Robotic Artist:Walter
G400 Computer Science, CS39440
Final Report

Author: Matthew Howard
Candidate Number: 150035512
User ID: Mah60
Supervisor Name: Helen Miles
Supervisor ID: Hem23
Last Modified: 04/05/2018
Version: 0.6
Status: Work In Progress(WIP)

Contents

0.1 Declaration

Declaration of originality

I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the regulations on Unacceptable Academic Practice from the University's Academic Quality and Records Office (AQRO) and the relevant sections of the current Student Handbook of the Department of Computer Science.
- In submitting this work I understand and agree to abide by the University's regulations governing these issues.

Name Matthew Howard

Date 03/05/2018

Consent to share this work

By including my name below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name Matthew Howard

Date 03/05/2018

0.2 Acknowledgements

I'd like to thank:

- Helen Miles - For monitoring the project and providing help when asked for,
- Dr.Pete Todd - For helping with setting up the plotter and helping when asked on the technical side of things. As well, for giving the idea to use the dithering algorithm as a style.
- Alun Jones - For helping with the OpenCV compiling error and suggesting to use python 2 as OpenCV was pre-installed on the device. As well, helping to try and track the segmentation error.

Chapter 1

Final Report

1.1 Introduction

This document is the final report for the project 'Robotic Artist: Walter'. The purpose of this document is to talk about how the project was run and executed, describing the problems that were encountered during the implementation, design and testing stages of the project. The final outcome at the end of the document will be evaluated and discussed as to how the project could have been improved overall. Then plans for future developments will also be discussed and how the project has been designed to make future development plans easier to be completed.

1.1.1 Project Description

The project that has been undertaken is to develop a 'Robotic Artist' from the Roland DXY-990 Pen Plotter. This is a device that has been collecting dust within the department building for a few years now and the project has been undertaken to bring the device back to life. The plotter and the other components for the artist can be seen in Figure 1.1.

The main requirements of this project were to be able to convert the Roland DXY-990 Pen plotter to be able to plot out artistic drawings of peoples faces. So the next stage of the project was to understand what was required to be able to develop such an artist and what form of method would be undertaken to do this. However, before this, the report will discuss how the project was maintained and developed as time went on.

Figure 1.1: Roland DXY-990 pen plotter - This figure demonstrates the Roland DXY-990 Pen Plotter and the other components that are being used for the robotic artists. The picture was taken during the development of creating the initial plotting algorithm.

1.2 Process/Methodology

The project's development needs a process or methodology to help with the development process and provide an efficient workflow. For this project, the methodologies that were mainly used were Feature Driven Development(FDD)[5] alongside a Kanban board[6]. The FDD gave a structure to the development process as features/requirements were developed to give an optimal success chance of a working product. The Kanban board would take those features into consideration and help keep track of deadlines for individual features and the structures that they will need to be developed within. These methodologies are expanded upon below.

- Feature Driven Development(FDD)[5]- This process involves taking the project that was given and splitting it into key features. Then from those features, requirements are made and planned out on how to develop each one of them.

  FDD is an agile methodology that provides development of multiple features at a time, depending on how large the workforce is. The method follows a routine of developing the overall model => Build a feature list => Plan by Feature and then design the feature and implement it.

- Kanban Board[6] - This process helps to build a timeline for controlling the process of the project. The general idea of the Kanban Board is to create a workflow that provides a suitable structure to maintain the process of project development. To give the optimal workflow the Kanban Board makes multiple columns to say what stage of the job of designing and implementing the code is at. The jobs can be developed from the FDD.

  The different stages can be a TODO, InProgress, Review and Complete stage. The TODO section has all the jobs that are still required to be done. These Jobs have a title, description, priority and a due date. They will need to be planned out correctly to get the most optimal output from this methodology. Then these jobs are moved along the columns when they enter different stages of their design, they can also move back when required. For example, if they are reviewed and still need improvements, it will either be moved back to the InProgress stage or the TODO stage to be looked at later when the modifications are required.

These two methodologies helped to improve the process and workflow of the project. At the beginning of the development, the individual features were taken into account, these were dependent on the equipment and the ideas that were given to the project at the initial stages of development. When designing each feature, they were written down onto paper and then developed further to show how the features worked. The construction of the features can be seen in Figure 1.2. After this was complete, the individual features were pushed towards the Kanban board to generate a suitable workflow to reduce the time it takes to develop the individual features. This will require looking at each feature and designing the process based on what development is required to be done to allow the features to evolve at different stages of the project. The final output of these features can be seen in the requirement specifications[1].

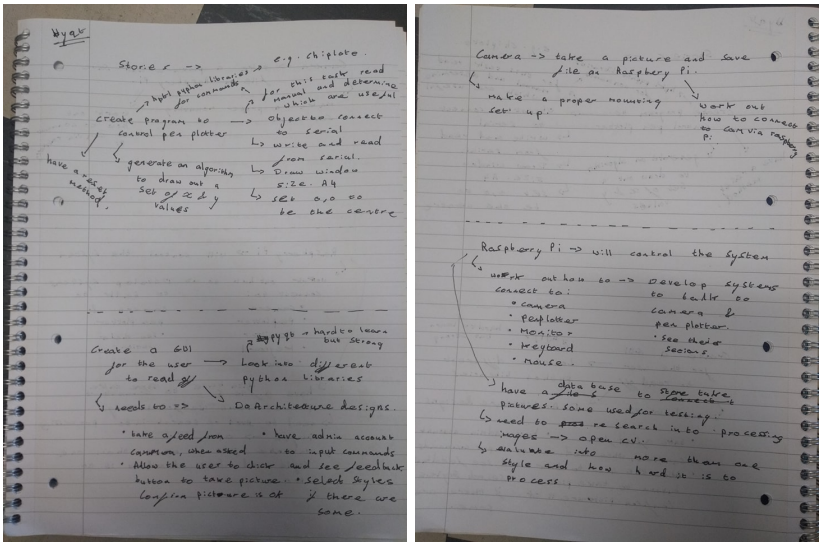


Figure 1.2: Designing of Requirements - This figure demonstrates the development of the individual features on paper. The process was done by writing down the features and then expanding on them depending on what is needed to be completed to achieve that feature.

The Kanban Board was developed using the site Trello[7], to help develop and maintain an efficient workflow in the project. The Kanban Board has four key columns TODO, InProgress, Review and Complete as mentioned in the description of the Kanban Board and also followed the same process. An example of this Kanban Board can be seen in Figure 1.3.

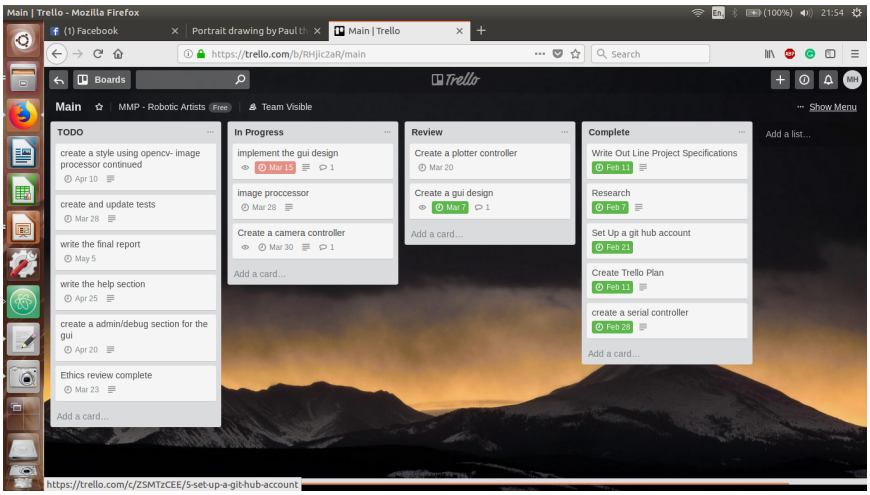


Figure 1.3: KanBan Board - This figure demonstrates the Kanban board that was used during the project. This figure also shows the TODO, InProgress, Review and Complete stages in use. The stage that this version of the Kanban board is before the mid-project demonstration, around the 19/03/18. The Kanban was developed with the help of Trello[7].

