

**Diploma in**

**Mechatronics Engineering**

Final Year Project Report

**Project Title:**

Research and Development of an Education robot

**Project Supervisor:**

Dr Edwin Foo

**Project Student:**

Ong Xian Hao, Jordan [154358G]

**Table of Contents**

**Acknowledgement**

**Table of Contents**

**Introduction**

1.0 Background / System Overview

1.1 Project Objectives

**Project Scope**

\* Key Specifications & Features

\* Major Deliverables

\* Gantt Chart

**Project Implementation**

\* Hardware Development

\* Software Development

\* Integration and Testing

\* Key Technical Issues and Solutions

**Conclusion**

\* Accomplishment / Outcome

\* Future Enhancement

Appendices

Appendix A -

Appendix B -

**Acknowledgement**

# Thanks

**1.0 Introduction**

The education robot is a social robot targeted at early childhood.

A social robot is an artificial intelligence (AI) system that is designed to interact with humans­1. An example of a social robot, which is currently in many households, is Jibo, a small robot that is able to empathize. Using machine learning, speech and facial recognition, and language processing, it is able to learn from previous interactions and adapt to situations.2

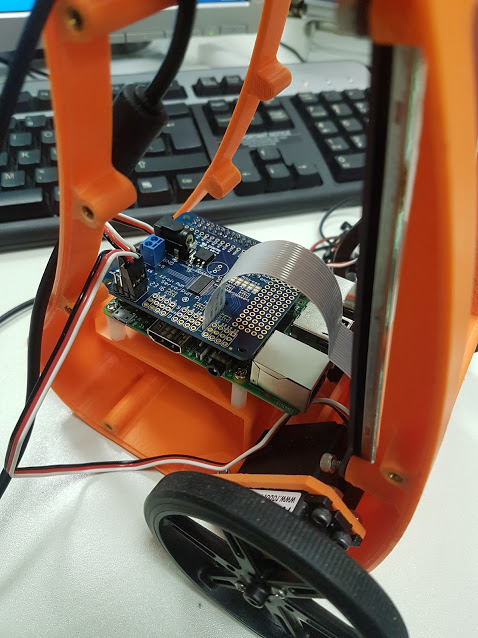
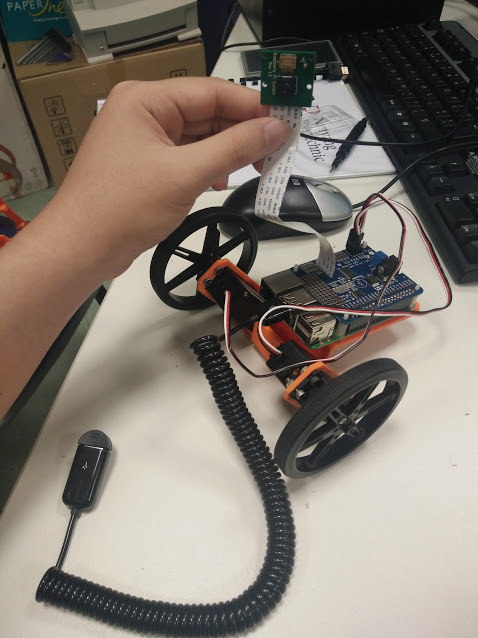
The education robot will very much be like Jibo using speech and facial recognition to interact with its users. But unlike Jibo, the education robot is a teacher, instead of a companion.

The robot is able to move around on its two wheels that are attached onto continuous rotation servo motors that are mounted on the side of the robot. The motors are controlled by a Raspberry Pi using a PWM motor hat. This is a feature that most social robot like Jibo does not possess.

The robot is also able to recognise faces and understand speech using a camera and a microphone, respectively. The purpose for facial and speech recognition is for the robot to be able to interact with the children, giving an impression that the robot is able to communicate and socialise like as though it is a person.

The motors controls, facial and speech recognition processing is all done on a Raspberry Pi 3B.

On the front of the robot is an 800x400mm LCD resistive touch screen that users can interact with. It can be used to display the robot’s “emotion” in the future.



**2.0 Project Objectives**

My project is about the research and development of an education robot. I’m tasked with researching various components of the robot and implementing them on to the robot.

My main objectives are to:

1. Implement motors controls on the robot
2. Implement facial and speech recognition
3. Create responses to various emotions and commands

The robot has to be able to recognize users from their faces and detect their emotions, be able to understand speech, and move around on its own.

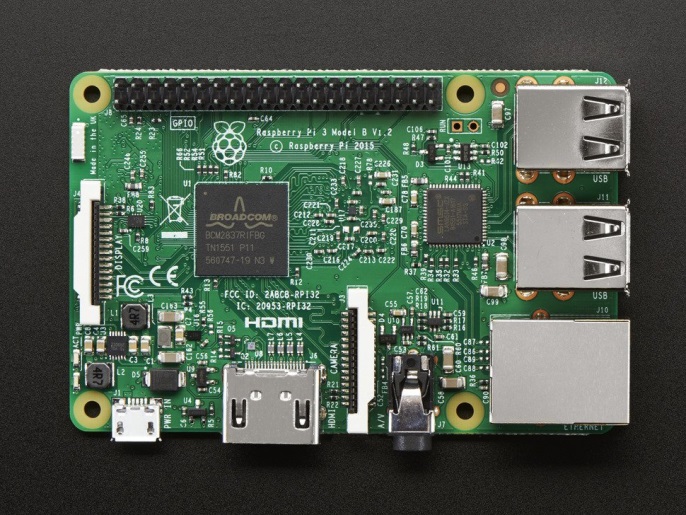
**2.1 Gantt Chart**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ****Weeks****  ****Activity**** | ****1-2**** | ****3-4**** | ****5-6**** | ****7-8**** | ****9-10**** | ****11-12**** |
| ****Research on various components**** |  |  |  |  |  |  |
| ****GUI building**** |  |  |  |  |  |  |
| ****Motor controls setup**** |  |  |  |  |  |  |
| ****Implementing Face Detection**** |  |  |  |  |  |  |
| ****Implementing Voice Recognition**** |  |  |  |  |  |  |
| ****Testing**** |  |  |  |  |  |  |

