimum	Larger active region, low mouse speed.
Maximum	Maximum	Small active region, fast mouse speed.
Minimum	Maximum	Slow horizontal speed, fast vertical speed.
Maximum	Minimum	Fast horizontal speed, slow vertical speed
Arbitrary mid-range value	Arbitrary mid-range value	The speed and active region is adjusted accordingly with no errors.

The mouse hand tracking works as intended, with the only issues being that it picks up objects and colours in the background. This is to be expected, and the only way to change this is to change the tolerance value in the calibration or to change the algorithm that this program uses.

All the settings work. The 'Low processing mode' toggle does switch to a low processing mode that does not run face detection (which limits the load put onto the processor), and the speed sliders change the active region as intended.

105

Calibration

Calibration has a lot of room for error in terms of the data input. To start with I made sure that there was no face in the frame, and clicked 'next' in the calibration window. This was the error:

```
ValueError: need more than 0 values to unpack
```

This means that part of the program was expecting to receive the image of a face, but there was nothing there and the program cannot handle this, and crashed. To avoid crashed I wrote a function to handle crashes. It restarts the calibration process and closes the current window:

```
def crash(self):
    mainApp.calibrateWindow()
    self.close_windows()
```

The crash occurs when the stage4 function is called with no face in the image. The 'Yes' button that the user clicks calls this function. I have created a new function called gotostage4 that tries to call stage4, and if there is an error it calls the crash function above:

```
def gotostage4(self):
    try:
        self.stage4()
    except ValueError:
        self.crash()
```

Testing this showed that it stopped the program from crashing when incorrect data is entered.

It is possible for the program to only identify one eye, or to identify the mouth as an eye and have three eyes in the frame. This seems problematic, but due to how the program handles the two eyes it still works and picks samples from the face. If there are no eyes, then the calibration will restart using the function above.

Hardware

Testing with different hardware that is within the specification showed that the program still works on different machines, so long as they are powerful (which is specified in the specification). When using different webcams the program still worked, and if more than one webcam is plugged in (which is a rare setup) the program picks one of them. None of the stakeholders specified using multiple webcams so a menu to switch between them is not necessary. If the webcam is capable of more than a resolution of 640x480 then the program sets it to 640x480 so no issues occur.

Checklist

Using the checklist, I created in the development section I have gone through and thoroughly checked the functionality of each part of the program. This is the checklist that I ticked off as each thing on the list was checked:

Action to test	Working?
Check the text for formatting errors	Y
'Settings' button opens settings menu	Y
'Info button' opens information menu	Y
'Calibration instructions' button opens calibration instructions	Y
'Next' buttons show the next instruction then close the window	Y
'Calibrate' button opens the calibration window	Y
'Next' and 'Yes'/'No' buttons work in the calibration sequence	Y
'Calibration Needed' image shows when not calibrated	Y
Calibration recognises face and eyes	Y

Computer Science Coursework – Hand Tracking Application

If there are errors the 'No' can be used to retry	Y
Tolerance slider changes the amount of white in the image	Y
Calibration completes and allows the program to function	Y
'Calibration Needed' image is replaced by webcam image	Y
Overlay showing the hand outline and the 'Active Region'	Y
'Start' button stars the program and turns into a red 'Stop'	Y
'Stop' button stops the program and turns into a green 'Start'	Y
The push to stop key stops the program	Y
Moving the hand causes the mouse to move	Y
The mouse cannot move when the hand is outside the 'Active Region'	Y
The horizontal and vertical sliders can be adjusted	Y
The horizontal and vertical sliders adjust the 'Active Region' correctly	Y
Changing the push to stop key changes the push to stop key	Y
Low processing mode stops face recognition	Y

Now that I am confident that the program has no bugs and functions as intended I can pass a copy over to the stakeholders so that they can try it and then provide some feedback.

Stage 8, stakeholder testing

For the stakeholder testing I will give each stakeholder a copy of the software and then ask them to fill in a questionnaire to get their feedback.

These are the questions that I asked:

How did you find the calibration process?

Once calibrated, how did the software perform?

Did you use the settings to change anything?

Did you use the information window?

How did you then use the software?

I received the following responses:

How did you find the calibration process?

Ryan:

I managed to calibrate the software first time. I liked how it is automated and that I don't really have to do much to get it to work. The only issue I had was that sometimes it detected my mouth as an eye and I had to redo that part. When adjusting the tolerance, I had a little bit of white still in the background but I don't think it caused any problems.

Ollie:

It was pretty easy, and I managed to do it quickly. I wasn't sure if I was doing the tolerance bit correct, but later on when I saw the calibration instructions I knew that I had done it correct. I wasn't sure if the amount of white in the background was okay, but it was.

Sarah:

I wasn't too sure what to do so I looked at the instructions first. After looking at the instructions it seemed simple and I could have probably done it without looking at them. Getting it to calibrate to a child was simple, he just had to sit still and hold up his hand. Originally the room had too much background noise, so I moved the webcam then it worked.

Once calibrated, how did the software perform?

Ryan:

Pretty good, when I moved my hand the mouse moved fairly smoothly, but with occasional jitters. I tested it by painting some stuff in paint, and it was quite fun.

Ollie:

It worked fine, I held my mouse in one hand and moved my other hand to control the mouse. I was able to move around some windows on my computer and tried browsing the internet with it. I had to increase the scale of the websites to 200% to make them bigger for this to work.

Sarah:

The first time that I calibrated it, it didn't work so well. The mouse was very shaky but this was because the background was too noisy. After I moved the webcam and calibrated it again it was better.

Did you use the settings to change anything?

Ryan:

I used the settings to play around with the mouse speed to find a good speed. I also tried changing the push to stop key, although I didn't need it as I could use the big button in the program to stop the mouse from moving.

Ollie:

I had a look at the settings to see what the low processing mode did, but other than that I didn't really need to use them. The hand tracking worked fine without changing the speed of the mouse.

Sarah:

I had a look at the settings but it seemed that the only thing that I needed to adjust in the program was the tolerance which is part of the calibration. I think this is good as the settings are not essential to use.

Did you use the information window?

Ryan:

I didn't use it to find the calibration instructions as I didn't need them, but I had a look and read the text there even though I know what the software is for.

Ollie:

I used it to look at the calibration instructions which really helped when checking the tolerance slider.

Sarah:

Yeah, I used the instructions in that menu. Other than that I knew about the software already so I didn't read that bit.

How did you then use the software?

Ryan:

I tried it with paint and pained some doodles, and it worked fairly smoothly. I Then tried it with a game. I chose Civilisation 5, a game where you use the mouse to create cities and armies. It worked well, and I had to hold the mouse in my other hand to click with. After about half an hour my arm began to ache, but other than that it was very fun.

Ollie:

I used it to play Sims. It was really fun and worked well. Sometimes my hand would go out of frame without me realising because I had the program minimised. I started to have it open just looking at the webcam feed and it worked well.

Sarah:

I had a play with it in paint to draw some stuff, and then tried it on one of my students. The calibration worked well after we moved to a clearer room, and then tried playing some drag and drop games in a website. He really enjoyed it!