This process was then monitored by the project supervisor in meetings that occurred every week. The meetings provided a breaking point in the workflow that allowed the project to be reviewed on what has been done in the last week, what problems occurred, whether they were fixed and then what is needed to be done next. For the questions that were asked, appropriate responses will occur depending on the event. This could be another feature that is needed to be completed or developed to progress the current feature or making alterations to the development plan to be able to complete the process in an optimal time period.

The overall workflow of this project was to take a look at one key feature or requirement at a time. This involved researching, designing and then implementing that feature. Until another feature was required to be developed to make the connections between the individual features and their current goals.

Then when modifications were made to the code there was version control, this used GitHub[4] version history to detail and record changes towards the code and documentation. This process is expanded upon within the Design Specification[2]. However, the general idea is with GitHub there are individual versions of the repository commits for alterations of the project and the version history provides a description and date of when modifications were made to the code and documentation.

These methodologies then helped to research and analyze the project. As when developing the individual requirements that can be seen in the Requirement Specifications[1] there was research and analysis towards how that feature would be developed. Then Kanban provided the workflow that was needed to develop the project alongside those chosen features.

1.3 Background and Analysis

The 'Paul the Robot' project[9] was what inspired the initial ideas of the design and development of this project. The idea is to be able to develop a robot that can take a captured image of someone's face and create an artistic drawing from that picture. Figure 1.4 demonstrates what 'Paul' was capable of drawing. This then leads to the idea of the project that has been developed and completed over the past few months.

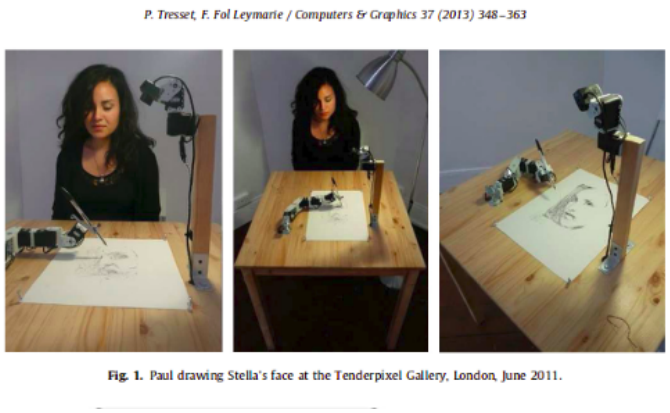


Figure 1.4: Paul The Robot - This figure is taken from the 'Paul the robot' article[9]. The article talks about a project that developed an abstract robotic artist, meaning the robot could draw out someone's face without knowing any of their facial features.

There are other projects that have inspired and provided fundamental research towards the project. The Joto[10] creative drawer can be seen as one of those devices, as it takes a drawing that someone has done on a mobile device, such as a phone and then draws it out. The layout of this product can be seen to have similarities with the pen plotter, as it has the bar that stretches across the portrait area. Then the bar has a board pen that is moved around to draw out the sketches that were drawn on the mobile device. This idea gave a possible path that could be taken with the robotic artist and how the plotter could go about drawing artistic styles that are generated from the image processing stages.

Another device that helped the development of the project was the Cloud Painter[11]. This device has been in the process of constant development over the last twelve years. This robot uses artificial intelligence, deep learning and computer creativity to generate art from pictures of people, locations or other artist portraits. The cloud painter takes ideas from previous portraits that it has drawn and uses those projects to learn what would be the most optimal style to try drawing the current portrait in. This would be from what previous artwork has succeeded. This process is done through neural networks and machine learning to work out what styles pass as a successful portrait. This helped research on the project, as it demonstrates that to create a working artistic style the algorithm will need to be applied to the pictures. This will transform the pictures in an artistic portrait. This could be done through methods of machine learning or generated algorithms to apply to the process.

From the research we can gain that the overall idea of the project is to use image processing to generate an artistic style algorithm that is able to be plotted out on the Roland DXY-990 plotter. This will involve generating an ideal method of looking into an image and then analyzing them in

a way that enables the image to be transformed. One way of doing this would involve looking into the library 'OpenCV' that has been designed to do image processing and capture techniques. This library and the components that were given to the project's development, helped to mould the start of the development of the project.

The first part that influenced the design of the project was 'OpenCV'[12]. This library is used mainly in two different languages Python or C. There has recently been a branch out to Java. However, due to this the library for OpenCV in Java is currently not well documented. The development of OpenCV was to make it so that image processing techniques were available such as; looking at pixels, Gaussian blur, and canny edge detection. These techniques are very useful when trying to process individual images as you can find patterns and features in an image that could be used to develop an artistic algorithm. Another use of OpenCV is that it simplifies the technical work towards creating a working camera feed and image capturing techniques. This, therefore, makes OpenCV a valuable resource to be used within the project and was of high importance when designing the product.

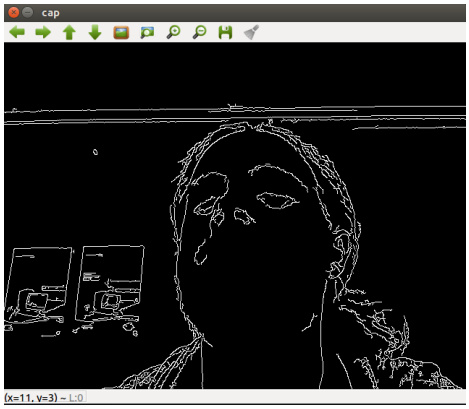


Figure 1.5: Canny Edge - OpenCV - Demonstrates OpenCv[12] being used to apply the canny edge detection algorithm against an image. The image was taken using OpenCv to capture a frame with the video camera.

The next part that helped to make a decision on how the projects final outcome came about was the components that were given to the project to help achieve the final goal. These were the use of a Raspberry Pi 3, a web camera and a monitor with a mouse and keyboard. These key components are important as they defined the structure of the robot given and how the components communicated with each other.

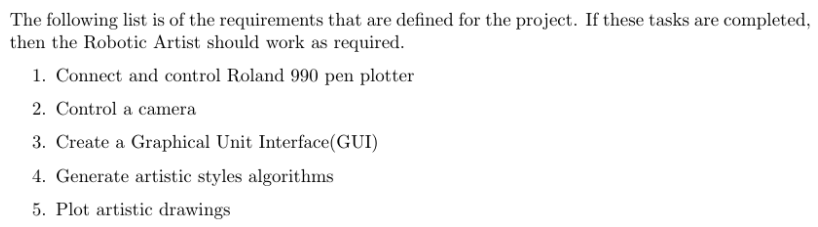
The initial ideas towards the components are expanded upon below:

- The Raspberry Pi will act as a controller that would help make the overall decisions on what will be done and then communicate to the individual devices to give an output that was generated from the input from the monitor.
- The web camera would either be a standard camera or pi camera as it is designed to work with the Raspberry Pi. This camera will capture multiple frames a second to display on the monitor in a form of graphical user interface(GUI) and will capture an image when commanded to.
- The monitor will be there to display the video capture that is required for the robot and then communicate any input from the user. The monitor will also display a GUI, which will control

the events that are occurring within the robot.