**3.0 Specifications**

**Raspberry Pi**

The Raspberry Pi is a fully functioning Linux computer approximately the size of a credit card. Its creator, Eben Upton, wanted to create a cheap device that could teach programming skills and hardware understanding for students at the pre-university level. But thanks to its small size and accessible price, many people quickly adopted it for projects that require more than a basic microcontroller. The Pi will be powered by a standard phone charger wire.

­­

**Waveshare 5” HDMI LCD (B)**

The screen on the robot is an 800x400 LCD screen connected to the Pi via HDMI. It supports resistive touch and it is powered by a phone charger wire just like the Pi.

To set up the screen we have to edit the config files

Add in these new lines.



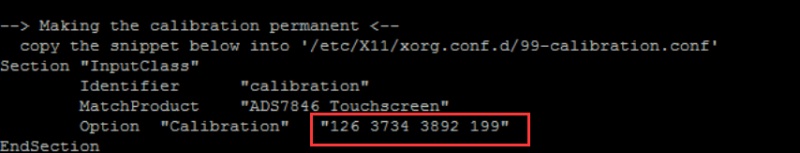
These are just for the display. If you want to use the touch screen, you will have to calibrate it. Install the calibrator to be able to calibrate the touchscreen

****

You can open the calibrator from the menu or you can use command lines

If you’re running from SSH



A GUI will come up on the HDMI touchscreen and you will have to click on the four points that pop up to calibrate the screen. Once you done that the GUI will close and there will be instructions on the terminal.

It make look a bit different and that’s is okay. Open another terminal and on it enter this line.



Overwrite everything with the previous instructions into here and save by pressing Ctrl-X and then Y, press enter once more to save the file and reboot for the calibration to take place.

**Facial Recognition – Microsoft Cognitive Services**

Microsoft Cognitive Services provides an open source facial recognition API, among other recognition APIs. The facial recognition API is able to detect faces, identify and verify faces, and recognise facial attributes such as gender, age, and emotion. I mainly use it for face detection, and to determine facial attributes such as gender, age and emotion from the user.

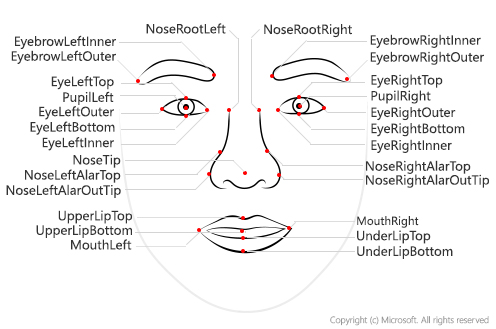
**Start**

**API returns analysis in a .json format**

**Send file/URL for analysis**

**API analyse image from file/URL**

**End**

The API is able to determine gender, age and emotion from our facial attributes. It is able to do so by analysing each facial attribute. The facial attributes it will analyse is shown in the figure.

*Figure 3.0.1 – Facial Attributes*

It is able to determine a variety of emotion from faces, the emotions the API is able to detect are; anger, contempt, disgust, fear, happiness, neutral, sadness, and surprise. It will guess the emotion based on confident levels, and return all the emotions and its respective confident levels.

**Facial Recognition – Pi camera**

The Pi camera is what the robot will use to take pictures for facial recognition. It is the standard camera module for the pi. It attaches via a 15cm ribbon cable to the CSI port on the Raspberry Pi and it is able to record videos and capture still images.



**Speech Recognition – Google Cloud API**

Google provides its own open source speech recognition API. I chose Google Cloud instead of Microsoft Cognitive Services because Google Cloud’s API serves my project better. It is based on the same speech recognition that powers Google’s voice search on the android phones and I’m able to input words that the API will more likely recognise among similar sounding alternatives. For that reason, I decided to use Google Cloud instead of Microsoft Cognitive Services.

**End**

**API translate speech to text**

**Send audio file for analysis**

**Returns transcribed speech based on confidence level in .json format**

**Start**

**PyQt**

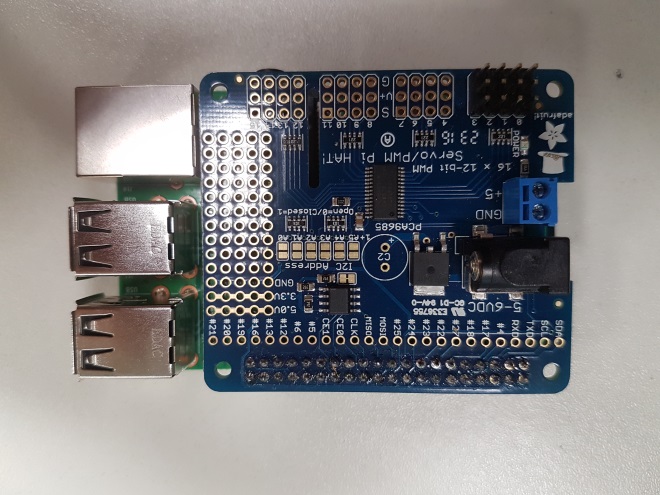
The GUI wrapper I chose to use was PyQt due to it being easy to pick up. I considered using TKinter at first since it was already installed on the Pi but when I looked up tutorials on making python GUIs most people recommended using PyQt or wxPython for big applications. I didn’t even considered using wxPython because I read a lot of issues on forums and videos about how difficult it was to do simple thing. I ran an example of a GUI made with wxPython; the application would crash a lot.

