Robotic Artist:Walter
G400 Computer Science, CS39440
Design Specification

Author: Matthew Howard
Candidate Number: 150035512
User ID: Mah60
Supervisor Name: Helen Miles
Supervisor ID: Hem23

Last Modified: 27/04/2018
Version: 0.3.1
Status: Work In Progress(WIP)

Contents

1 Introduction

This document is going to define the overall design of the product as a whole. It describes the structure of the product and how it maintains the integrity of the design.

The Design specifications will include version control and the UML diagrams that describe the structure of the product that has been developed. This includes UML diagrams for sequences and class layouts to describe dynamic and static views of the design.

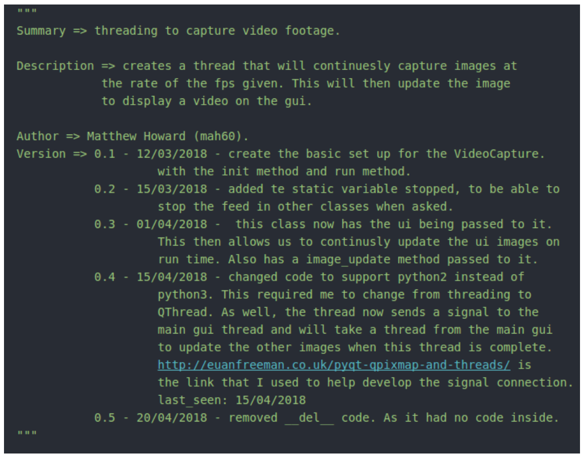
The design that has been developed must meet the requirements within the Requirement Specifications.[1]

2 Version Control

Version control is maintained in this project through a Github Repository. Github stores different versions of the code that have been committed to the repository as the project has progressed. Github also allows for branches to be made in which you can try out new versons of code and decide when to implement it to the master version of the product stored on Github.

Github has been used in this project to maintain different versions of the code and could be used to backtrack the code if there are major errors or faults in the project.

Another, form of version control within the code and documentation is that at the top of every module and the end of the document there is a version history section. This talks about what has been done to modify the document and why it has been done in a clear way and the date that this modification was done. An example of this can be seen in Figure 1.



```
"""
Summary => threading to capture video footage.

Description => creates a thread that will continuesly capture images at
            the rate of the fps given. This will then update the image
            to display a video on the gui.

Author => Matthew Howard (mah60).
Version => 0.1 - 12/03/2018 - create the basic set up for the VideoCapture.
                    with the init method and run method.
           0.2 - 15/03/2018 - added te static variable stopped, to be able to
                    stop the feed in other classes when asked.
           0.3 - 01/04/2018 -  this class now has the ui being passed to it.
                    This then allows us to continusly update the ui images on
                    run time. Also has a image_update method passed to it.
           0.4 - 15/04/2018 - changed code to support python2 instead of
                    python3. This required me to change from threading to
                    QThread. As well, the thread now sends a signal to the
                    main gui thread and will take a thread from the main gui
                    to update the other images when this thread is complete.
                    http://euanfreeman.co.uk/pyqt-qpixmap-and-threads/ is
                    the link that I used to help develop the signal connection.
                    last_seen: 15/04/2018
           0.5 - 20/04/2018 - removed __del__ code. As it had no code inside.
"""
```

Figure 1: Demonstrates the Version control that is being used in the video_capture.py Module. This includes the version number, description and date of change.

3 UML Diagrams

In this section of the design specifications, the document describes the framework of the code, how the code is structured and runs on real-time events that occur. This will be done by using UML diagrams.