This set up was given to the project for development based on the structure of the code and gave the project straightforward directions to take. The overall robot design was not going to be like 'Paul the robot'[9], however, it would take the general concepts and work around the equipment that was provided to the project. For example, Paul had a robotic arm that was designed to copy the drawing styles of a human hand but this cannot be done with a pen plotter. Other concepts that were taken into account at the time of the designing the robot is whether to use a Raspberry Pi or an Arduino that would have been programmed in a form of C or another form of computer hardware. The Pi was chosen because the pi was installed with USB ports to connect to other devices and provided a connection to the wifi.

Based upon expanding the original ideas, the project will be to create an image processor that would use artistic style algorithms to generate an image that could then be translated into commands to be plotted out on the pen plotter. This led to the development of the following features/requirements that can be seen in the requirement specifications[1].



The following list is of the requirements that are defined for the project. If these tasks are completed, then the Robotic Artist should work as required.

1. Connect and control Roland 990 pen plotter
2. Control a camera
3. Create a Graphical Unit Interface(GUI)
4. Generate artistic styles algorithms
5. Plot artistic drawings

Figure 1.6: List of Requirements - This is a snippet of text from the Requirement Specifications[1] that demonstrates the key requirements for this project. The requirements are expanded upon within the document itself.

The requirements in Figure 1.6 are the key features that will be needed to be implemented in the final product in order to achieve the optimal outcome. The overall idea was to make a graphical user interface(GUI) with four key stages that would control the events that were occurring within the robot. These are video capture, picture acceptance, style selection and style confirmation pages. This would allow the user to take and confirm the captured image is appropriate for use. Then they would be able to select from a range of artistic styles, to transform their picture to. This would then let the user see the final output before plotting the style.

The next stage of the project was to do research into the different methods to complete the key requirement and then to decide on which method would give the optimal output.

1.3.1 Research

The research for this project was not all done at the start but was rather done throughout the project and there were multiple sections of research committed. This is because as the individual features were looked at the research was done alongside them. They were done in such an order that meant the research for the feature was done before the implementation of the code and as well at the same time future development on the feature was taken into account.

However, the project had some straightforward decisions such as choosing to code the project in Python. This was because python was one of the main languages that were pre-installed onto the raspberry pi and when setting up the serial communications between the pi and the plotter, python was the language used to get the communication working. Another reason that made the choice easier is that python is an object-oriented language. This would make it easier to design a code structure that would support the final outcome and make future development easier. As well, there are multiple libraries that have the functions that are required to program this project within python. An example of these libraries can be seen below.

- pySerial[14] - This allows for the communication between the pi and the pen plotter.
- OpenCv[12] - This supports the image processing algorithms. Allowing the robot to look into individual pixels, detect edges and other features within the image.

Another feature that was chosen at the beginning of the project was to use serial instead of parallel. The reasoning for this is simply because the university already had serial to USB adapters set up for use, however, they would need to get the resources to use parallel to USB adapters. As well, the serial is easier to communicate between the devices, if slower, as data is transmitted bit by bit. Parallel communicates in 8 bits making the process faster but harder to make a working communication between devices.

The next stage of the research was working out how the pen plotter worked and how to plot out the artistic style algorithms. This was done by looking at the 'Roland DXY-885 manual'[13] which contained the documentation of the individual commands used within the pen plotter. These are written in HPGL, this is a form of G-code and is used in printers and older machines to perform fast and simple processes from two characters and a list of arguments.

The rest of the research was done when designing the individual algorithms and functions of the code. In which the developer of the project, looked at the different features of the project and how they would be developed. This then generated a priority of what needed to be developed first to enable the next stage or the initialization of another feature to be started. An expansion of further research for the project can be seen in the Technical Work chapter of this document. This will go through how the research for the python library that would be used to develop the graphical user interface(GUI).

1.4 Technical Work

This chapter of the report is going to go over the technical work that was done over the time period of the project. The chapter is split into multiple sections that discuss the different features and how they were designed and implemented to get to the final stage of the product. At the end of this chapter, there will be a discussion on how the final stage occurred and how the code was tested to see if it was working as expected.

The individual features can be seen in Figure 1.6 and from these features/requirements, the development of the project was split up depending on what was required to be done first and thereby created the iterations through which the code was developed. The methodology towards this can be seen in the 'Process/Methodology' chapter.

1.4.1 Final Design

Before talking about the individual iterations that the project went through it would be worth discussing the final design of the product. This will give a better understanding of the individual classes that were developed during the project and how they were developed.

The final design supports the key requirements that were given within the requirement specifications[1]. This is because it has been designed to support future development and to be able to control each feature that was given. The overall design can be seen from the UML class diagram in Figure 2.2 and is expanded upon within the Design Specifications[2]. This document talks about the structure of the product using UML diagrams, version control and the design of the GUI.

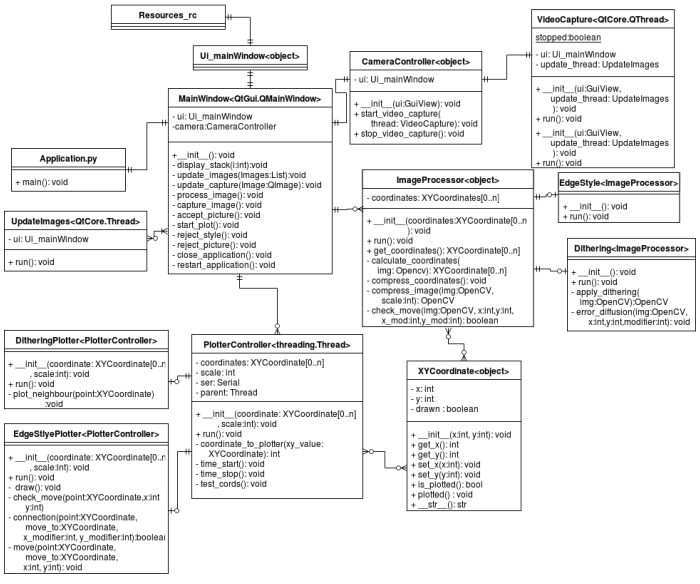


Figure 1.7: UML Class Diagram - This figure demonstrates the overall structure of the classes. The UML can be found within the Design Specifications/UML Diagrams[2].

The class diagram has four key sections. These are:

- GUI - The first section includes the classes for the MainWindow and UpdateImages. This set could create and control the GUI, this then was used to make events occur within the robot. Depending on actions done within the GUI. The code was developed using the Python library PYQT[3] and was developed using QT Designer, which is the GUI_view within the class diagram. The section also has a QThread called UpdateImages, that will send a signal to the main GUI thread to update all the non-static images within the GUI.
- Camera Control - This section of the code talks to the camera and gets the input that creates a video capture from the web camera. Then when the QThread that does this is stopped using the static variable 'stopped' inside video_capture.py, the camera saves the frame it was in and that will be the captured image.
- Image Processors - This section of the code has a parent class that has the generic methods that are used in the image processing styles. These are what apply the artist style algorithms to the captured image from the camera controller. Each style inherits the ImageProcessor, then takes the methods and uses them to support the style algorithm. This could be classes like calculate_coordinates() to work out where each black pixel is and mark that as a coordinate for the plotter to use later on.
- Plotter Controller - This section of the code has a parent class that contains all of the generic plotting methods. For example, the coordinate_to_plotter() method which transforms coordinate values to plotter coordinates. The next parts of this section are the individual style plotter controllers, which inherit the plotter controller. These classes have the code inside that will be used to apply the plotting algorithm that will be used to plot the coordinates given from the ImageProcessors.