Here is a table to summarise my points.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **TKinter** | **wxPython** | **PyQt** |
| **Versions** | Python 2 and 3 | Python 2 | Python 2 and 3 |
| **Installation** | Installed by default | Not installed by default | Not installed by default |
| **Stability** | Stable | Not stable | Stable |
| **Designing an application** | Difficult to make a huge application | Simple to do with wxFormBuilder | Simple to do with QtDesigner |

Hence I used PyQt because it was quite easy to pick up, relatively simple to design a huge application with, and it does not crash due to unknown errors.

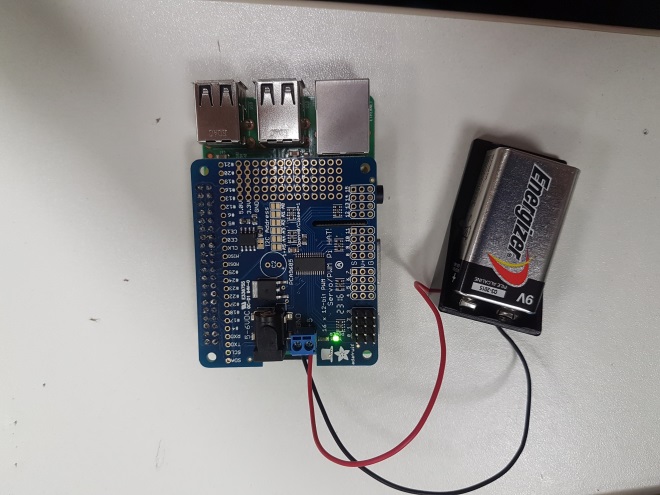
**Motor Hat**

The motors are two SpringRC SM-S4303R that are powered by and controlled with a 16 channel PWM/Servo hat from Adafruit. The hat is connected to the Pi via the GPIO pins. Using i2c, we can send signals to control the motors connected to the hat.

The hat connects to the Pi via the GPIO pins

Adafruit 16 channel PWM/Servo hat

The hat has two power supplies. One of them is VCC, the 3.3V power from the Raspberry Pi. It is used to determine the i2c and PWM signal logic level and to power the PWM chip. As long as it is mounted onto a working Pi, this power supply will be on.

The second power supply is for the servos, this power supply should be 5 or 6 VDC. The power is connected through the blue terminal block or the DC jack at the side of the hat. If it is powered up, a green led should light up.

DC power supply connected to the DC jack

9V battery connected to the blue terminal

**Continuous Rotation Servo Motor**

As mentioned before, the motors we are using to move the wheels are two SpringRC SM-S4303R. These are the specifications.

|  |  |  |
| --- | --- | --- |
| **6V** | **Max Speed** | 54 RPM |
| **Stall Torque** | 71 Oz-in |
| **4.8V** | **Max Speed** | 43 RPM |
| **Stall Torque** | 46 Oz-in |
| **Weight (g)** | | 41 |
| **Size (mm)** | | 41.3x20.3x40.2 |

This motor has great stall torque; it is more than enough to support the weight of the education robot.

|  |  |
| --- | --- |
| **Connections** | |
| **Servo wires** | **Pins** |
| **Black** | **Ground (G)** |
| **Red** | **Voltage supply (V)** |
| **White / Yellow** | **Control (S)** |

**3.1 The code**

**PyQt – The GUI**

The QtDesigner is a GUI builder tool. Using it is a drag and drop interface, with it you can quickly build a GUI without having to write and single line of code. It also possible to quickly edit the properties of each widget from the bottom right table.

