

Smart Mirror with Drowsiness Detection

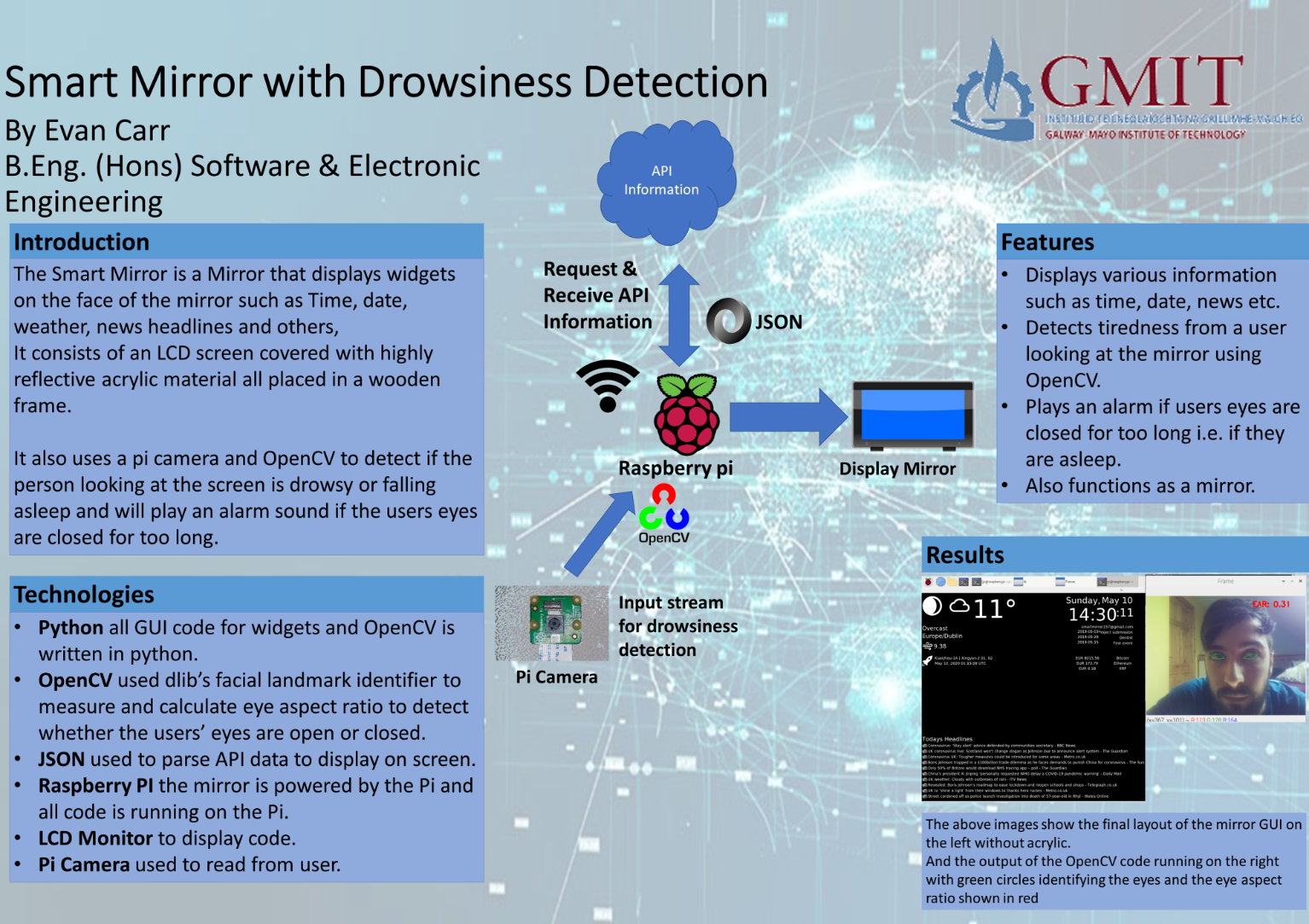
Evan Carr

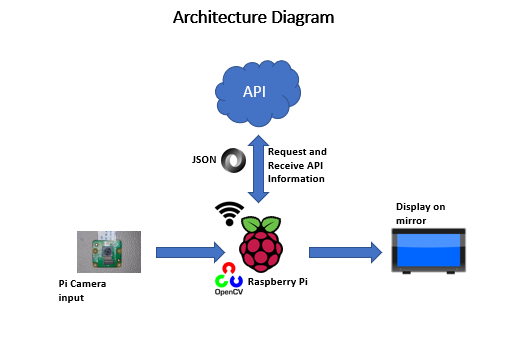
BEng (Hons) Software & Electronic Engineering

Galway-Mayo Institute of Technology

2019/2020

# Project Poster

****



**Declaration**

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering in Software & Electronic Engineering at Galway-Mayo Institute of Technology.