The design has developed a total of two styles the Edge-Style and Dithering style. These styles apply the algorithms and plot the portraits that can be seen in Figure 1.8. The sequences of the dithering algorithm can be seen within the Design Requirement in the 'UML Diagrams/Sequence Diagrams' section. This talks about the sequence that is taken for the user of the robot to plot out the dithering algorithm. After this the project was completed, however, there was one major error that was not fixed by the end of the project and was found in testing. As seen in the Testing Records[4].

Figure 1.8: Dithering and Edge-style example - These images are examples of the Dithering and Edge-style, when they have been plotted using the Roland DXY-999 plotter.

This error was that a segmentation error occurred on the video capture page. The error is not constant and randomly occurs but only when used on the Raspberry PI as can be seen in Test 2 of the Test Records[4]. The understanding that came about was that the error was being caused by the PYQT source code and when too many processes were done at the same time with the video capture. Something was accessed that wasn't there. There were tests that were done to check if the fault was with the code or if the Raspberry PI was running out of memory. However, this was not the case as commands such as 'free-m', 'top', 'dmesg' and 'strace' were used to check the memory and what was running in the background. This method of debugging found that the Raspberry PI was not running out of memory. So during this, the code that was developed for this project was also debugged to try and find the location of the error.

The segmentation error was either coming from the source code of the PYQT library or an error within the code that the developer could not find. The next stage of the document talks about how the individual features of the project were developed in iterations. The iterations were based on the requirements and the stages that were needed to be done to proceed with requirements that were being developed at the time.

1.4.2 Plotter Set up - Iteration One

The first iteration was to get the Roland DXY-990 Pen Plotter working. This would be to make a stable connection between the plotter and the Raspberry Pi. The first stage of this development was to create a serial connection between the devices, this was done by looking at the DXY-885 manual[13]. The serial connection was made by changing the DIP switches that are shown on page 4-3 to 4-10. The settings that were set to make the connection for the serial were; Baud-rate = 9600, Stopbits = 1 and Parity = Even. Then this allowed for communication between the Raspberry Pi and Serial.

The next stage to set up a suitable connection was to be able to send commands back and forth from the plotter and the Raspberry Pi. This was developed using the library pySerial[14] and setting the settings of the serial to be the same as the serial. Then the connection was then tested by sending the command 'PA 500 500;', which can be seen on page 6-32 of the manual. This then moved the plotters pen to the coordinates 500 500, which are the x and y values of a graph. However, for each x,y coordinate is proportional to 40 plotter coordinates. This allows for more detail when plotting and allows for better scaling of plots. There was a set of code that was then written to send commands that are written from the robot. This can be seen in Figure 1.9.

```
def write(self, command):
    """
    Summary => handles all messages being sent to the serial.

    Description => Will write to the serial given to this class and then
        will read from the serial to see if there is a returned message.
        If not will return none.

    args =>
        command => this will be a string of the command you want
            to send to the serial, connected to this object.
    return => string, if there is any response from called command. If not
        the response will be None
    """
    # print('write')
    # writes to the serial and converts message to bytes
    self.ser.write(command)  # encode('utf-8')
    # will make the program sleep for 100 ms to see if there is a response
    sleep(0.25)
    # if command is PA wait until plotter is in correct location
    self.wait(command)
    # returns a response if one exists
    # print(command)
    response = self.read_all()
    # print(response + "-----")
    return response
```

Figure 1.9: SerialController - write() method - The write(), writes to the serial by using the command Serial.write(command). The command is handled through the arguments after this command has been written the serial controller checks if there is any response by using the read_all method.

A further development that was done on setting up communications was to be able to read responses from the plotter. This involved using the method that use's Serial.read(), to read each incoming characters from the plotter until there is no message left. This can be seen in Figure 1.10.

```
# print('read')
buffer = ""
# check if there are any expected input from the serial
while self.ser.in_waiting:
    # print('true')
    # if there is incomming input, throw into loop and
    # read the loop
    char = self.ser.read()
    # turn byte into ascii
    # char = char.decode("utf-8") - use if python3
    # when '\r' appears end reading process
    if char == '\r':
        break
    else:
        # add to buffer
        # print(char)
        buffer += char
# print(buffer)
return buffer
```

Figure 1.10: SerialController - read_all() method -The read_all(), checks if the serial is waiting for a response. By using the serial.in_waiting() method. This then loops around until this value is false, and adds any characters that are read to a string which is returned at the end of the method.

These two sets of code then allowed for stable communication between the Serial and Raspberry Pi and were put into the SerialController class. That is an object that allows for communication with serial, this class contains the methods that write and read commands to the serial. As seen in the Figures 1.9 and 1.10. The next part of the project was to gain a full understanding of how the plotter worked, this was done by reading the manual[13].

What was learned from the manual was that the individual commands; 'PA x y;' - that told the plotter were to move the pen to and 'PU;' and 'PD; which told the pen to be brought up and down. These were the main commands that were used throughout the project. There was also the Scaling sizes that can be seen on page 5-5 of the manual. Which changed the size of the coordinates depending on what pages you were using.

At this stage of the project, it was decided to develop the code in Python 3 as it contains the most up to date versions of python code and the libraries.

The communication of the project was developed and implemented by the end of this stage. This allowed for stable communication between devices using serial. The next stage of the project was to set up the GUI, which would control the events that will occur within the project.

1.4.3 GUI Development - Iteration Two

The next stage of development is to create a graphical user interface (GUI) that will be used to control the events that occur at runtime. The start of this development was to research what GUI libraries within python would best suit the project. The library that was used at the end of this research was PYQT[3]. This is because it gave a large support of widgets and other graphical tools to help develop the overall design of the project. The library also supports Python 2 and 3, this means there is a wider range of development that can be done with the library. PYQT is also an open source library that has a piece of software called 'QT Designer'[2] that was used to help develop the skeleton of the GUI. Once knowing the library that the GUI was being developed within, a set of tutorials by Sentdex[15] helped to learn how to apply actions and give basic control over the GUI that was developed.

The development of the GUI started by drawing the designs and templates of the individual pages on paper. The templates were designed from the requirements within the requirement specifications and would take the project through the actions required to make the planned Robotic Artist. These designs showed the skeleton of the GUI and how it would function overall. The development of the GUI templates can be seen in Figure 1.11 These designs were altered as the project went on, however, the overall outcome stayed the same.