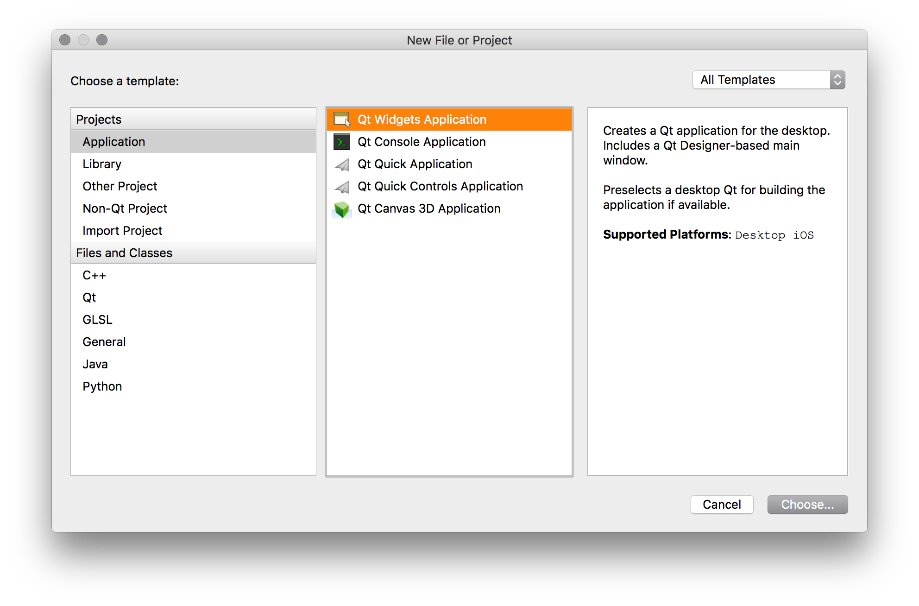
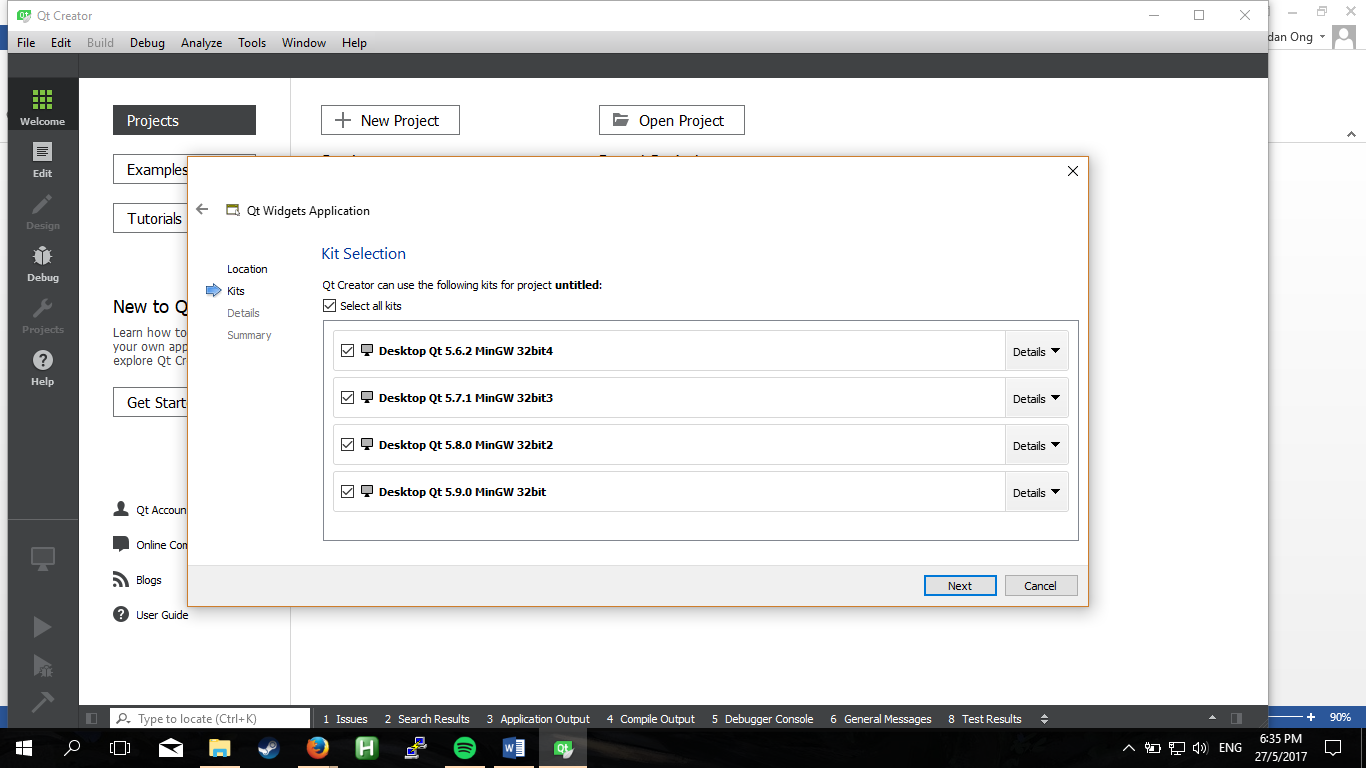
This section will walk you through installation and implementing the GUI into the main code. It will not go through the functions of every widgets in QtDesigner as there are too many.

**Installation**

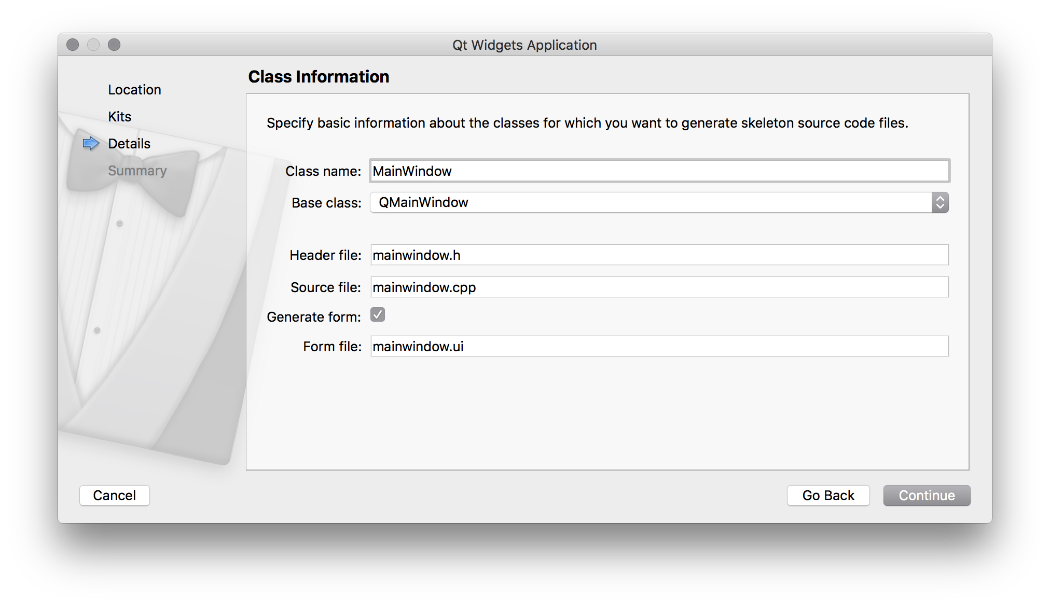
Download Qt open source at <https://info.qt.io/download-qt-for-application-development> for the GUI creation.