Review:

All the stakeholders managed to use the software well and enjoyed it. They liked the simplicity of the calibration and how the program was easy to navigate and had clear instructions. The settings menu had just enough detail as it was not essential to visit it, but some of them did to adjust the mouse speed settings.

The only complaints were issues with the premise of this software, such as arm fatigue when using it for long periods of time. Obviously, there is no simple solution for this, but lowering the height of your arm may help. Another issue was that the hand could go out of frame. A solution could be to design a new version of the software that would just be the webcam image that could go on top of games, or to show the window on a separate monitor (but not all users would have two or more monitors). These are things that could be revised in future versions of this software.

Evaluation

Criteria met

Success criteria

Criteria	Met?
Main window that shows webcam feed	Yes
Webcam feed shows what the computer 'sees'	Yes
Simple, lightweight design.	Yes
A simple, non-technical calibration	Yes
A large stop button with colour showing if the program is running	Yes
The option to press a button to stop the program	Yes
Text that shows which button this is	Yes
A settings menu in a separate window	Yes
The option to change the push to stop button	Yes
The option to change the mouse speed	Yes
A low processing power mode for lower-end computers	Yes
The option to set gestures	**No**
An information menu in a separate window	Yes
Text explain how to use the software and a link to calibration instructions	Yes
Clear, picture based calibration instructions	Yes
Instructions easy to find	Yes
Moving the hand causes the mouse to move	Yes

Computer Science Coursework – Hand Tracking Application

Evidence

Main window that shows webcam feed:

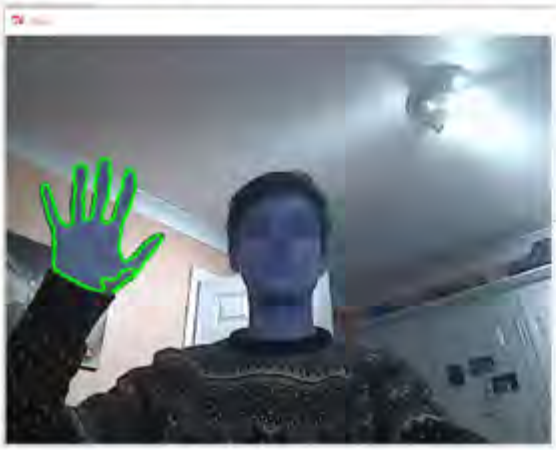
Webcam feed shows what the computer 'sees':

Simple, lightweight design:

A large stop button with colour showing if the program is running:

The option to press a button to stop the program:

The majority of the window is taken up by the camera feed. There is an overlay that shows the areas which the program thinks is skin (the green) and a red box that shows the active region. Here is another example of the program showing the hands:

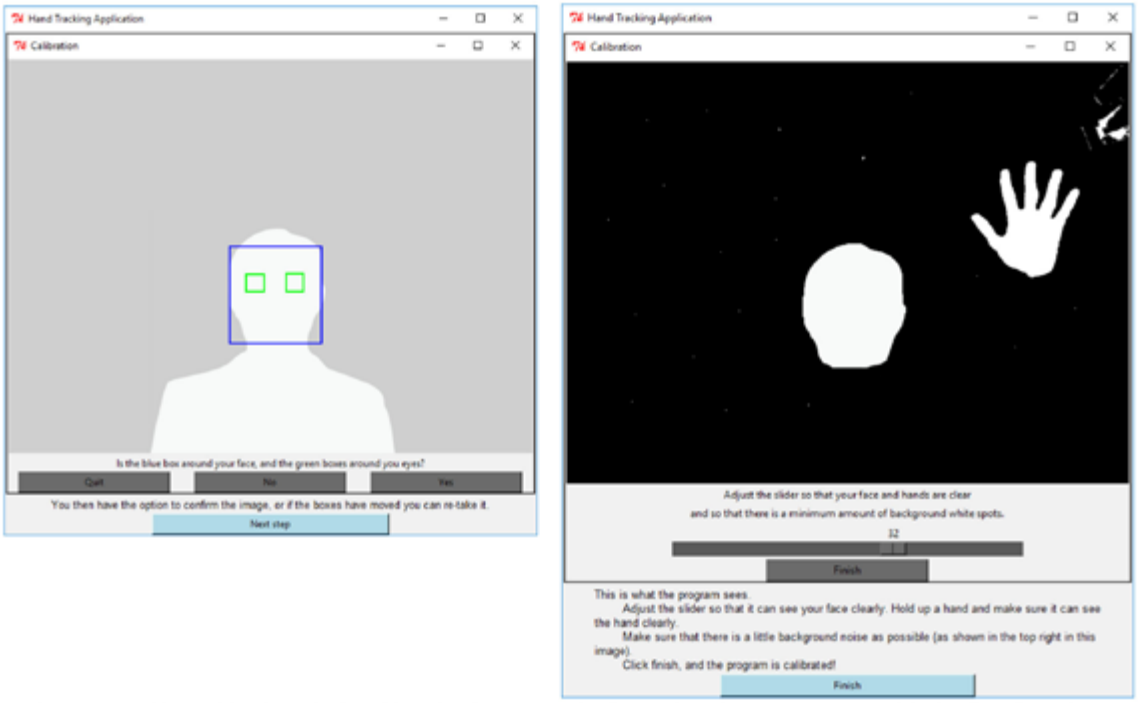


A simple, non-technical calibration:

Clear, picture based calibration instructions:

112

Computer Science Coursework – Hand Tracking Application



These are the calibration instructions that show both the calibration process and the instructions in the info menu that show how to calibrate the software with text and pictures.

An information menu in a separate window:

Text explain how to use the software and a link to calibration instructions:

Instructions easy to find:

This is the information window that can be found from the main window. This shows some text explaining how to use the software and a button that navigates to the calibration instructions.

113

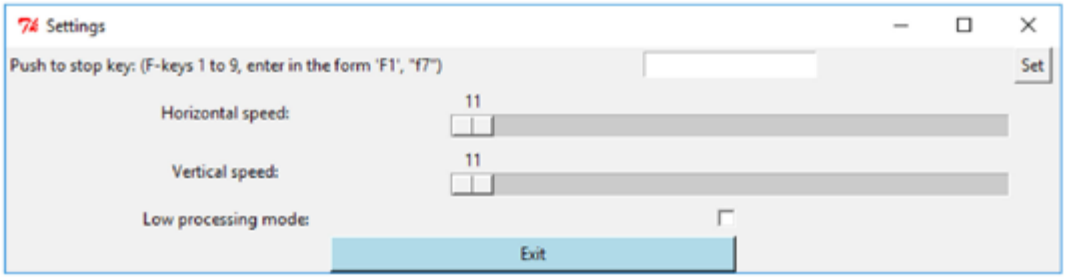
Computer Science Coursework – Hand Tracking Application

A settings menu in a separate window:

The option to change the push to stop button:

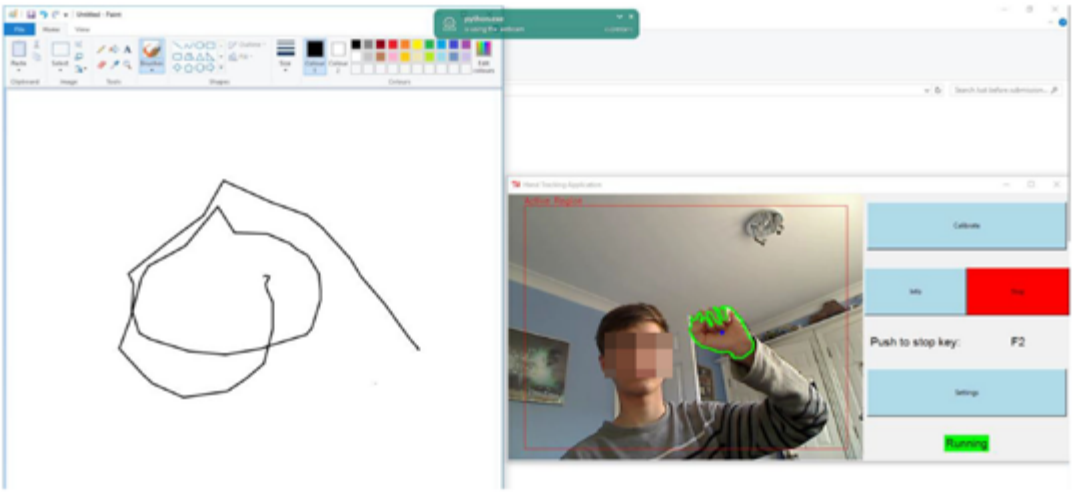
The option to change the mouse speed:

A low processing power mode for lower-end computers:



This is the settings menu where there is the option to change the push to stop button via text entry, the option to change the speed in both the vertical and horizontal axes via two sliders, and the option to turn on a low processing mode that prevents the program from doing face recognition on each frame taken.

Moving the hand causes the mouse to move:



Because of the nature of this program, this is hard to evidence through images. This image shows me using the software to draw in paint.

The option to set gestures:

This is the only item on the success criteria that was not met. The goal was to have the program recognise different gestures and bid these gestures to key presses. The gestures would have been edited in the settings and could be assigned to any key (such as left click).

114

Towards the end of development, it became clear that this would not be easily done. Implementing this feature would have been a large task and it is likely that it would not have been implemented well, and would have been prone to lots of errors such as false positives (the program thinks you performed a certain gesture when you did not, and so it presses a button). It was not feasible within the time frame I had, so I had to remove it from the final program.

All of the features included in the program have been thoroughly tested to ensure that they work. The program went through quality control during development and when it was finished I checked all of the text and user interface functionality, and made changes to fix any issues found such as typos in the text and alignment issues.

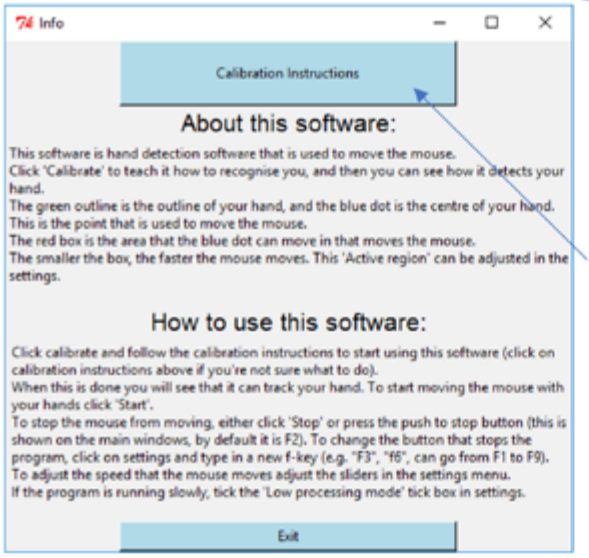
I then destructively tested the features of the program. The entry box in the settings was tested by inputting a range of incorrect data to see if it could break (this is described at the end of the development section). To test the calibration tried a range of things such as clicking 'next' when the program was expecting to see a face when there was no face in the frame to see if the program would crash, and it did. To fix this I added a crash subroutine that is called when the program wants to crash that closes and reopens the calibration window.

Usability features

The buttons are large and well labelled. It is easy to navigate the program because of this, especially when using the program to control the mouse.

As you can see the large buttons take up half of the window.

To keep the program simple looking there is a consistent design scheme throughout each window.



For example, the blue buttons in the windows

There are instructions to make sure that the user can use the calibration successfully. These are in the information window and are easy to get to as they are only two clicks away from the main menu (one click to the information menu, another to get to the instructions).

The calibration itself is very easy. The user does not have to do anything technical, only press next when prompted to do so and to move one slider. This is the instructions for how to operate the slider, showing how easy it is:

All of the complicated parts to the calibration are handled by the program so the user finds it easy to calibrate.

The windows are also well labelled, and the image from eh webcam shows what the computer sees (image overlay, outlines the hands and face) to show that the program is working.

116

Limitations

The biggest limitation is the lack of gesture support which is mentioned earlier. Future versions of this software could have support for this. Because of the mouse clicks must be done by the mouse in the other hand, which may be awkward for some users.

After long periods of time their arm may become tired. This is due to the nature of the program and is not easily solved without changing the concept of the solution.

When in use, the user may not be able to see the window, and therefore will not be able to see if their hand is within the frame or not. This means that when they are playing a game they may suddenly not be able to control the mouse anymore.

To use this software, you need a very specific environment with a clear background and good lighting. The background cannot be the same colour as your hands otherwise the software will not be able to distinguish you from the environment.

How to avoid these limitations

The gesture support could be accomplished with more development time. It would include image recognition to know which gesture is being performed. This would also mean that the user would not have to have one hand using the mouse to perform clicks.

After long periods of time some user's arms get tired, to prevent this the arm position could be lower down, which could impact the movement available but it could work as a compromise.

A small pop-up window could be implemented that goes in the corner of the screen on top of games that shows the webcam feed so that the user keeps their hands inside the image. There could also be a visual warning if their hands leave the image, or an audible alert.

To prevent the software from detecting the environment a new method of hand detection could be used, such as motion detection. This would mean that the software looks for movement to determine where the hand is instead of colour. This could also be incorporated into the current program to ignore pixels that never move (background object will not move, and so can be discarded).

Maintenance

The largest limitation with the program in its current state is that it detects a lot of background noise. The program uses colour to work out if a pixel in the image is skin, and then through other processes works out if it is a hand. If the background of the picture has a lot of colour (for example if the user has a poster in the background) then the program can pick that up as a hand, and this can cause issues. The way I have solved this in program in its current state is to tell the user to have a clean environment as the background, such as a blanc wall.

Future maintenance could change the algorithm that finds the hands to something that would not recognise this background noise. It could also add new criteria for what counts as a hand, for example looking for movement in the image.

If the stakeholders wish to add new features or changes this will be easily done as the program is heavily modular and the image processing subroutines could be swapped out and the program would still run fine. The important variables are global to the image processing program so any new features could easily view the image data without interfering with other functions.

I designed the software into two parts (two programs), one for image processing and the other for the user interface. This means that if the stakeholders wanted a new design then it would not affect the way that the program functions, and vice versa (if they want to change the way it detects hands it would not affect the user interface).

The code is also annotated mostly, and for each subroutine the inputs and the outputs are commented so that a new developer could easily interact with the subroutines.