3.1 Class Diagram

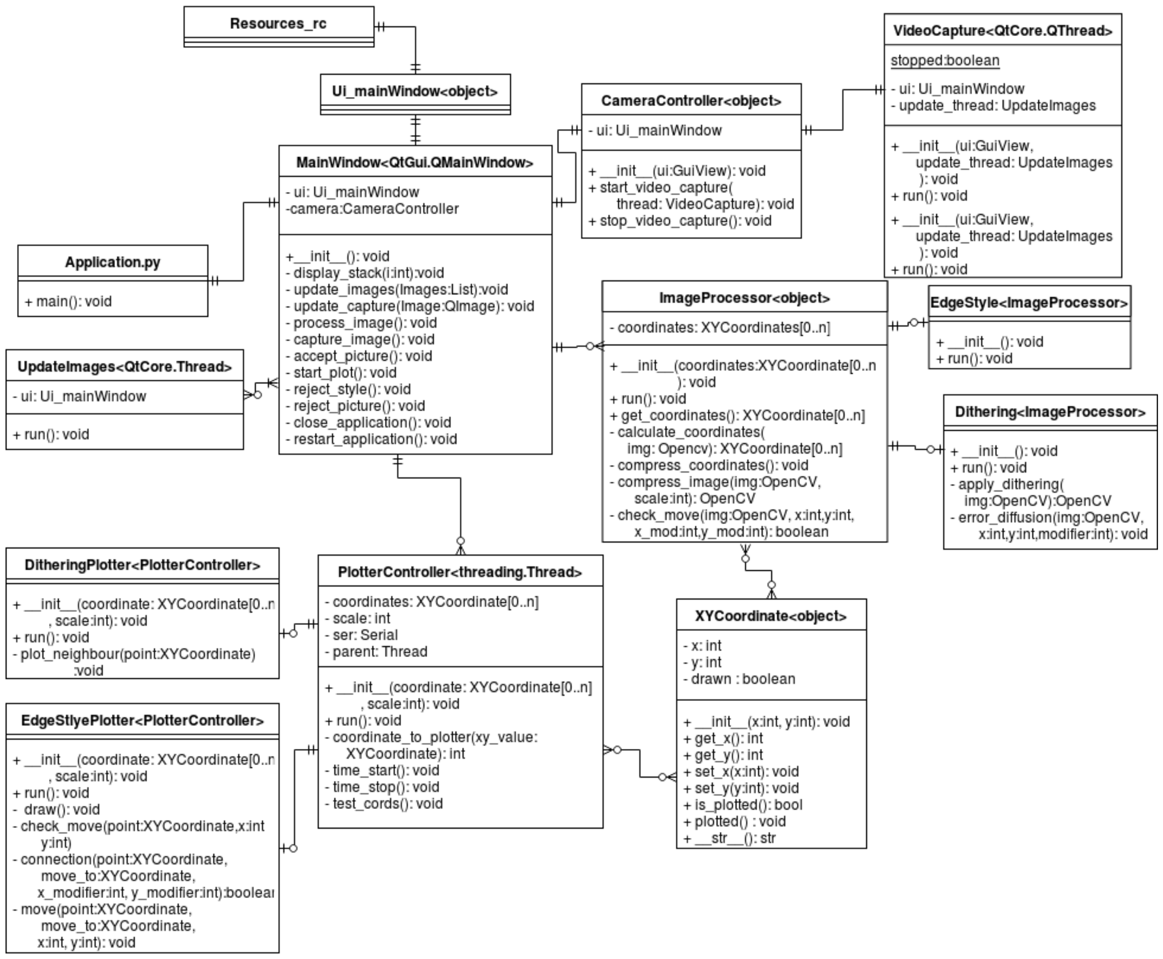

Resources_rc
Ui_mainWindow<object>
MainWindow<QtGui.QMainWindow>
- ui: Ui_mainWindow
-camera:CameraController
+__init__(): void
- display_stack(i:int):void
- update_images(Images:List):void
- update_capture(Image:QImage): void
- process_image(): void
- capture_image(): void
- accept_picture(): void
- start_plot(): void
- reject_style(): void
- reject_picture(): void
- close_application(): void
- restart_application(): void
Application.py
+ main(): void
UpdateImages<QtCore.Thread>
- ui: Ui_mainWindow
+ run(): void
CameraController<object>
- ui: Ui_mainWindow
+ __init__(ui:GuiView): void
+ start_video_capture(
thread: VideoCapture): void
+ stop_video_capture(): void
VideoCapture<QtCore.QThread>
stopped:boolean
- ui: Ui_mainWindow
- update_thread: UpdateImages
+ __init__(ui:GuiView,
update_thread: UpdateImages
): void
+ run(): void
+ __init__(ui:GuiView,
update_thread: UpdateImages
): void
+ run(): void
ImageProcessor<object>
- coordinates: XYCoordinates[0..n]
+ __init__(coordinates:XYCoordinate[0..n
): void
+ run(): void
+ get_coordinates(): XYCoordinate[0..n]
- calculate_coordinates(
img: Opencv): XYCoordinate[0..n]
- compress_coordinates(): void
- compress_image(img:OpenCV,
scale:int): OpenCV
- check_move(img:OpenCV, x:int,y:int,
x_mod:int,y_mod:int): boolean
EdgeStyle<ImageProcessor>
+ __init__(): void
+ run(): void
Dithering<ImageProcessor>
+ __init__(): void
+ run(): void
- apply_dithering(
img:OpenCV):OpenCV
- error_diffusion(img:OpenCV,
x:int,y:int,modifier:int): void
DitheringPlotter<PlotterController>
+ __init__(coordinate: XYCoordinate[0..n
, scale:int): void
+ run(): void
- plot_neighbour(point:XYCoordinate)
:void
EdgeStlyePlotter<PlotterController>
+ __init__(coordinate: XYCoordinate[0..n
, scale:int): void
+ run(): void
- draw(): void
- check_move(point:XYCoordinate,x:int
y:int)
- connection(point:XYCoordinate,
move_to:XYCoordinate,
x_modifier:int, y_modifier:int):booleai
- move(point:XYCoordinate,
move_to:XYCoordinate,
x:int, y:int): void
PlotterController<threading.Thread>
- coordinates: XYCoordinate[0..n]
- scale: int
- ser: Serial
- parent: Thread
+ __init__(coordinate: XYCoordinate[0..n]
, scale:int): void
+ run(): void
- coordinate_to_plotter(xy_value:
XYCoordinate): int
- time_start(): void
- time_stop(): void
- test_cords(): void
XYCoordinate<object>
- x: int
- y: int
- drawn : boolean
+ __init__(x:int, y:int): void
+ get_x(): int
+ get_y(): int
+ set_x(x:int): void
+ set_y(y:int): void
+ is_plotted(): bool