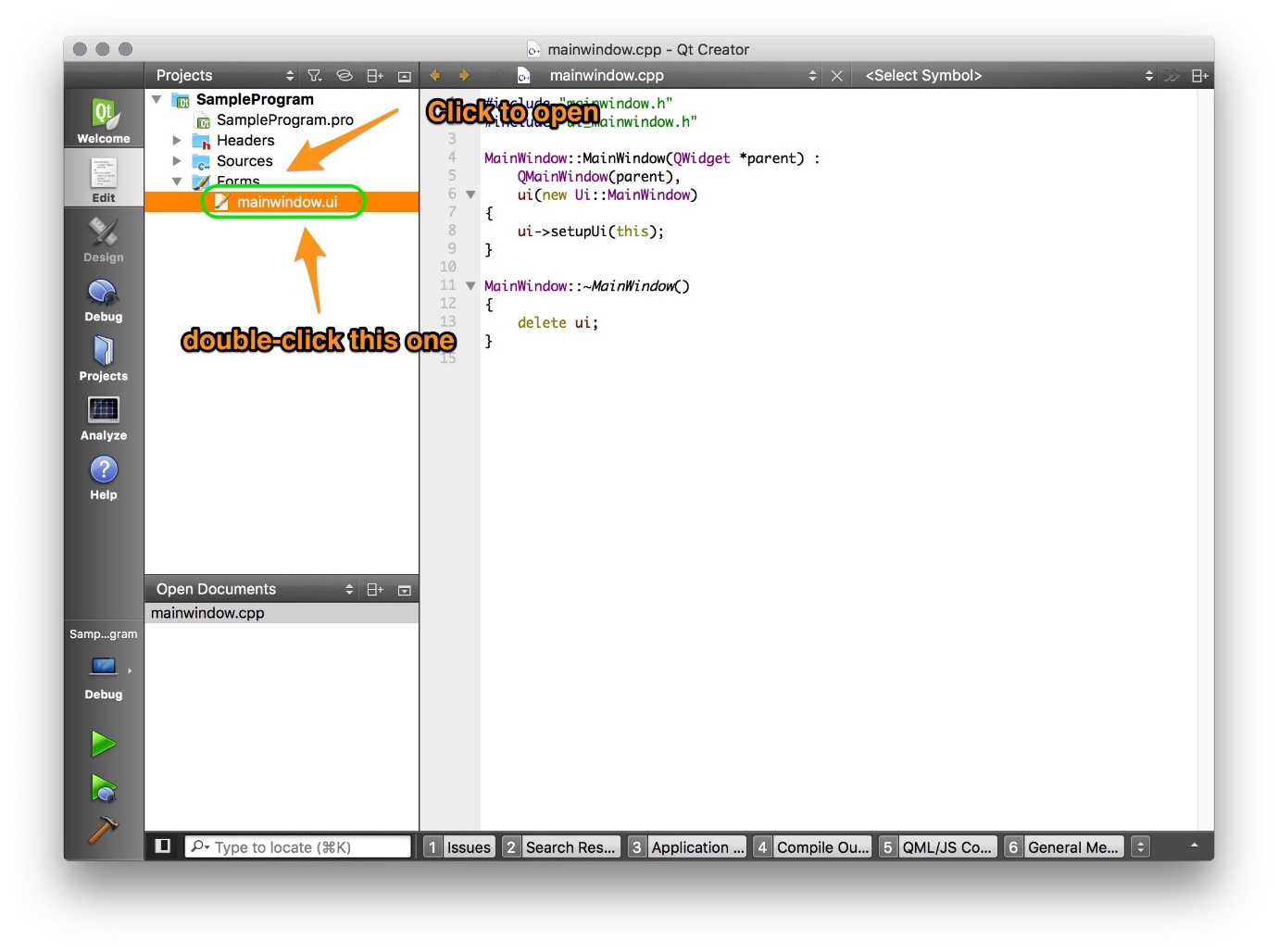
Download python3 at <https://www.python.org/downloads/> so that you can run the GUI on the Raspberry Pi.