Future versions of this software could incorporate new features such as the support for new arm positions that would have the arm lower down to reduce aching after long periods of time. The window could be designed to be smaller to just show the webcam image and have it on top of games so that the user could see if their arm is off the screen.

Final code

'Main.py'

```
import cv2
import numpy as np
import win32api

class detection():
    def __init__(self):
        self.cap = cv2.VideoCapture(0)#Camera
        self.cap.set(3, 640)#Sets the resolution
        self.cap.set(4, 480)
        self.imageBGR = []#Colour image
        self.imageGRAY = None#Grayscale image
        self.imageHSV = None#HSV image
        self.faceCoords = None#Coordinates of face
        self.eyeCoords = None#Coordinates of eyes
        self.handCoords = None#Coordinates of hand
        self.rawSamples = []#Samples collected from image [[
        self.samples = []#Samples corrected with the tolerance
        #                   V   S   H
        #                   L   D   C
        self.tolerance = [30, 30, 30]#Tolerance of upper and lower sample bounds
        self.mask = None#Mask of pixels that are within the range of the samples
        self.face_cascade =
cv2.CascadeClassifier('haarcascade_frontalface_default.xml')#Face cascade file
        self.eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')#Eye cascade
file
        self.xSF = 1.1#X and y scale factors the adjust the speed of the cursor
        self.ySF = 1.1
        self.cameraHeight = 480.0
        self.cameraWidth = 640.0
        self.screenHeight = win32api.GetSystemMetrics(1)
        self.screenWidth = win32api.GetSystemMetrics(0)
        self.lowMode = 0

    def faceDetect(self, frame):#input: grayscale frame
        faces = self.face_cascade.detectMultiScale(frame, 1.3, 5)#Haar cascade with
parameters
        if len(faces) > 0:#If it found a face
            largest = 0
            index = []#Largest algorithm
            for i in faces:
                if i[2] * i[3] > largest:#i[2]*i[3] = width*height of face
                    largest = i[2] * i[3]
                    index = i
            self.faceCoords = index#output: assigns self.faceCoords to coordinates
[x, y, w, h] of largest face in frame
        else:
            self.faceCoords = []

    def featureDetect(self, frame):#input: grayscale frame
        eyes = self.eye_cascade.detectMultiScale(frame)#Haar cascade with no
parameters
        self.eyeCoords = eyes#Output: assigns self.eyeCoords to the eye coordinates
in the form [x, y, w, h]

    def calibrateFaceDetection(self):#Takes an image, returns image of user with
face, eyes highlighted
        _, self.imageBGR = self.cap.read()
        self.imageGRAY = cv2.cvtColor(self.imageBGR, cv2.COLOR_BGR2GRAY)
        self.imageHSV = cv2.cvtColor(self.imageBGR, cv2.COLOR_BGR2HSV)
        self.imageHSV = cv2.blur(self.imageHSV, (5, 5))
        self.faceDetect(self.imageGRAY)
```

Computer Science Coursework – Hand Tracking Application

```
        if len(self.faceCoords) > 0:#If there is a face
            [x, y, w, h] = self.faceCoords
            cv2.rectangle(self.imageBGR, (x, y), (x+w, y+h), (255, 0, 0), 2)#Draws
a rectangle around face
            self.featureDetect(self.imageGRAY[y:y+h, x:x+w])#calls featureDetect
for eye detection
            for [ex, ey, ew, eh] in self.eyeCoords:
                cv2.rectangle(self.imageBGR, (ex + x, ey + y), (ex + x + ew, ey + y
+ eh), (0, 255, 0), 2)#Draws a rectangle aroudn the eyes
            if len(self.eyeCoords) != 2:#If there are not two eyes, prints and
error message
                pass#print "Eye count error"
        return self.imageBGR#returns the image

    def calibrateSamples(self):
        [x, y, w, h] = self.faceCoords
        imageFace = self.imageHSV[y:y + h, x:x + w]
        by = 0  # biggest ey
        sx = 10000  # smallest ex, arbitrarily large number
        bx = 0  # biggest ex
        for [ex, ey, ew, eh] in self.eyeCoords:
            # cv2.rectangle(imageFace, (ex, ey), (ex+ew, ey+eh), (0, 0, 0))
            if ey + eh > by:
                by = ey + eh
            if ex < sx:
                sx = ex
            if ex + ew > bx:
                bx = ex + ew
        for i in range(1, (bx - sx) // 10 + 1):
            self.rawSamples.append(imageFace[by][sx + i * 10])
            upper, lower = [], []
            for x in range(len(self.tolerance)):
                upper.append(imageFace[by][sx + i * 10][x] + self.tolerance[x])
                lower.append(imageFace[by][sx + i * 10][x] - self.tolerance[x])

            for y in range(len(upper)):  # so that 0<upper<255
                if upper[y] < 0:
                    upper[y] = 0
                if upper[y] > 255:
                    upper[y] = 255
            for y in range(len(lower)):  # so that 0<lower<255
                if lower[y] < 0:
                    lower[y] = 0
                if lower[y] > 255:
                    lower[y] = 255
            lower = np.array(lower, dtype=np.uint8)
            upper = np.array(upper, dtype=np.uint8)
            self.samples.append([lower, upper])

    def createMask(self):
        self.mask = cv2.inRange(self.imageHSV, self.samples[0][0],
self.samples[0][1]) #just to create mask of correct size
        for i in self.samples:
            tempMask = cv2.inRange(self.imageHSV, i[0], i[1])
            self.mask = cv2.bitwise_or(self.mask, tempMask)

    def findContours(self):
        self.imageGRAY = cv2.cvtColor(self.imageBGR, cv2.COLOR_BGR2GRAY)
        if self.lowMode == 0:
            self.faceDetect(self.imageGRAY)
            if len(self.faceCoords) != 0:
                [fx, fy, fw, fh] = self.faceCoords
                cv2.rectangle(self.mask, (fx, fy), (fx+fw, fy+int(fh*1.5)), (0, 0,
0), -1)
        _, contours, _ = cv2.findContours(self.mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
        if len(contours) != 0:
            largest = 0
```