+ plotted() : void
+ __str__(): str

Figure 2: Displays the UML class diagram. This is the structure of all the classes in the products code.

Figure 2 describes the structure of the code that has been developed. The structure of the code has been developed around the MainWindow class, which is the code for controlling the graphical unit interface(GUI). This code is connected to the Ui_Mainwindow, which is a set of code that has been developed from the QT Designer software[2]. This allowed the overall framework for the GUI to be developed using various tools within the software.

After this there are three major sections to the class diagram(Figure 2). These are the Camera, Image Processor, and the PlotterController.

The CameraController controls the inputs and outputs from the camera. The VideoCapture is the main thing the controller works with. This is done by creating a thread that continuously sends frames to the GUI to be updated. Once the method stop_video_capture is called, then the video capture is stopped and the picture is taken from the frame that the thread was on at that time. The sequence of events can be seen in Figure 3. The thread is stopped by there being a static variable within the VideoCapture class called 'stopped', this variable is a boolean. This is changed to True when the video thread should be stopped and to False when the capture is running.

The next sections of the class diagram are the ImageProcessor classes. The ImageProcessor class will be a parent to all the styles that are developed for this program. The code that is generated for this class will be used over all the individual ImageProcessors. The main requirements of each style is that they will require a partner PlotterController to plot the final output and an output of coordinates to be plotted. This is where the XYCoordinate class is used to create a generic structure to translate data between the two classes. The XYCoordinate stores x,y values of points to be plotted. As well, a boolean that determines whether the point has been plotted or not.

Finally, we have the PlotterConroller which is a parent class to all plotters for their specific style. The PlotterController will contain code that is generic to all styles plotting routines. An example of this is the coordinate_to_plotter() method which will translate coordinate values to plotter values. The overview of the plotter requirements is that it must send serial commands in HPGL to the plotter, to plot out the specified style for that plotter controller.