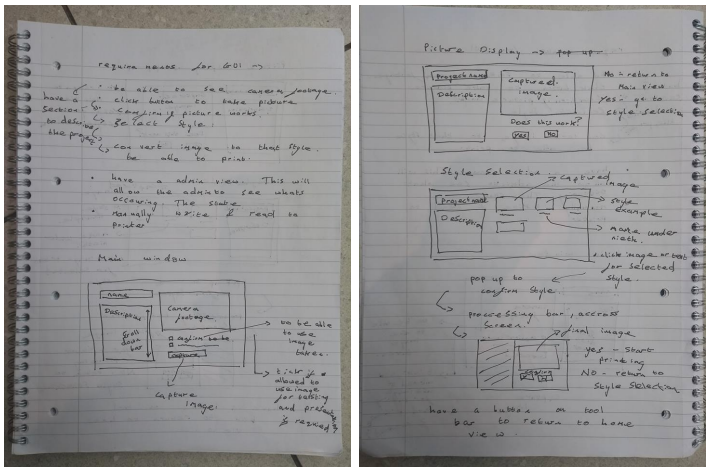


Figure 1.11: GUI development - This figure demonstrates examples of how the GUI was developed. These images talk about the requirements of the GUI and go over the video capture, picture acceptance, style selection and style confirmation pages.

After the GUI was development, the next stage was to implement that skeleton. This was done by using 'QT Designer'[2], where you can drag and drop the components of your GUI onto the window and place them in the locations given from the designing process. QT Designer helped in the GUI as time saved was as the whole GUI didn't need to be manually coded. The only part that was coded was the actions that took place within the GUI. This can be events such as exiting the application to starting the process to process to plot out the coordinates calculated within the image processor. The set of code to develop this event can be seen in Figure 1.12.

```
    # creating exit pop-up message
    choice = QMessageBox.question(self, 'Leaving Application',
                                        "Leave Application?",
                                        QMessageBox.Yes |
                                        QMessageBox.No)
    # do actions depending on response
    if choice == QMessageBox.Yes:
        # close the camera feed, if it is still running
        self.camera.stop_video_capture()
        # reset each of the images to blank, when closed
        cv2.imwrite("Images/takenPicture.jpg", cv2.imread(
                                                    "Images/blank.jpg"))
        cv2.imwrite("Images/processedImage.jpg", cv2.imread(
                                                    "Images/blank.jpg"))
        print("Leaving Application")
        sys.exit()
    else:
        pass
```

Figure 1.12: Exit Application - This figure shows the code that is used to exit the application. The code use's PYQT objects to ask first if the user is sure, using a QMessageBox and if the answer is yes. The line of code sys.exit() will be called after doing shutting down all required functions in the background.

Once the skeleton of the GUI was developed inside 'QT Designer' then the design was exported to python 3. This then leads to the implementation of the GUI into the code, as the MainWindow class was developed to control the events occurring within the GUI. This class built the GUI and displayed it on the monitor. This involved writing code that would attach events to different components of the GUI, such as buttons. This included developing code that would check that the checkboxes or other rule defining states within the GUI are in the correct state. Examples of the code that was implemented to control the events that occurred within the GUI can be seen in the Figures 1.14.

```
print("plotting")
# create the plotter conroller (coordinates , scale)
# plot coords for dithering style
self.plotters = [
                DitheringPlotter(
                                self.processors[0].get_coordinates(),
                                1),
                EdgeStylePlotter(
                                self.processors[1].get_coordinates()
                                , 1)
                ]
# call run, when ready to throw through the process.
for i, plotter in enumerate(self.plotters):
    # check if specified style is ticked
    if self.style_check_boxes[i].isChecked():
        # need to check if thread is running, so doesn't start more
        # than one thread from pressing button muiltiple times.
        if not plotter.is_alive():
            plotter.start()
```

Figure 1.13: Plotter Checking - This code belongs to the MainWindow.start_plot() method. The structure of the code will loop through each plotter controller style, during this process it checks if the respective checkbox is checked or not and whether the plotter is already or not.

When the GUI was designed and implemented to the basic version, in which it could be updated when different stages of the project were developed later on. The next stages were to create an algorithm that will plot out the coordinates that will be given from the Image Processing stage.

1.4.4 Plotter Controller - Iteration Three

This stage was about creating a PlotterController, this class will take a list of x and y coordinates and use an algorithm to plot the coordinates in a specific way. The first class that was developed to generate a generic structure that will be used in the project. This class contains three variables x, y and drawn. The drawn value is a boolean that tells the program whether that specific coordinate has been plotted yet. The following stages of this iteration were to create an algorithm that would plot the coordinates that were given to it.

The first algorithm that was developed involved generating lines within a form of a linked list. This would allow the plotter to follow each individual line and make plotting time efficient. The idea was to use a recursive loop that would search through each coordinate and connect the neighbouring coordinates. This would also create a new line if there was a split point in the line. From this, there would be a map generated from the lines that are needed to be followed to plot the desired style. However, this idea was scrapped, as the plotting algorithm was deemed to be too complex and no code was implemented for this plotting algorithm. Another reason that the algorithm was rejected was that it underestimated the size of a pixel within the image and this made it harder to calculate lines from the image.

The next algorithm that was generated was of a much simpler variety. This was to take each coordinate and check if they have a neighbor. If a neighbor is found the plotter will draw to that point and then return to the original location. After the coordinate had drawn to each neighbor that is found the XYCoordinate drawn value will be set to true so that this point will not be redrawn again.

The implementation of this code can be seen in Figure 1.14. The class has a method that would transform the XYCoordinate values to plotter values. Which are of a ratio of one to four plotter values and this value can be scaled depending on the scale that was given at the initialization of the object.

```
self.ser.write('SP 1;')
for point in self.coordinates:
    # place pen at this points location
    x = self.coordinate_to_plotter(point.get_x())
    y = self.coordinate_to_plotter(point.get_y())
    str_command = "PA {} {};".format(x, y)
    self.ser.write(str_command)
    # put pen down
    self.ser.write('PD;')

    # check if there is a connection in each direction,
    # think of it like a compass North, North-East etc.
    self.check_move(point, x, y)
    # mark this point as drawn
    point.plotted()
```

Figure 1.14: Edge style - Using XYCoordinate class - This code belongs to the EdgeStylePlotter. The stages use the XYCoordinate to get the x and y location, then converts them to plotter values. Then the point is marked as drawn, to make sure the point is not plotted again.

The design and implementation of the serial controller has now been completed, this will allow for tests on different coordinates that are developed from image processing techniques. The next stage is to develop the camera and video capture that will be used to capture pictures to be processed into the artistic styles.

1.4.5 Camera Controller - Iteration Four

The general idea behind the development of the camera was to be able to capture pictures and a video feed of what was in front of the web camera. This would require the camera to continuously transmit frames from the camera to the GUI and to be updated on the video capture page. As well, to capture a picture to use for the image processing stages later on in the project.

This was a straightforward design to be implemented. OpenCV[12] was the main source of getting the camera to work as you can use a VideoCapture object that will make a connection to the camera connect to the device being used. This then allowed for image capturing when the method read(), which returns the current frame and a boolean, which tells you if the capture was successful or not. This could then could be used to capture a frame that would be sent to the GUI, this then can be used to capture a picture through saving the frame that the camera was on at the time.

The classes CameraController and VideoCapture were developed to do this. The Camera controller would be used to activate and deactivate the VideoCapture class. This is a thread that will constantly send frames from the camera to the GUI when active. Furthermore, when the thread is stopped through the CameraController a frame is captured, that will be used later on for applying artistic algorithms to.

There was a major problem that occurred during the implementation of the code. This was the error 'VIDEOIO ERROR: V4L2: Pixel format of the incoming image is unsupported by OpenCV'. This was caused by OpenCV not being able to find the camera that was trying to be allocated to the video capture object. This problem was solved by recreating the environment in another python module and then gaining a full understanding of how OpenCV worked. The code that was generating the error was found using this method, as the problem was not how the code was implemented but within the CameraController class, the video capture was already created and connected to an object within it. Therefore when trying to connect to the camera in the VideoCamera thread there was no free camera to connect to.