120

Computer Science Coursework – Hand Tracking Application

```
            contour = None
            for cnt in contours:
                x, y, w, h = cv2.boundingRect(cnt)
                if w*h > largest:
                    largest = w*h
                    contour = cnt
            cv2.drawContours(self.imageBGR, [contour], 0, (0, 255, 0), 2)
            hx, hy, hw, hh = cv2.boundingRect(contour)
            self.handCoords = [hx + hw//2, hy + hh//2, hw*hh]
            cv2.circle(self.imageBGR, (self.handCoords[0], self.handCoords[1]), 4,
(255, 0, 0), -1)

    def temploop(self):
        _, self.imageBGR = self.cap.read()
        self.imageHSV = cv2.cvtColor(self.imageBGR, cv2.COLOR_BGR2HSV)
        self.imageHSV = cv2.blur(self.imageHSV, (5, 5))
        self.createMask()
        self.findContours()
        x1 = int((self.cameraWidth - self.cameraWidth/self.xSF)/2)
        x2 = int(self.cameraWidth - x1)
        y1 = int((self.cameraHeight - self.cameraHeight/self.ySF)/2)
        y2 = int(self.cameraHeight - y1)
        cv2.rectangle(self.imageBGR, (x1, y1), (x2, y2), (0, 0, 255))
        cv2.putText(self.imageBGR, "Active Region", (x1, y1 - 5),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255))

    def changeTolerance(self, value):
        self.tolerance = [value, value, value]
        self.samples = []
        for i in self.rawSamples:
            upper, lower = [], []
            upper.append(i[0] + self.tolerance[0])
            upper.append(i[1] + self.tolerance[1])
            upper.append(i[2] + self.tolerance[2])

            lower.append(i[0] - self.tolerance[0])
            lower.append(i[1] - self.tolerance[1])
            lower.append(i[2] - self.tolerance[2])

            for y in range(len(upper)):  # so that 0<upper<255
                if upper[y] < 0:
                    upper[y] = 0
                if upper[y] > 255:
                    upper[y] = 255
            for y in range(len(lower)):  # so that 0<lower<255
                if lower[y] < 0:
                    lower[y] = 0
                if lower[y] > 255:
                    lower[y] = 255
            lower = np.array(lower, dtype=np.uint8)
            upper = np.array(upper, dtype=np.uint8)
            self.samples.append([lower, upper])

    def moveMouse(self):
        x = self.handCoords[0] / 640.0
        y = self.handCoords[1] / 480.0
        x = 1920 - int(x*1920)
        y = int(y*1200)
        print (self.handCoords)
        win32api.SetCursorPos((x, y))

    def newMoveMouse(self):
        screenHeight = 1200.0
        screenWidth = 1920.0
        x = self.handCoords[0]#Camera frame x
        y = self.handCoords[1]#Camera frame y
        x = (x/self.cameraWidth)*self.screenWidth#One to one mapping onto the
screen resolution
```

121

Computer Science Coursework – Hand Tracking Application

```
        y = (y/self.cameraHeight)*self.screenHeight
        x = x - (self.screenWidth/2)#Moves the origin (point at coordinates (0, 0)
to the middle of the frame
        y = y - (self.screenHeight/2)
        x = x * self.xSF#Applies the stretch
        y = y * self.ySF
        x = -x#Flips  the x axis
        x = x + (self.screenWidth/2)#Replaces the origin to the corner
        y = y + (self.screenHeight/2)
        x = int(x)#Puts into integer form
        y = int(y)

        if x > self.screenWidth:#Checks the coordinates are on the screen
            x = int(self.screenWidth)
        elif x < 0:
            x = 0

        if y > self.screenHeight:
            y = int(self.screenHeight)
        elif y < 0:
            y = 0

        win32api.SetCursorPos((x, y))#Moves the cursor


"""
test = detection()
for i in range(100):
    cv2.imshow("Testing", test.calibrateFaceDetection())
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break
if len(test.faceCoords) != 0:
    test.calibrateSamples()

while True:
    test.temploop()
    cv2.imshow("Image", test.imageBGR)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break
cv2.destroyAllWindows()
test.cap.release()
"""
```

122

Computer Science Coursework – Hand Tracking Application

'GUI.py'

```
import Tkinter as tk
from PIL import Image, ImageTk
import cv2
import Main
import win32api
import win32con

calibrated = 0

class mainClass:
    def __init__(self, master):
        self.master = master
        self.master.title("Hand Tracking Application")

        self.running = 0

        self.img = ImageTk.PhotoImage(Image.open("CalibrationNeeded.png"))#The
calibration image
        self.imageLabel = tk.Label(master, image = self.img)#Sets the frame to show
the calibration image by default
        self.imageLabel.grid(row = 0, column = 0, rowspan = 5)#Location of the
image
        self.imageLoop()

        self.calibrateButton = tk.Button(master, text = "Calibrate", command =
self.calibrateWindow, height = 5, width = 50, bg = 'LIGHT BLUE')
        self.calibrateButton.grid(row = 0, column = 1, columnspan = 2)

        self.infoButton = tk.Button(master, text = "Info", command =
self.infoWindow, height = 5, width = 25, bg = 'LIGHT BLUE')
        self.infoButton.grid(row = 1, column = 1)

        self.stopButton = tk.Button(master, text = "Start", height = 5, width = 25,
bg = 'GREEN', command = self.startStop)
        self.stopButton.grid(row = 1, column = 2)

        self.stopLabel = tk.Label(master, text = "Push to stop key:",
font=("Helvetica", 16))
        self.stopLabel.grid(row = 2, column = 1)

        self.settingsButton = tk.Button(master, text = "Settings", command =
self.settingsWindow, height = 5, width = 50, bg = 'LIGHT BLUE')
        self.settingsButton.grid(row = 3, column = 1, columnspan = 2)

        self.runningLabel = tk.Label(master, text = "Not Running",
font=("Helvetica", 16), bg = 'RED')
        self.runningLabel.grid(row = 4, column = 1, columnspan = 2)

        self.stopKeyTextUpdate()

    def stopKeyTextUpdate(self):
        try:
            self.stopKeyLabel = tk.Label(self.master, text =
self.app.stopKeyVar.upper(),  font=("Helvetica", 16))
            self.stopKeyLabel.grid(row = 2, column = 2)
            stopKeys = {
                "F1":win32con.VK_F1,
                "F2":win32con.VK_F2,
                "F3":win32con.VK_F3,
                "F4":win32con.VK_F4,
                "F5":win32con.VK_F5,
                "F6":win32con.VK_F6,
                "F7":win32con.VK_F7,
                "F8":win32con.VK_F8,
                "F9":win32con.VK_F9,
                }
```

123

Computer Science Coursework – Hand Tracking Application

```
            self.stopKey = stopKeys[self.app.stopKeyVar.upper()]
            print (self.stopKey)
        except AttributeError:
            self.stopKeyLabel = tk.Label(self.master, text="F2", font=("Helvetica",
16))
            self.stopKeyLabel.grid(row=2, column=2)
            self.stopKey = win32con.VK_F2

    def calibrateWindow(self):
        self.newWindow = tk.Toplevel(self.master)
        self.app = calibration(self.newWindow)

    def infoWindow(self):
        self.newWindow = tk.Toplevel(self.master)
        self.app = info(self.newWindow)

    def settingsWindow(self):
        self.newWindow = tk.Toplevel(self.master)
        self.app = settings(self.newWindow)

    def imageLoop(self):
        global calibrated
        if calibrated:#checks the calibrated variable
            cameraFeed.temploop()#If it has been calibrated, it shows the camera
image
            image = cameraFeed.imageBGR
            cv2image = cv2.cvtColor(image, cv2.COLOR_BGR2RGBA)
            img = Image.fromarray(cv2image)
            imgtk = ImageTk.PhotoImage(image=img)
            self.imageLabel.imgtk = imgtk
            self.imageLabel.configure(image=imgtk)
            if self.running:
                cameraFeed.newMoveMouse()
                if win32api.GetAsyncKeyState(self.stopKey) != 0:
                    self.startStop()
        else:
            self.imageLabel.configure(image=self.img)#Else, it shows the
'calibration needed' image
        self.imageLabel.after(10, self.imageLoop)#Loops

    def startStop(self):
        if calibrated:
            self.running = not(self.running)
            if self.running:
                self.stopButton.configure(bg = 'RED', text = "Stop")
                self.runningLabel.configure(bg = 'GREEN', text = "Running")
            else:
                self.stopButton.configure(bg = 'GREEN', text = "Start")
                self.runningLabel.configure(bg='RED', text="Not Running")

class calibration:
    def __init__(self, master):
        global calibrated
        calibrated = 0

        self.master = master
        self.master.title("Calibration")

        self.imageLabel = tk.Label(self.master)
        self.imageLabel.grid(row=0, column=0, columnspan=2)
        self.showFrame()

        self.quitButton = tk.Button(self.master, text='Quit', width=25,
command=self.close_windows)
        self.quitButton.grid(row=3, column=0)

        self.nextButton = tk.Button(self.master, text='Next', command =
self.stage2, width=25)
```