3.2 Sequence Diagrams

Figures 3, 4 and 5 describe the sequence of events that are taken towards plotting a Dithering style.

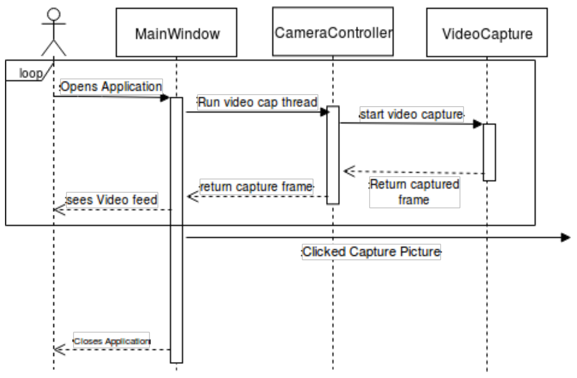


Figure 3: Displays the sequence that is taken to display the video feed at the start of the application. The sequence diagram shows that the MainWindow will go into a continuous loop of talking to the Camera and VideoCapture class to get frames to display on the GUI as the video capture. When the Capture Image button is pressed by the user, then the GUI will move onto the next page asking if the captured image is suitable for what the user wants.

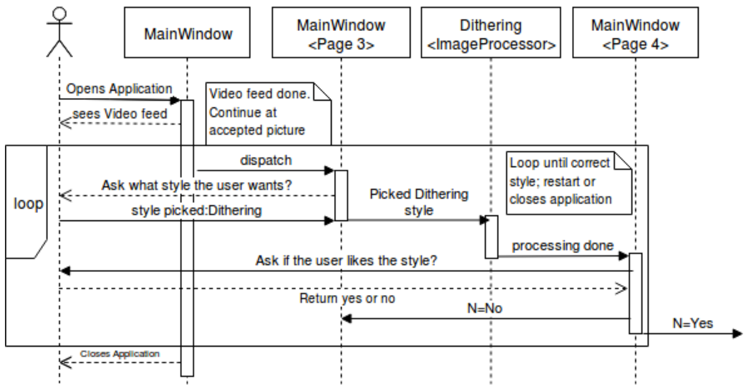


Figure 4: Displays the sequence that is taken to process an Image. This is after the sequence of events in Figure 3 and the Image has been accepted to be processed. The user will be taken to page 3 of the GUI. This is the style selection page. After one style has been selected the image will be processed for the specified style. In this case, it is the Dithering style. So the program will run the method run() in the image processor Dithing.py. After this the finalized style will be displayed on page 4. Then there are two sequences of events either the user can accept the style and start the plot or reject the style and go back to page 3.