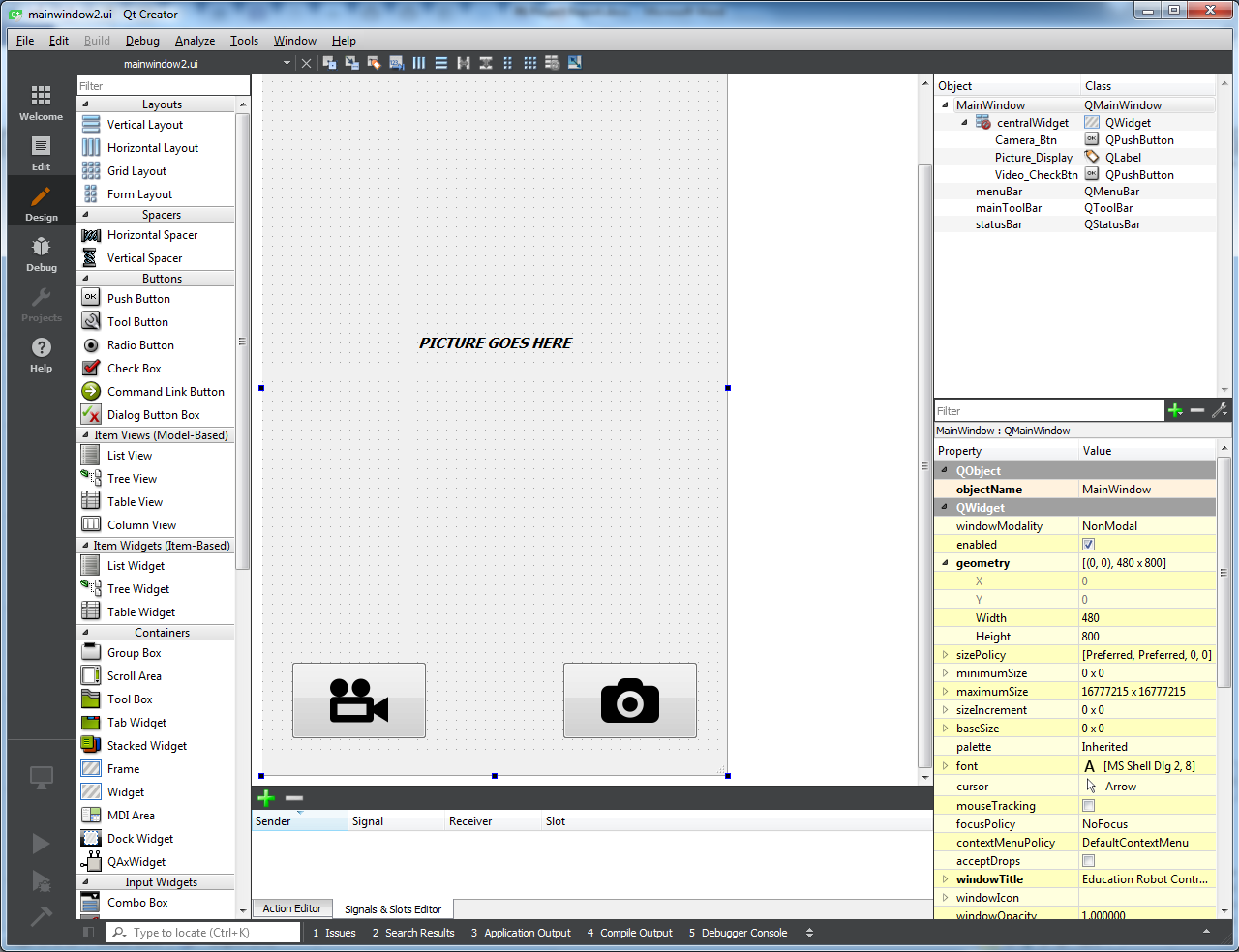
**Creating a new GUI**

Start a new project in QtCreator and select “Desktop Application” and choose where you wish to save the program at.

It does not matter what kit you pick, just pick all the kits.

The name for the class is important as it will be referenced in the code later. Its best to leave it as it is so that it is easier to follow the examples.

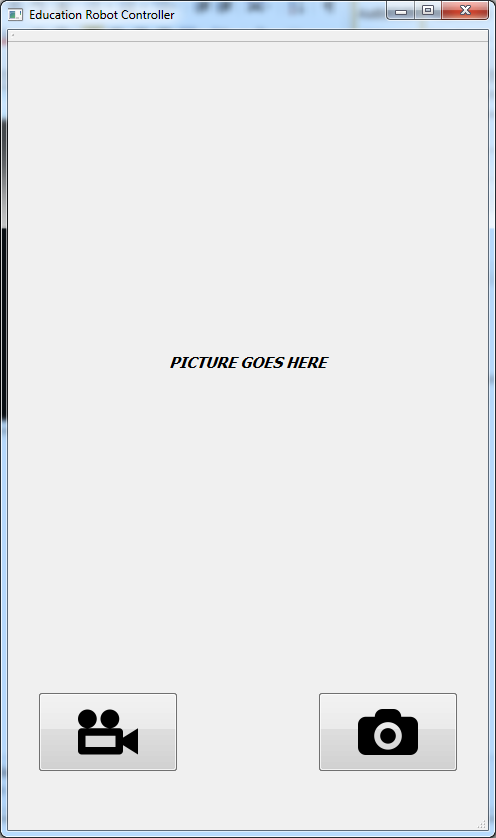
Select **None** for version control and click **Finish.** Qt then will create a C++ framework. What you should be interested in is mainwindow.ui, that is what you will use to create GUIs

 This is the finished GUI using QtDesigner.

Once you are satisfied with the GUI, save it and run this command in the same folder as where the GUI file is saved.



It will generate a file that is able to execute launch a standalone application with the GUI that you designed.



**Events handling**

The buttons are there but nothing would happen if they are clicked on. That is because event handling have not been implemented.

This is an example of event handling. Event handling is done in another program so that the code is much neater.

****

The GUI python code is imported in with the class and it will launch with **form.show().** The event handlings are inside the **signals()** function.

In this case, once the camera button is clicked, the function **camera()** will run.