After that problem was solved that camera worked currently and the camera was implemented into the structure of the code. The next stage was to create artistic style algorithms to be plotted.

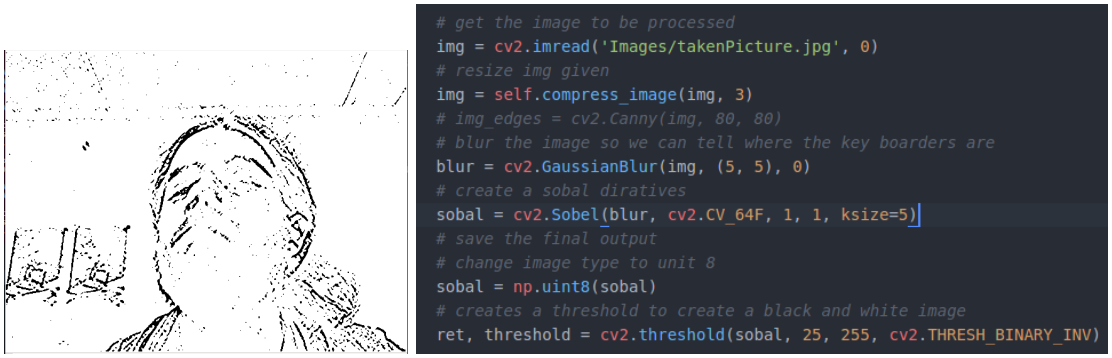
1.4.6 Artistic Style Algorithms - Iteration Five

This iteration was to develop an artistic algorithm that could be later plotted out on the pen plotter. The first idea of a style algorithm can be seen in what is now called the edge style, this involves detecting the edges within the image taken and plotting them to create a facial structure from the borders of the face. This technique was a collection of different OpenCV techniques.

The techniques used for this are;

- Gaussian blur[18] - This is a blurring technique, that applies a kernel across the image and alters the form of the pixels. To give a blurring effect.
- Edge detection techniques - Sobal[18] and Canny Edge[18] - These techniques looks for sharp changes in the pixel colour to detect where the edges of the images are.

The algorithm was to gaussian blur against the image to reduce the amount of noise that appears while using the edge detection. Then applying either the sobal or canny edge effect against the image. This then gave the effect that can be seen in Figure 1.15. The code for doing this effect is very simple as the majority of the functionality is from the library OpenCV[12]. A snippet of the code used can be seen in the Figure 1.15.



```
# get the image to be processed
img = cv2.imread('Images/takenPicture.jpg', 0)
# resize img given
img = self.compress_image(img, 3)
# img_edges = cv2.Canny(img, 80, 80)
# blur the image so we can tell where the key boarders are
blur = cv2.GaussianBlur(img, (5, 5), 0)
# create a sobal diratives
sobal = cv2.Sobel(blur, cv2.CV_64F, 1, 1, ksize=5)
# save the final output
# change image type to unit 8
sobal = np.uint8(sobal)
# creates a threshold to create a black and white image
ret, threshold = cv2.threshold(sobal, 25, 255, cv2.THRESH_BINARY_INV)
```

Figure 1.15: Edge Processor Example - On the left is an example of what is produced by the code on the right. The code applies the Gaussian blur and Sobal effects. When processing pictures with this effect the surroundings must be considered, for example, the lighting.

The design and implementation for the artistic style algorithm have been developed to detect edges within the captured image and gather coordinates from that. This style was generated from the idea of mainly using OpenCV to create the individual algorithms, however, the result was the edge style. Which was not visually pleasing, this, therefore, meant the style was going to be scrapped or altered in the later generations to make a style that more appealing to the user. However, a problem occurred that delayed this. The problem was that device was that the Raspberry PI 3 did not have OpenCV installed for python 3 and when compiled an error occurred that stopped OpenCV from being installed. The decision was made that the project would proceed faster by changing the code to Python 2, this was because the pi had OpenCV pre-installed for python 2.

1.4.7 Class Structure Modifications- Iteration Six

This iteration was about changing the code to python 2. The decision was made because the Raspberry PI did not have OpenCV for python 3 pre-installed. Then when OpenCV was attempted to be installed onto the device, there was a compile error halting the process. Therefore, a decision was made to change the code to Python 2 because the device had OpenCV pre-installed for that version of python.

The first change was to downgrade the version of the GUI from PYQT 5 to 4. This allowed the interface to work within python 2 correctly. However, there were some alterations to the code that also allowed for the code to work within python 2. The first was to change the method of threading when making an alteration to the GUI at runtime. This was to change the thread to a QThread, this could be used to send signals from within the thread and called the methods that were within the main GUI thread. This was applied as there was an error with the normal threading that did not allow for modification of the GUI from external sources. This meant a signal must be passed to the main GUI thread that would call a method that would update the images required. This could be seen in Figure 1.16, the classes that this modification was made to was the VideoCapture and UpdateImages classes.

```
# set checked_boxes to false
self.ui.check_process.setChecked(False)
# creating a thread to update images when complete.
# update images in gui
update = UpdateImages(self.ui)
# connect signal
self.connect(update, QtCore.SIGNAL("update_images(PyQt_PyObject)"),
             self.update_images)
```

```
images = ["Images/takenPicture.jpg", "Images/processedImage.jpg"]
self.emit(QtCore.SIGNAL("update_images(PyQt_PyObject)"), images)
```

Figure 1.16: Signals - This figure goes over the new code which is being used to update images in the GUI. The image on to displays the set up for the signals in the MainWindow.display_stack() method and the bottom image displays the signal being sent to call the method MainWindow.update_images().

Another alteration that was made to the code, was changing the Serial Controller. This was that the method to write and read the code was changed. Because within python 2, the strings send through serial were automatically formatted to be sent through serial. This meant for the device to work within python 2 the formatting was needed to be removed from the individual strings received and sent. This can be seen in Figure 1.17. The modification was to make code that would reduce lag within the plotter and make sure each point is plotted in the correct location. This was done by sending signals to the plotter and reading the response until the response matched the location that plotter was meant to move to. This code can be seen in Figure 1.17.

```
if command_list[0] == 'PA':
    while True:
        # get coords to wait for plotter to reach sent coords
        self.ser.write("OC;")  # .encode('utf-8') - use for python3
        # read response
        sleep(0.1)

        response = self.read_all()
        # print(response + " ---- " + command)
        # check response to
        if response is not '':
            new_response = self.remove_semi(response)
            # split by space
            response_list = new_response.split(',')
            # if command and respnse match continue plot
            if float(response_list[0]) == float(
                    command_list[1]) and float(
                    response_list[1]) == float(command_list[2]):
                break
```

Figure 1.17: SerialControl wait() - This code goes through a loop that continuously sends the 'OC;' command, which returns the coordinates of the pen and its status. This can be found in the Roland Manual on page 6-25[13]. This code also uses the remove_semi() to take a semicolon at the end of a response to make reading the current coordinates easier. Once the expected coordinates and the current coordinates of the pen match the loop is broken and the plotting process continues.

The next major change that was committed to the code to make future development of the project easier to write was changing the ImagePocessor and PlotterController to parent classes. This meant the main methods that were used for those sections of the code would be kept in the parent class and then there were child classes to develop and plot multiple styles. The current plotting and processing algorithms were redesigned to become the Edge-style. Each style that will be developed will be required to have a plotter and image processor. The reason this was implemented into the code was that inheritance would help support future development and multiple styles. This is because the main code that would be required to develop a style would be plotter and image processor, this would make the only alteration to code would be within the MainWindow to add the events to the GUI.