124

Computer Science Coursework – Hand Tracking Application

```
        self.nextButton.grid(row=3, column=1)

        self.instructions = tk.Label(master, text="Make sure your image has a clear
background and")
        self.instructions2 = tk.Label(master, text="do not wear short sleeves.")
        self.instructions.grid(row=1, column=0, columnspan=2)
        self.instructions2.grid(row=2, column=0, columnspan=2)

    def showFrame(self):
        _, image = cameraFeed.cap.read()#Captures an image
        cv2image = cv2.cvtColor(image, cv2.COLOR_BGR2RGBA)#Converts the format of
the image
        img = Image.fromarray(cv2image)#Converts it again to a format that tkinter
accepts
        imgtk = ImageTk.PhotoImage(image=img)
        self.imageLabel.imgtk = imgtk
        self.imageLabel.configure(image=imgtk)#Sets the image
        self.imageLabel.after(10, self.showFrame)#Loops

    def showFace(self):
        image = cameraFeed.calibrateFaceDetection()
        cv2image = cv2.cvtColor(image, cv2.COLOR_BGR2RGBA)
        img = Image.fromarray(cv2image)
        imgtk = ImageTk.PhotoImage(image=img)
        self.imageLabel.imgtk = imgtk
        self.imageLabel.configure(image=imgtk)
        self.imageLabel.after(10, self.showFace)

    def showMask(self):
        cameraFeed.changeTolerance(self.toleranceSlider.get())
        _, cameraFeed.imageBGR = cameraFeed.cap.read()
        cameraFeed.imageHSV = cv2.cvtColor(cameraFeed.imageBGR, cv2.COLOR_BGR2HSV)
        cameraFeed.imageHSV = cv2.blur(cameraFeed.imageHSV, (5, 5))
        cameraFeed.createMask()

        image = cameraFeed.mask
        img = Image.fromarray(image)
        imgtk = ImageTk.PhotoImage(image=img)
        self.imageLabel.imgtk = imgtk
        self.imageLabel.configure(image=imgtk)
        self.imageLabel.after(10, self.showMask)

    def close_windows(self):
        self.master.destroy()

    def stage2(self):
        self.instructions.destroy()
        self.instructions2.destroy()
        self.nextButton.destroy()
        self.quitButton.destroy()
        self.imageLabel.destroy()

        self.imageLabel = tk.Label(self.master)
        self.imageLabel.grid(row=0, column=0, columnspan=2)
        self.showFace()

        self.instructions = tk.Label(self.master, text = "Ensure your face is
within the blue box, and there is a green box around each eye.")
        self.instructions2 = tk.Label(self.master, text = "When there is, click
'Next'")
        self.instructions.grid(row = 1, column = 0, columnspan = 2)
        self.instructions2.grid(row = 2, column = 0, columnspan = 2)

        self.quitButton = tk.Button(self.master, text = 'Quit', width = 25, command
= self.close_windows)
        self.quitButton.grid(row = 3, column = 0)

        self.nextButton = tk.Button(self.master, text = 'Next', width = 25, command
```

125

Computer Science Coursework – Hand Tracking Application

```
= self.stage3)
        self.nextButton.grid(row = 3, column = 1)

    def gotostage2(self):
        self.yesButton.destroy()
        self.noButton.destroy()
        self.stage2()

    def stage3(self):
        self.instructions.destroy()
        self.instructions2.destroy()
        self.nextButton.destroy()
        self.quitButton.destroy()
        self.imageLabel.destroy()

        self.imageLabel = tk.Label(self.master)
        self.imageLabel.grid(row=0, column=0, columnspan=3)

        image = cameraFeed.imageBGR
        cv2image = cv2.cvtColor(image, cv2.COLOR_BGR2RGBA)
        img = Image.fromarray(cv2image)
        imgtk = ImageTk.PhotoImage(image=img)
        self.imageLabel.imgtk = imgtk
        self.imageLabel.configure(image=imgtk)

        self.instructions = tk.Label(self.master, text="Is the blue box around your
face, and the green boxes around your eyes?")
        self.instructions.grid(row=1, column=0, columnspan=3)

        self.yesButton = tk.Button(self.master, text='Yes', width=25, command =
self.gotostage4)
        self.noButton = tk.Button(self.master, text='No', width=25, command =
self.gotostage2)
        self.yesButton.grid(row=2, column=2)
        self.noButton.grid(row=2, column=1)

        self.quitButton = tk.Button(self.master, text='Quit', width=25,
command=self.close_windows)
        self.quitButton.grid(row=2, column=0)

    def gotostage4(self):
        try:
            self.stage4()
        except ValueError:
            self.crash()

    def stage4(self):
        self.instructions.destroy()
        self.yesButton.destroy()
        self.noButton.destroy()
        self.quitButton.destroy()
        self.imageLabel.destroy()

        self.imageLabel = tk.Label(self.master)
        self.imageLabel.grid(row=0, column=0, columnspan=3)

        self.instructions = tk.Label(self.master, text = "Adjust the slider so that
your face and hands are clear")
        self.instructions2 = tk.Label(self.master, text="and so that there is a
minimum amount of background white spots.")
        self.instructions.grid(row = 1, column = 0, columnspan = 3)
        self.instructions2.grid(row=2, column=0, columnspan=3)

        self.toleranceSlider = tk.Scale(self.master, from_ = 0, to = 50, orient =
tk.HORIZONTAL, length = 400)
        self.toleranceSlider.grid(row = 3, column = 0, columnspan = 3)

        self.finishButton = tk.Button(self.master, text = "Finish", width = 25,
```