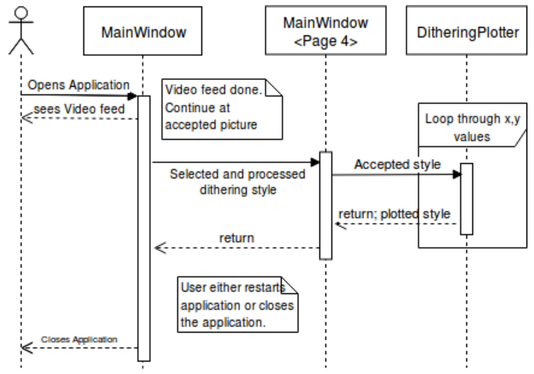


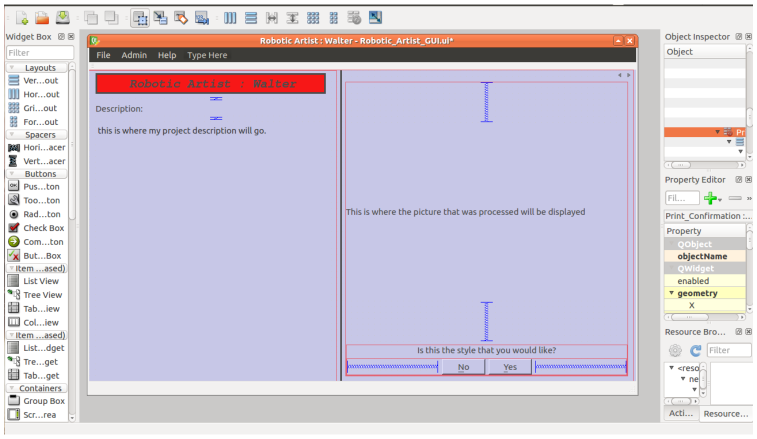
Figure 5: Displays the sequence that is taken to plot the style that was processed through the given style. This involves when the style has been accepted on page 4, as continued from Figure 4. Then the coordinates are gathered from that style in the MainWindow class, then the thread for plotting that specified style begins. This involves sending multiple HPGL commands through serial communication to the plotter. After this task is plotted and all coordinates that were made are complete then the application is restarted from the beginning or the application is closed.

4 Graphical Unit Interface(GUI) Design

This section will discuss the interface design that was developed for the project. The design was developed for python 2.7 using the library PYQT 4[3]. The overall development was done using the software called 'QT Designer'[2]. This allowed the project to save time as the GUI didn't take as much time to develop. The project being developed can be seen in Figure 6.

Figures 7, 8, 9 and 10 show the design of the overall GUI. From pages one to four in respective order. These designs were all made with QT Designer and then the actions and code were added to the MainWindow class that can be seen in Figure 2.

The design also has a description of what the project is about on the left-hand side of the GUI. This is a label that takes text from the description.txt file that can be found in the code folder.


Robotic Artist : Walter - Robotic_Artist_GUI.ui*

Robotic Artist : Walter

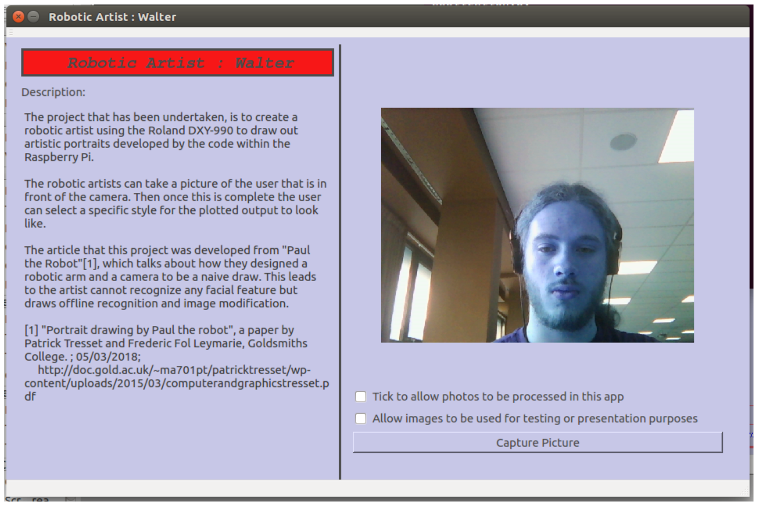
Description:

this is where my project description will go.

This is where the picture that was processed will be displayed

Is this the style that you would like?

Figure 6: This is an example of how the QT designer software looked when developing the GUI for the project



Robotic Artist : Walter

Robotic Artist : Walter

Description:

The project that has been undertaken, is to create a robotic artist using the Roland DXY-990 to draw out artistic portraits developed by the code within the Raspberry Pi.

The robotic artists can take a picture of the user that is in front of the camera. Then once this is complete the user can select a specific style for the plotted output to look like.