After this, the code was successfully translated to python 2. This allowed the image processing to work correctly on the Raspberry Pi and lead the project one step closer to a Robotic Artist. The next stage was to develop an artistic algorithm that looked visually appealing.

1.4.8 Artistic Style Algorithms Continued- Iteration Seven

The development of the next artistic algorithm was to apply the dithering algorithim[19]. This would make an image that could be seen with only one colour and dithering was used during the 90's to help print pictures with one colour. An example of dithering can be seen in Figure 1.18. The way dithering works are that there are two stages, these are pixel segregation which is followed by error diffusion.

Figure 1.18: Dithering example - This figure demonstrates a example of the Floyd-Steinberg Dithering algorithm[19].

- pixel segregation - This was moving from left to write of the image and changing the colour of the image to black or white depending what end of the colour palate they were at, the split point was around the 127. This would then cause error defusion.
- error defusion - This is spreading the values of the pixel to the neighboring pixels. This means if the value was 65 and was segregated to be black, which has a value of 0. The remaining 65 values would be defused to the neighboring pixels. Then if the value was closer to 255, then the error would be defused to surrounding pixels. The error would be calculated by subtracting the difference between the current pixel and 255.

The defusion method that was used to create this artistic algorithm was Floyd-Steinberg dithering alogrithm[19]. This worked from left to right, like other dithering algorithms but handled the error diffusion effect differently. The diffusion would be done in a two-dimensional plane, this means instead of just defusing the value to the right, to also diffuse the value downwards. This was done by splitting the value into sixteenths and separating the values depending on the value of the kernel around the current pixel being looked at. The error defusion can be seen in Figure 1.19.

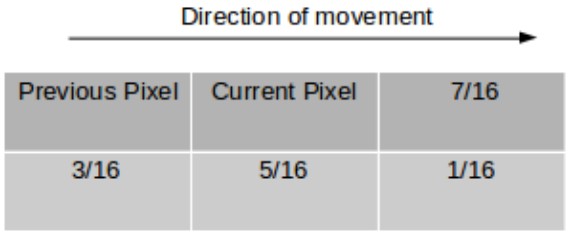


Figure 1.19: Floyd-Steinberg dithering algorithm - This figure demonstrates the how the error diffusion is applied to the surrounding pixels[19].

The design and implementation of the code started by creating an image processor child for the dithering algorithm. This would then apply the process of dithering to the image and return the coordinates generated from it. There were two major problems that were encountered when developing the code these were:

- Coordinate confusion - When handling an image and using x and y values to make a graph of the image. The image (0,0) value is contained within the top-right value and y-axis it going to the right and the x-axis is downwards. When first developing the code for the dithering algorithm, it was to be believed that the (0,0) point was at the bottom-left corner and the x-axis was to the right and the y-axis was upwards. This meant when trying to apply the algorithm the values were being defused in the completely wrong direction, causing confusion.
- Mishandling error diffusion - A major problem that occurred was mishandling the error diffusion. This was when developing the code the pixels the defusion was going to was entered incorrectly and applied it to the current pixel, meaning the colour value of the pixel was being doubled and then segregated. This caused the effect to incorrectly alter the image.

After these problems were fixed the dithering algorithm worked correctly and applied the style to any image that was taken as seen in Figure 1.18. The next stage was to develop a plotting algorithm, that would plot the dithering style.

1.4.9 Plotting Algorithms - Iteration Eight

The final stage of creating the Robotic Artist was creating a plotting algorithm that would be able to plot the dithering style. This would require creating a set of code that would loop through the coordinates that were developed from the Image Processor.

The initial stages of the code were to communicate the plotter to move to every point of the dithering style and place the pen up and down. This used the commands 'PA x y;', 'PU;' and 'PD;', to control the plotter and what was occurring. This process successfully plotted the dithering algorithm, however, it took around three to four hours to run the commands. So a method was developed to detect if the coordinates had a neighboring coordinate. If this was the case instead of bringing the pen back up, the pen would move to that neighbor and reduce the time for plotting. The points would then be marked as drawn and would not be drawn again in later iterations.

This finalizes the design and implementation of the code, the next part of this would be to test the final product for the last time to see any improvements or faults that require attention.

1.4.10 Testing

After and during the development of the code, tests were made and implemented to work alongside the requirements and code that were being developed. These tests were unit tests and testing specifications that can be seen in the Test Plan[1]. The two types of tests are expanded upon below.

- Unit Tests - These tests are created to test the source code of the project. These tests were developed using the python library Unittest, they are designed to test if a method works when the output cannot be visually seen. For example, if the coordinate_to_plotter() inside the PlotterController actually changes the one coordinate values to represent forty plotter values. If this is true that unit test will pass.
- Testing Specifications - These tests are developed to test code that cannot be tested through unit tests. This can be components that don't have a computation output but a visual one, for example, how do the final plots of styles look.

The unit tests can be called by calling the command 'python run_tests.py', this is a testing suite that connects to every set of unit tests that were developed. The testing specification tables can be seen within the Test Plan[1] along with an expansion on how the unit tests work. When the test specifications are all tested they are recorded inside the Test Records[4]. This takes the plan from the Test Plan and fills out what the actual outcome is and who did the tests on what date. These tests reference the testing specification plan of the version of the Test Plan that was created around the date on the record.

These tests help to find errors in the code, such as the segmentation error that was discussed in the Final Design chapter. The tests also helped to calculate that the dithering style will take around 3-4 hours and the edge style can take over 6 hours to plot. After the tests were done and problems that were found were looked at the project was complete. However, the final version did not have the functions admin, toggle admin view and the tutorial menu. This was due to time constraints within the project.

1.5 Critical Evaluation and Insight

Overall the project as a whole was successful but with errors or faults that have remained within the final product. The robot has managed to follow the requirements that were developed at the start of the project and can successfully plot an artist portrait of someone's face. However, the final outcome has the problem that is the amount of time a plot can take is over 3 hours at least. This makes it so the plotter cannot be used to demonstrate and is not time efficient, as if someone wanted a portrait plotted you will have to take a picture and then wait over 3 hours to get the portrait. This could be because the algorithms used to plot the device were inefficient and could have been planned or researched better during development.

The project's plan that was used to develop the overall project worked correctly to use Feature Driven Development and Kanban Board to help control and develop the workflow of the project. However, it could have been better if more research had been done before design and implementation stages of the project and not mainly alongside it. Then some of the problems that occurred may not have happened and this would have saved time. This can be seen when finding out that there were problems compiling OpenCV for python3 and then time was taken to change the language to python 2 after working out OpenCV was pre-installed for python 2 on the pi.

Overall, the project developed the two styles. These styles were the Edge-style, which used edge detections to represent the portrait and the Dithering style, which used Floyd-Steinberg Dithering algorithm[19]. These styles both worked correctly by the end of implementation, however, the dithering style was said to look good but the edge-style in most cases looked strange and underdeveloped. This meant that the edge-style failed a requirement that was to make the portraits developed from the code visually appealing.

Then other parts that could improve the end product is to make sure that all the features that were developed for the design were implemented into the GUI. This is the admin and helps menus and there functions. These were due to time constraints that were developed from problems occurring within the code.