126

Computer Science Coursework – Hand Tracking Application

```
command = self.finish)
        self.finishButton.grid(row = 4, column = 0, columnspan = 3)

        cameraFeed.calibrateSamples()
        self.showMask()

    def finish(self):
        global calibrated
        calibrated = 1
        self.close_windows()

    def crash(self):
        mainApp.calibrateWindow()
        self.close_windows()

class info():
    def __init__(self, master):
        self.master = master

        self.calibrationInstructionsButton = tk.Button(self.master, text =
"Calibration Instructions", height = 3, width = 40, bg = 'LIGHT BLUE', command =
self.instructionsWindow)
        self.calibrationInstructionsButton.grid(row = 0, column = 0)

        self.infoText = tk.Label(self.master, text="About this software:", font =
("Helvetica", 16))
        self.infoText.grid(row = 1, column = 0)
        aboutThisSoftware = """This software is hand detection software that is
used to move the mouse.
Click 'Calibrate' to teach it how to recognise you, and then you can see how it
detects your hand.
The green outline is the outline of your hand, and the blue dot is the centre of
your hand.
This is the point that is used to move the mouse.
The red box is the area that the blue dot can move in that moves the mouse.
The smaller the box, the faster the mouse moves. This 'Active Region' can be
adjusted in the settings.
        """
        self.aboutThisSoftwareLabel = tk.Label(self.master, text =
aboutThisSoftware, justify = tk.LEFT, wraplength = 500)
        self.aboutThisSoftwareLabel.grid(row = 2, column = 0)

        self.infoText2 = tk.Label(self.master, text="How to use this software:",
font = ("Helvetica", 16))
        self.infoText2.grid(row = 3, column = 0)
        howToUse = """Click calibrate and follow the calibration instructions to
start using this software (click on calibration instructions above if you're not
sure what to do).
When this is done you will see that it can track your hand. To start moving the
mouse with your hands click 'Start'.
To stop the mouse from moving, either click 'Stop' or press the push to stop button
(this is shown on the main windows, by default it is F2). To change the button that
stops the program, click on settings and type in a new F-key (e.g. "F3", "f6", can
go from F1 to F9).
To adjust the speed that the mouse moves adjust the sliders in the settings menu.
If the program is running slowly, tick the 'Low processing mode' tick box in
settings.
        """
        self.howToUseLabel = tk.Label(self.master, text = howToUse, justify =
tk.LEFT, wraplength = 500)
        self.howToUseLabel.grid(row = 4, column = 0)

        self.exitButton = tk.Button(master, text="Exit",
command=self.close_windows, width = 40, bg = 'LIGHT BLUE')
        self.exitButton.grid(row = 5, column = 0)

    def close_windows(self):
        self.master.destroy()
```

127

Computer Science Coursework – Hand Tracking Application

```
    def instructionsWindow(self):
        self.instrWindow = tk.Toplevel(self.master)
        self.app = instructions(self.instrWindow)

class instructions():
    def __init__(self, master):
        self.master = master
        self.img = ImageTk.PhotoImage(Image.open("infoImg1.png"))
        self.imgLabel = tk.Label(self.master, image=self.img)
        self.imgLabel.grid(row=0, column=0)
        text = """Set up your camera so that it is pointing at you.
This program uses colour to detect your hands, so wear long sleeves so that it does
not detect your arms instead of hands.
Try to be in a location with a clear background so that it does not detect
background objects."""
        self.textLabel = tk.Label(self.master, text = text, justify = tk.LEFT,
wraplength = 600, font = ("Helvetica", 10))
        self.textLabel.grid(row = 1, column = 0)

        self.nextButton = tk.Button(self.master, text = "Next step", command =
self.step2, width = 40, bg = 'LIGHT BLUE')
        self.nextButton.grid(row = 2, column = 0)

    def step2(self):
        self.imgLabel.destroy()
        self.textLabel.destroy()
        self.nextButton.destroy()

        self.img = ImageTk.PhotoImage(Image.open("infoImg2.png"))
        self.imgLabel = tk.Label(self.master, image=self.img)
        self.imgLabel.grid(row=0, column=0)
        text = """The program will then try to find your face and eyes.
When the blue box is around your face and there is a green box over each eye (and
nowhere else), click next."""
        self.textLabel = tk.Label(self.master, text=text, justify=tk.LEFT,
wraplength=600, font=("Helvetica", 10))
        self.textLabel.grid(row=1, column=0)

        self.nextButton = tk.Button(self.master, text="Next step",
command=self.step3, width = 40, bg = 'LIGHT BLUE')
        self.nextButton.grid(row=2, column=0)

    def step3(self):
        self.imgLabel.destroy()
        self.textLabel.destroy()
        self.nextButton.destroy()

        self.img = ImageTk.PhotoImage(Image.open("infoImg3.png"))
        self.imgLabel = tk.Label(self.master, image=self.img)
        self.imgLabel.grid(row=0, column=0)
        text = """You then have the option to confirm the image, or if the boxes
have moved you can re-take it."""
        self.textLabel = tk.Label(self.master, text=text, justify=tk.LEFT,
wraplength=600, font=("Helvetica", 10))
        self.textLabel.grid(row=1, column=0)

        self.nextButton = tk.Button(self.master, text="Next step",
command=self.step4, width = 40, bg = 'LIGHT BLUE')
        self.nextButton.grid(row=2, column=0)

    def step4(self):
        self.imgLabel.destroy()
        self.textLabel.destroy()
        self.nextButton.destroy()

        self.img = ImageTk.PhotoImage(Image.open("infoImg4.png"))
        self.imgLabel = tk.Label(self.master, image=self.img)
```

128

Computer Science Coursework – Hand Tracking Application

```
        self.imgLabel.grid(row=0, column=0)
        text = """This is what the program sees.
Adjust the slider so that it can see your face clearly. Hold up a hand and make
sure it can see the hand clearly.
Make sure that there is a little background noise as possible (as shown in the top
right in this image).
Click finish, and the program is calibrated!"""
        self.textLabel = tk.Label(self.master, text=text, justify=tk.LEFT,
wraplength=600, font=("Helvetica", 10))
        self.textLabel.grid(row=1, column=0)

        self.nextButton = tk.Button(self.master, text="Finish",
command=self.close_windows, width = 40, bg = 'LIGHT BLUE')
        self.nextButton.grid(row=2, column=0)

    def close_windows(self):
        self.master.destroy()


class settings():
    def __init__(self, master):
        self.master = master
        self.master.title("Settings")

        self.topKeyText = tk.Label(self.master, text="""Push to stop key: (F-keys 1
to 9, enter in the form 'F1', "f7")""")
        self.topKeyText.grid(row = 0, column = 0, columnspan = 1)

        self.topKeyText2 = tk.Label(self.master, text="")
        self.topKeyText2.grid(row = 0, column = 3)

        self.stopKeyVar = tk.StringVar()
        self.stopKeyEntry = tk.Entry(self.master, textvariable = self.stopKeyVar)
        self.stopKeyEntry.grid(row = 0, column = 1)

        self.stopKeyButton = tk.Button(self.master, text = "Set", command =
self.stopKeyGet)
        self.stopKeyButton.grid(row = 0, column = 2)

        self.xSpeedText = tk.Label(self.master, text = "Horizontal speed:")
        self.xSpeedText.grid(row = 1, column = 0)
        self.xSpeedSlider = tk.Scale(self.master, from_ = 1, to = 10, orient =
tk.HORIZONTAL, length = 400)
        self.xSpeedSlider.set(cameraFeed.xSF*10 - 10)
        self.xSpeedSlider.grid(row = 1, column = 1)

        self.ySpeedText = tk.Label(self.master, text="Vertical speed:")
        self.ySpeedText.grid(row=2, column=0)
        self.ySpeedSlider = tk.Scale(self.master, from_=1, to=10,
orient=tk.HORIZONTAL, length=400)
        self.ySpeedSlider.set(cameraFeed.ySF*10 - 10)
        self.ySpeedSlider.grid(row=2, column=1)

        self.lowModeText = tk.Label(self.master, text = "Low processing mode:")
        self.lowModeVar = tk.IntVar()
        self.lowModeCheckbutton = tk.Checkbutton(self.master, variable =
self.lowModeVar, onvalue = 1, offvalue = 0)
        self.lowModeText.grid(row = 3, column = 0)
        self.lowModeCheckbutton.grid(row = 3, column = 1)
        if cameraFeed.lowMode == 1:
            self.lowModeCheckbutton.toggle()

        self.exitButton = tk.Button(master, text="Exit",
command=self.close_windows, width = 40, bg = 'LIGHT BLUE')
        self.exitButton.grid(row = 4, column = 0, columnspan = 4)

    def stopKeyGet(self):
        if len(self.stopKeyEntry.get()) == 2:
```