Each type of widget will emit its own signals so you will have to look through the documentation for other widgets or search online for a few examples.

As you can notice, the functions for the buttons have not been defined yet.

The functions will be explained later.

**3.1.2 Functions – Motor**

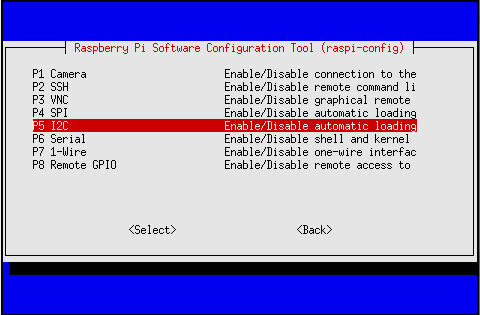
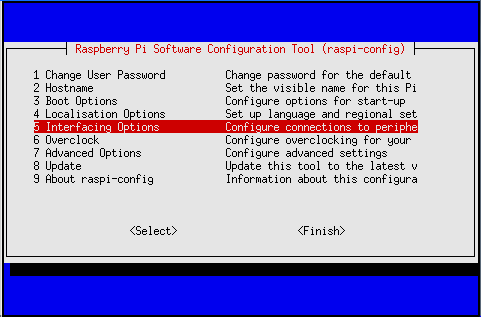
The motors are what moves the robot around. This section will explain on how to setup the motors and implement them into the code.

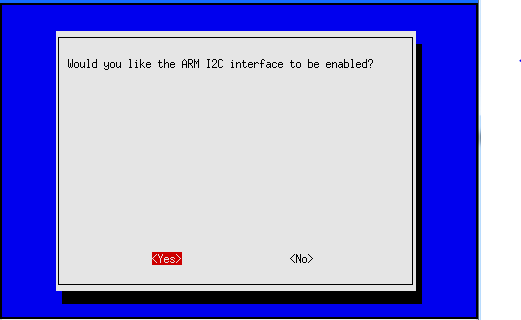
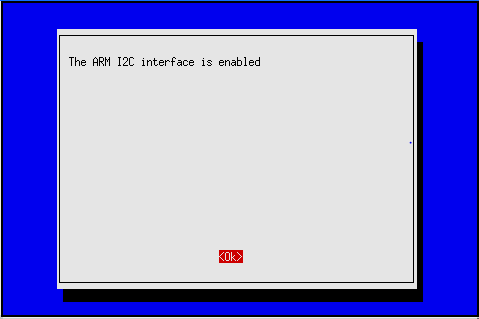
**Configuration**

Before we can start controlling the motor, we must first set up the Raspberry Pi. The i2c bus allows multiple devices to be mounted onto the Pi, each with its own unique address. It is very crucial to be able to see the addresses of the devices that are connected to the Pi.

Install the i2c-tools utility first.



Next we have to enable i2c kernel support in the Pi. To do so, run **sudo raspi-config** in the Terminal and follow the following steps.

After the enabling I2C, reboot and we are done with configuration.

**Getting I2C address**

To get the I2C address of the hat, type the following command in the terminal



This shows that two addresses are in used, 0x40 and 0x70. 0x70 is the address of all the hats that are stacked on the Pi. It means that addressing 0x70 will address all the hats.

**The base code for motors**

Install the modules for the hat first.



Once that is installed, we can start on the code for the motors. This is how to initialize a new object for each hat.



First, set the address of the hat to an object ‘pwm’ and set the PWM frequency from a range of 40 to 1000. Frequency determines how 'long' each pulse is in duration from start to finish, taking into account both the high and low segments of the pulse. 60 freq is a good frequency for servos.

Now you can start the motors. These are the only two lines of code that starts the motor turning.



These are the values used in the code.

|  |  |  |
| --- | --- | --- |
|  | **Left** | **Right** |
| **Frequency** | 60 | |
| **Forward movement** | (0,150) | (0,600) |
| **Reverse movement** | (0,600) | (0,150) |
| **Stop** | (0,0) | |

The motor functions are controlled by W, A, S, D and Q. PyQt is able to detect key presses fairly easily and the key presses are quite explicit, allowing the code to be very simple and neat.



**3.1.2 Functions – Facial Recognition**

The robot will use Facial Recognition to be able to read emotions and identify users from strangers. This section will explain how the code interact with the API, the Pi camera and the face detection portion of the code.

The face detection part of the code is a bit tricky to explain. Firstly, you need to understand how to make a request to the API.



This is the ‘quick start’ example the documentation provided, I added a few comments to explain how the API works on a basic level.

This is the response expected with a request parameter of face id, face landmarks and face attributes of gender, age, glasses and emotion. This response is in a .json format. The response will be empty if no faces are detected. 

Before moving on to implementation, first install the API module by running the following code in the Linux terminal of the Pi. (*You don’t have to do this if you want to just use the ‘quick start’ example)*

sudo pip3 install cognitive\_face

After installing the module, you can proceed to implement the API to the program. This is how the request code will basically look like.

With the modules it is much easier and simpler to make request to the API.

It is much cleaner as you can notice, all the arguments needed are the subscription keys and if you want the API to return the face id or rectangle and what face attributes you want it to analyze.

The module is able to figure out if the image sent is a URL or an image file so there is no need to declare the type of content you are sending for request.

The results of the analysis will be inside a list instead of a .json file, making it easy to obtain the desired values. So all that’s left to do is to slice the list and get the required values from it.

This is how I did it.