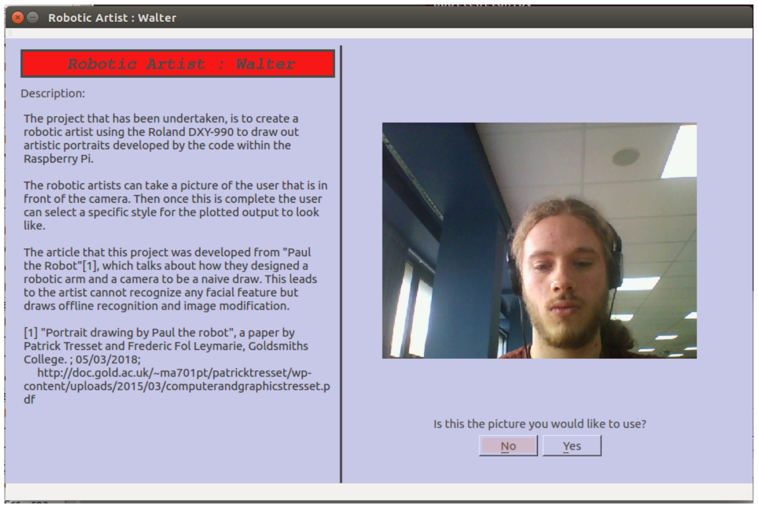
The article that this project was developed from "Paul the Robot"[1], which talks about how they designed a robotic arm and a camera to be a naive draw. This leads to the artist cannot recognize any facial feature but draws offline recognition and image modification.

[1] "Portrait drawing by Paul the robot", a paper by Patrick Tresset and Frederic Fol Leymarie, Goldsmiths College. ; 05/03/2018;
    http://doc.gold.ac.uk/~ma701pt/patricktresset/wp-content/uploads/2015/03/computerandgraphicstresset.pdf

☐ Tick to allow photos to be processed in this app

☐ Allow images to be used for testing or presentation purposes

Figure 7: Page 1: Video capture. This page was designed to contain the video capture from the camera. The design of the page is to have a label that a QImage can be added to. During runtime, the image will continuously be updated through a QThread. The other parts of the page are two checkboxes and a button at the bottom of the page, which stops the video capture and take the frame it was on as a picture. This routine can be seen in Figure 3.



Robotic Artist : Walter

Robotic Artist : Walter

Description:

The project that has been undertaken, is to create a robotic artist using the Roland DXY-990 to draw out artistic portraits developed by the code within the Raspberry Pi.

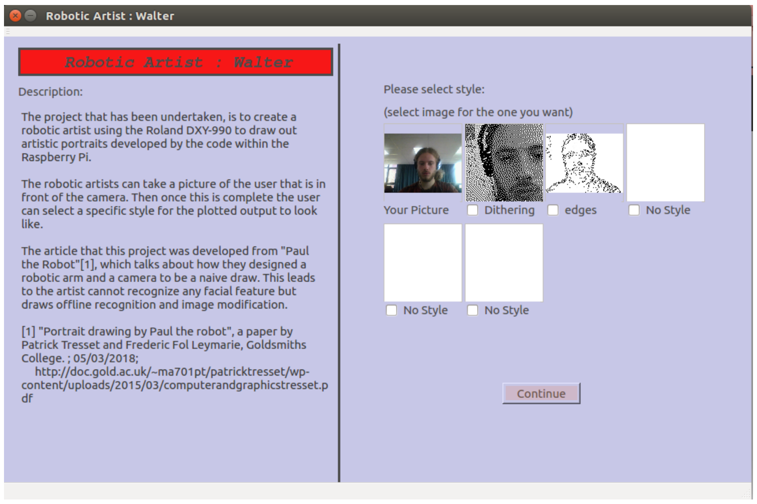
The robotic artists can take a picture of the user that is in front of the camera. Then once this is complete the user can select a specific style for the plotted output to look like.

The article that this project was developed from "Paul the Robot"[1], which talks about how they designed a robotic arm and a camera to be a naive draw. This leads to the artist cannot recognize any facial feature but draws offline recognition and image modification.

[1] "Portrait drawing by Paul the robot", a paper by Patrick Tresset and Frederic Fol Leymarie, Goldsmiths College. ; 05/03/2018;
    http://doc.gold.ac.uk/~ma701pt/patricktresset/wp-content/uploads/2015/03/computerandgraphicstresset.pdf

Is this the picture you would like to use?

Figure 8: Page 2: Picture Acceptence. This page has a label that will contain the captured frame. Then there are a set of buttons that ask yes or no to whether the picture taken is correct.