Computer Science Coursework – Hand Tracking Application

```
            if self.stopKeyEntry.get()[0].upper() == "F" and
int(self.stopKeyEntry.get()[1]) >= 1 and int(self.stopKeyEntry.get()[1]) <= 9:
                self.stopKeyVar = self.stopKeyEntry.get()
                print(self.stopKeyVar)
                mainApp.stopKeyTextUpdate()
                self.stopKeyButton.configure(bg="WHITE")
                self.topKeyText.configure(fg="BLACK")
            else:
                self.stopKeyInputError()
        else:
            self.stopKeyInputError()

    def stopKeyInputError(self):
        self.stopKeyButton.configure(bg = "RED")
        self.topKeyText.configure(fg = "DARK RED")

    def close_windows(self):
        cameraFeed.xSF = (self.xSpeedSlider.get() + 10) / 10.0
        cameraFeed.ySF = (self.ySpeedSlider.get()  + 10) / 10.0
        cameraFeed.lowMode = self.lowModeVar.get()
        self.master.destroy()

cameraFeed = MainDevelopment.detection()

if __name__ == '__main__':
    root = tk.Tk()
    mainApp = mainClass(root)
    root.mainloop()
```

130

Examiner commentary

Question/Part: AO 2.2 Analysis

Marks: 10/10

The student's identification of the problem is clear and introduces the project well. Suitable stakeholders are described and relevant examples of how each may use the solution are given.

The computational methods section is excellent and demonstrates understanding of a range of appropriate concepts from section 2.2.2 of the H446 specification. The student justifies and relates the use of the methods to the features their solution will incorporate.

A thorough investigation is identified and carried out. There are concise summary sections of key points discovered after each interview and existing solutions are analysed well; the ideas to be carried through to the user requirements are clearly explained and justified.

The requirements section paints a picture of the solution – each requirement is explained and justifications given for its inclusion, including relevant hardware & software requirements. In the following Success Criteria section greater clarity of some of some of the less measurable requirements is added, defining what the student will be looking for as evidence they have met each point.

Several suitable limitations of the solution are explained. This section could be a little more thorough; however, each is justified and it is detailed enough that it does not affect the overall quality of the section. The teacher's mark of 10 is agreed for this section.

Question/Part: AO 3.1 Design

Marks: 14/15

A structure diagram has been attempted, showing an understanding of how the solution can be decomposed. This is further clarified on pages 25-27 & 35 as each bullet point outlined is then explained and justified.

Each interface is defined and its features linked clearly to the relevant user requirement. Usability features are well explained and justified.

The approach to sampling images clarifies how the student is planning to input the image data and then a series of key algorithms are explained in detail through pages 31- 33.

There are several diagrams used to explain some of the subroutines which help to make the processes understandable. These pages form a clear picture of the system as a whole.

Pages 37 and 38 list key variables and explain their use and any validation that may be required for input data.

Testing of the system is explained at the end of this section using descriptions alongside test tables where appropriate. Most of the testing described is developmental testing and the weakness of the Design section is in this area – only a few vague comments are given about stakeholder testing being carried out post-development.

The teacher's mark of 14 is reasonable for the work in this section, which could be followed by others with a working knowledge of the program and techniques planned for use.

Question/Part: AO 3.2 Developing the coded solution

Marks: 14/15

This is a very good section and the teacher's comments are accurate.

The student has provided evidence for each of the stages of development and thoroughly explained the processes, testing and errors encountered during this work. For each prototype both code and the GUI itself are shown and it is easy to link the two aspects together and to see how the work is changing.

The summaries at the end of each stage review the prototypes and link back to the success criteria; they relate each section of work to the problem breakdown.

The code itself is well structured, modular and makes good use of naming conventions. Comments become a little sparser as development continues, however there are enough to explain each of the key components.

Question/Part: AO 3.2 Testing to inform development

Marks: 9/10

There is evidence of some testing at each of the stage of development, and a number of errors are identified and corrected. The errors themselves are shown and updates to code in the form of before and after screenshots are included. Each of these is well explained and the corrections justified clearly.

The mark awarded to this section does best-fit its quality. The testing of some aspects of the code could have been documented more thoroughly, for example testing of the sliders. Although this is covered in the post-development section it would have added to the completeness of this section.

It could be noted that as the testing is not being shown in more traditional test tables each test/error could be identified more clearly using subheadings or emboldened text, as the development section is a very detailed piece of work.

Question/Part: AO 3.3 Testing to inform evaluation

Marks: 4/5

Usability of the system is tested and documented nicely, with robustness of data entry covered in several parts (push to stop keys, sliders). Some errors are identified and corrected. There are a number of other tests explained within this part of the work but evidence of those is not provided.

End users are involved in the testing at this stage and a review of their comments given on pg. 110, although this is a little brief.

The teacher's mark is agreed.

Question/Part: AO 3.3 Evaluation of solution

Marks: 14/15

This section is marked a little generously.

The student has provided some evidence of meeting each of the groups of success criteria, however better cross-referencing to the evidence in the project itself would have improved this section.

Although each of the criteria are discussed, it is a little lacking in detail and real evaluative comment as to the extent that the criteria are met or not and in some cases is simply stating what the images show.

Usability is evaluated more effectively, although it would have been nice to see some discussion of the end user comments linked in or referenced to reinforce the statements made.

The limitations and maintenance sections are well-written and address future developments and improvements suitably and in detail.

The small print

We'd like to know your view on the resources we produce. By clicking on the 'Like' or 'Dislike' button you can help us to ensure that our resources work for you. When the email template pops up please add additional comments if you wish and then just click 'Send'. Thank you.

Whether you already offer OCR qualifications, are new to OCR, or are considering switching from your current provider/awarding organisation, you can request more information by completing the Expression of Interest form which can be found here: www.ocr.org.uk/expression-of-interest

Looking for a resource?

There is now a quick and easy search tool to help find **free** resources for your qualification:

www.ocr.org.uk/i-want-to/find-resources/

www.ocr.org.uk/alevelreform

OCR Customer Contact Centre

General qualifications
Telephone 01223 553998
Facsimile 01223 552627
Email general.qualifications@ocr.org.uk

OCR is part of Cambridge Assessment, a department of the University of Cambridge. *For staff training purposes and as part of our quality assurance programme your call may be recorded or monitored.*

Cambridge Assessment

LLOYD'S REGISTER · LRQA
ISO 9001
UKAS MANAGEMENT SYSTEMS
001