Depending on the request parameter, you will have change the code to obtain all the values you requested. The three main list are **faceId**, **faceRectangle** and **faceAttributes.**

**faceRectangle** contains sub-list that contains the data of the top, the extreme left, the width and the height of the rectangle that surrounds the detected face. The sub-list are **top**, **left**, **width**, and **height**.

Depending on what you request, the sub list of **faceAttributes** will change. The names are the same as the names used in the request.

**Pi camera**

The Pi camera comes into play here. Its job in the grand scheme of things is to capture the image that will be used for recognition.

This is the basic code for the Pi camera.



The camera function is called with a **with** function to make the code neater as the camera always has to close after each program to clean up resources. So when the **with** block ends, it will call **close()** implicitly.

**camera.start\_preview()** turns the camera preview on and display it on a HDMI screen connected to the Pi. This allows us to see what the camera is looking at in real time.

**camera.capture(‘foo.jpg’)** captures an image of what the camera is currently looking at, and save that image in the specified directory. In this case, it will save it as foo.jpg in the same directory as the code.

**The final code**

What we want to achieve with the face recognition feature is real-time recognition.

So when the user clicks on the video button the camera preview will turn on and the stream will start. The camera will then take a picture in the background and send that picture for facial recognition. After getting the results of the recognition back, it will overlay the results on the screen.

Here is the function that will do exactly that.



The stream will start and the face recognition portion will continuously loop as long as the running flag is true. When the user clicks on the video button again, the will running flag reset and the loop will stop. The image that was used for the real-time recognition will be deleted.

An image will be captured every few seconds for recognition and after the recognition is completed, it will overlay the results on the stream.



**Threading**

When the user clicks on the camera button, the capture flag will be raised and the capture thread will begin.



The camera function is in a thread because we do not want the real time facial recognition to slow down. Spawning a thread will allow two process to run simultaneously.

The thread will reset the flag that called it, capture the current preview and then display the image on the GUI.

I assigned image to picture as **picture = image**, because of the way I defined **image**.



**image** will always be according to the time it’s taken at, and the **image** location will change each time it is called. To avoid any error, I call it once by assigning **picture = image** and continue from there.

This is the **do\_recognition** function that handle request and sorting.



Now all the functions for face detection is ready, but there is still an issue we have to solve.

When the button to start the video stream is clicked on the stream will start normally but the GUI will freeze because it is waiting for the stream function to end and there is no way to end the stream function since the buttons stop working as the GUI is not responding.

Like previously, we ran a capture thread so that the real-time facial recognition will not be interrupted. So the same solution will be implemented here.

**QThread**

In PyQt, there is a class we can use that runs a function as a thread. This is how it normally looks like.

****

This is how to initialise the class.



The Qthread will be ready in the background after initialisation.

The proper way to call **run()** is to use **self.Thread.start()**.

So here is the code with the implemented solution



****

**3.1.2 Functions – Speech Recognition**

Speech recognition was a breeze to code with, thanks to the **speech\_recognition** module written by Anthony Zhang and his team. This section will explain how the modules work and how it is implemented in the final code.

The education robot will be using Google Cloud API speech recognition services as it has a keyword function that greatly helps with the robot responses.

Just like previously, installing is easy with pip.



And the modules required for it.



The module contains almost all the common speech recognition API such as Sphinx, Google Cloud, and Microsoft Cognitive Services. It also handles audio recording and the request to the speech recognition services, and formats the responses from them.

This is the code with background listening using Google Cloud API.



Just like before, it will be bad if the GUI freezes so we need a thread to prevent this. The module provides a function that will basically loop between listening and recognition in the background.

Call it with **r.listen\_in\_background(microphone, callback)**. The microphone argument is a microphone instance created by the module, and the callback argument is the name of the function where recognition will take place. The thread will call that function once it stop listening. To end the thread, just call it again.

**Microphone**

The module creates a microphone instance with **Microphone()**. By default the instance will use the default microphone, record at a sample rate of 16000 and a chunk size of 1024.The settings for the microphone instance can be changed when calling it.

The first parameter is the device index, it selects the microphone to use for recording. I highly recommend to not leave this as empty. To list microphone devices, run the following code and it will print the devices in ascending order.



The second parameter is sample rate, the sample rate of the recordings affects how clear the recording is. The higher the sample rate, the clearer the audio. With clearer audio, the speech recognition will be more accurate when deciphering what the user is saying. In our program we use the sample rate of 44100 which is the sample rate of FLAC media files.

The third and last parameter is the chunk size, this is how sensitive the microphone is to ambient noise. A higher chunk size value helps avoid triggering on rapidly changing ambient noise, but also makes detection less sensitive. This value should be left as its default.

The microphone classes instances are designed to use within a context manger like so.



**Callback**

This is the function that will handle the speech recognition.

One feature about Google Cloud API is it preferred phrases feature. This feature will allow the preferred phrases to more likely be recognized over similar-sounding phrases. It is very useful in the testing phrases as keywords will be used more often than not to test the robot responses. The parameter for the keywords must be strings in a list.

The API requires a credential that can be obtained after creating a Google Cloud Platform account. There are instructions at <https://cloud.google.com/speech/docs/getting-started> and set up your project.

You have to create a project, enable billing for that project, enable Google Cloud Speech API for the same project and set up the service account key credentials for the very same project. You will get a JSON file containing the credentials. The text content of the file will be the credentials.

For more information on the **speech\_recognition** module read up the references at <https://github.com/Uberi/speech_recognition/blob/master/reference/library-reference.rst> .