Description:

The project that has been undertaken, is to create a robotic artist using the Roland DXY-990 to draw out artistic portraits developed by the code within the Raspberry Pi.

The robotic artists can take a picture of the user that is in front of the camera. Then once this is complete the user can select a specific style for the plotted output to look like.

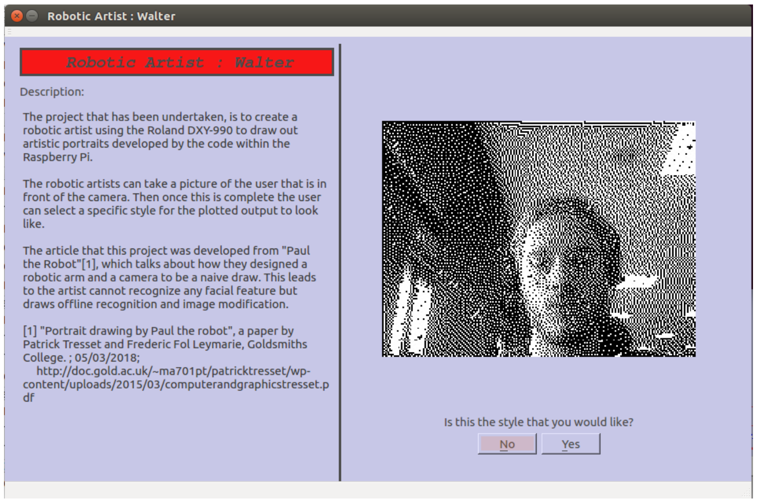
The article that this project was developed from "Paul the Robot"[1], which talks about how they designed a robotic arm and a camera to be a naive draw. This leads to the artist cannot recognize any facial feature but draws offline recognition and image modification.

[1] "Portrait drawing by Paul the robot", a paper by Patrick Tresset and Frederic Fol Leymarie, Goldsmiths College. ; 05/03/2018;
http://doc.gold.ac.uk/~ma701pt/patricktresset/wp-content/uploads/2015/03/computerandgraphicstresset.pdf

Please select style:

(select image for the one you want)

Figure 9: Page 3: Style selection. This page is all about selecting the individual style you want. There is a grid layout with labels inside them that display either text or images. These make a grid that displays the styles image and the name of each of the styles. In this section, you are only allowed to select one style. Then once you have selected that style and pressed the button the program will process the image from the style that you have selected. At the moment the GUI has a limit of 5 styles to pick from when they are all made. However, modification to the code will allow more in line with additional styles being developed.



Description:

The project that has been undertaken, is to create a robotic artist using the Roland DXY-990 to draw out artistic portraits developed by the code within the Raspberry Pi.

The robotic artists can take a picture of the user that is in front of the camera. Then once this is complete the user can select a specific style for the plotted output to look like.

The article that this project was developed from "Paul the Robot"[1], which talks about how they designed a robotic arm and a camera to be a naive draw. This leads to the artist cannot recognize any facial feature but draws offline recognition and image modification.

[1] "Portrait drawing by Paul the robot", a paper by Patrick Tresset and Frederic Fol Leymarie, Goldsmiths College. ; 05/03/2018;
http://doc.gold.ac.uk/~ma701pt/patricktresset/wp-content/uploads/2015/03/computerandgraphicstresset.pdf

Is this the style that you would like?

Figure 10: Page 4: Style confirmation. This page has a label that contains the stylized image and then a button at the end of the page that allows the user to accept or reject the style.

5 Versions

Version	**Description**	**Date Modified**
0.1	created basic structure for introduction and version control. Added sequence diagrams and class diagram.	23/04/2018
0.2	wrote the content for the UML diagram. As well, added to the section to the GUI design section.	25/04/2018
0.3	proofread and corrected the document. As well, added reference when required to the document.	26/04/2018
0.3.1	made minor corrects; changed page 4 to style confirmation from style acception	27/04/18

References

[1] Matthew Howard (mah60);Requirement Specifications; 25/04/2018
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] 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.

[3] 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.