By the end of the project, the requirements were met. However, if the project was going to be developed again from scratch it could be improved by looking into algorithms and then before developing them trying to predict how long they would actually take to plot. However, the project development alongside the version history and GitHub repository help to develop a structure in the code that tells any future users how the project evolved as it went along.

The main problem with the final outcome was the segmentation error which is discussed in the Final Design chapter. This could have been resolved through further research into the GUI library being used and the errors that may occur on some systems.

The next stage of the evaluation is talking about what could be done to develop the project further after the final deadline.

1.5.1 Future Development

This section of the evaluations is going to talk about the future development of the project. The future development that could occur for this project is listed below.

- Generating New Styles - The code structure has been designed to easily implement new artistic

style algorithms. This is because the code for the Image Processors and Plotter Controllers are using an inheritance format. This allows each style to have a plotter and image processor class, which inherit from the parent classes. Then when the algorithms are finally developed, there are few modifications that are required to make them work with the GUI. For example, with the style selection page, there is a Grid-Plane that currently contains the possibility of holding five styles but this number could be increased if more are required. An idea for more styles is to use deep learning techniques to generate portraits that evolve as each portrait is drawn, like the Cloud Painter[11].

- Look at the edge-style and try to find a way to make the overall drawing look more appealing. Maybe reduce the thickness of the lines or build a better facial structure using another algorithm.
- Fixing the segmentation error on the Raspberry Pi. This will most likely require time to try and track down the error and make alterations to fix this problem.
- Creating the admin login and toggle view, along with the tutorial window. This was part of planned items within GUI, however, they were never developed due to time constraints.
- Create a method inside the ImageProcessor class that removes the background from the taken picture. This could then be used to reduce the plotting time.

1.6 Versions and References

Version	Description	Date Modified
0.1	Set up the basic Documentation layout. This includes creating the content and the individual chapters	28/04/18
0.2	Wrote up the basic chapters for the 'Background and Analysis' and Introduction.	30/04/18
0.2.1	Made changes and alterations to 'Background and Analysis' to improve structure and quality of the text.	30/04/18
0.3	Wrote the baseline for the 'Process/Methodology' chapter and made grammar and spelling corrections.	01/05/18
0.3.1	Moved the 'Paul the robot' section in the Introduction chapter to the Background and Analysis chapter. This is because it talks about the background to the project and also talked about the background for the Cloud painter and Joto. Then how they helped to inspire the project. As well, proofread the document so far.	01/05/18
0.4	Wrote out the technical work chapter. The stages were changed to iterations and the Final Design Chapter was moved to before talking about the iterations.	02/05/18
0.5	Wrote out the Critical Evaluation and added the declaration and added the list of figures, tables and appendices to the document.	03/05/18
0.6	Connected all of the figures and cites. Ordered references.	04/05/18

Bibliography

[1] Matthew Howard (mah60);Requirement Specifications; 27/04/2018;
This document talks about the requirements that have been given to robotic artist project. The document can be found in the Appendices for this project.

[2] Matthew Howard (mah60); Design Specifications; 01/05/2018;
This document talks about the design of the robotic artist project. It goes over version control, the class structure and the GUI design. The document can be found in the Appendices for this project.

[3] Matthew Howard (mah60);Test Plan; 27/04/2018;
This document talks about the test plan that is used throughout this document. These are the different tests that are designed to be performed on the code to check if the code still works correctly.

[4] Matthew Howard (mah60);Test Records; 27/04/2018;
This document contains the tests that were performed onto the robotic artists. These tests have a description the results and who on what dates tested them.

[5] Agile Modelling; Feature Driven Development and Agile Modelling; 01/05/2018;
http://agilemodeling.com/essays/fdd.htm

This site gives a general idea of how the Feature Driven Development process is run within an agile methodology.

[6] Leankit by planview; Kanban Board; 01/05/2018;

https://leankit.com/learn/kanban/kanban-board/

This site talks about how the Kanban board works. It provides an input of how to prepare, use and optimize the efficiency of the project through this process.

[7] Trello; Trello homepage; 01/05/2018;
https://trello.com/home

This site was used to create and maintain the Kanban board that helped to control the workflow of this project.

[8] GitHub Inc; Github - Repository; 01/05/2018;
https://github.com/

This site contained and helped maintain the repository for the project to help maintain versions.

[9] Patrick Tresset, Frederic Fol Leymarie, Goldsmiths College; "Portrait drawing by Paul the robot"; 28/04/2018;
http://doc.gold.ac.uk/ ma701pt/patricktresset/wp-content/uploads/2015/03/computerandgraphicstresset.pdf

This is the main article that I will be using throughout the project, as it was where the idea from the project came from and talks about different methods and styles when it comes to developing 'Paul the Robot'.

[10] Joto; Joto creative drawer; 01/05/2018;
http://www.joto.rocks/

This site demonstrates a product that was developed by Joto that is a creative drawer. This product helped with the initial ideas and development of the project.

[11] Cloud Painter by Pindar Van Arman; Cloud Painter Creative Artist; 01/05/18;
http://www.cloudpainter.com/

This site talks a project of creating a robotic artist. However, this design involves making artificial intelligence, deep learning and computer creativity. This lead to good research on how to develop the project and what aspects to develop.

[12] OpenCv Documentation; OpenCv tutorials; 29/04/2018;
https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_tutorials.html

This resource talks about individual components that can be done with open cv to calculate image processing techniques. This site is a tutorial on how to use multiple of those techniques.

[13] Roland DG Corporation; X-Y Plotter DXY-885 Operation Manual; 30/04/2018; Cannot find a date of release;
The book looked at within this book was 5-1 to 5-5, Chapter 6. Look at the 8-13 to find the commands that were used within the project. This book provided instructions on how to send commands to the plotter and how to get it working through serial.

[14] pySerial Documentation; pySerial library; 30/04/2018
http://pyserial.readthedocs.io/en/latest/shortintro.html

This site talks about the pySerial library that was used to create a serial connection between the Raspberry Pi and the plotter.

[15] Sentdex; pyqt tutorial playlist; 02/05/2018;
https://www.youtube.com/playlist?list=PLQVvvaa0QuDdVpDFNq4FwY9APZPGSUyR4

The playlist provides a useful tutorial on how to use the PYQT library and how to create actions that you can apply to buttons and other features of the GUI.

[16] Python Wiki; Pyqt4 library; 26/04/2018
https://wiki.python.org/moin/PyQt4

This site talks about the pyqt library that I used to develop the GUI. As well, provides learning materials to understand the library.

[17] QT Documentation;QT Designer Manual; 26/04/2018;
http://doc.qt.io/qt-5/qtdesigner-manual.html

This site is a manual on how to use the Qt Designer software and demonstrates the software that was used to develop the GUI in this project.

[18] Computerphile; Filtering techniques youtube playlist; 03/05/18;
https://www.youtube.com/watch?v=C_zFhWdM4ic&list=PLzH6n4zXuckoRdljSlM2k35BufTYXNNeF

This youtube playlist goes over the three filtering techniques used within the edge-style. The filters are Gaussian blur, sobal and canny edge.

[19] Tannerhelland; Image Dithering: Eleven Algorithms and Source Code; 03/05/18;
http://www.tannerhelland.com/4660/dithering-eleven-algorithms-source-code/

This site was used to learn how the dithering algorithm worked and what the different versions of two-dimensional dithering are. This includes Floyd-Steinberg dithering algorithm, which is the version the dithering style is developed in.

Chapter 2

Appendices

2.1 Requirement Specifications

2.2 Design Specification

2.3 Test Plan

2.4 Test Record