This project is my own work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Acknowledgements**

I would like to thank the lecturers who have helped me to complete my project throughout the year.

**Table of Contents**

[Project Poster 2](#_Toc40641812)

[1 Summary 7](#_Toc40641813)

[2 Introduction 8](#_Toc40641814)

[3 Project Architecture 9](#_Toc40641815)

[4 System architecture and development tools 10](#_Toc40641816)

[5 Weather API 11](#_Toc40641817)

[6 News Headlines 15](#_Toc40641818)

[7 Crypto Currency Prices 16](#_Toc40641819)

[8 Google calendar events 17](#_Toc40641820)

[9 Time 18](#_Toc40641821)

[10 Date 19](#_Toc40641822)

[11 Rocket launches 19](#_Toc40641823)

[12 Drowsiness Detection using open cv 20](#_Toc40641824)

[13 Problem Solving 23](#_Toc40641825)

[14 Conclusion 24](#_Toc40641826)

[15 References 25](#_Toc40641827)

# Summary

My project is a Smart Mirror with Drowsiness Detection. A smart mirror consists of a monitor covered with a piece of highly reflective acrylic material which is what returns your reflection, displaying on the mirror is a series of widgets showing information such as the time, date, news headlines, stock prices and other information. Essentially when looking at the mirror the user would see their own reflection along with this information shown on the face of the mirror which shines through the back of the acrylic material.

The other functionality is drowsiness detection using OpenCV and by using the Pi camera as the input stream, this works by identifying facial landmarks of the user and detecting when the eyes are closed, and if the eyes are closed for a long period of time it will sound an alarm to wake the user up.

All the code for this mirror is running on the raspberry pi and is written in python. To display the widgets on the screen of the mirror I designed a GUI that runs and continuously updates as it is running. To retrieve the information for the widgets I am using API’s. When a request is sent to the API it returns a JSON object. Then by parsing the object and selecting only the necessary information that is needed I can take these values and store them in variables, which are then used to display the information on the screen of the GUI.

# Introduction

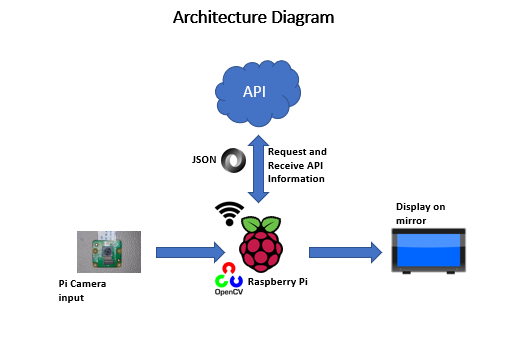
To give a quick Introduction into my report I will explain what it does from a high level. The Smart mirror is designed to function as a standard mirror would function. But also displayed on the face of the mirror there will be widgets displayed on the screen such as the time, date, news headlines, current weather, stock prices and others.

It also functions as a drowsiness detector by using a camera to monitor the users face when they stand in front of the mirror and detects if their eyes are closed, and if they are closed for a long enough period of time, it will play an alarm sound designed to wake the user up.

The reason I wanted to do this is I felt it would prove to be a challenging but also rewarding project for me to undertake. I think this mirror could also prove very useful as a mirror that customers could use in their own house. The technology used in this project is also highly adaptable and reusable and could serve many other purposes other than for a smart mirror.

# Project Architecture

The diagram below shows the components I am using and the interactions between them. The development platform I am using is the Thonny IDE for python, all running on the raspberry Pi, the Pi camera, and using OpenCV for the drowsiness detection code. I am also using a monitor for displaying the GUI. More detailed explanation can be found in the next section.



# System architecture and development tools

**Raspberry Pi 4 and Pi Camera**

I am running all of the code on the raspberry pi. The Pi 4 is the latest model of raspberry pi currently available and uses Raspbian Buster as its operating system. It is a lightweight yet fast and powerful tool for developing many projects. The Pi camera is used as the input stream for detecting the users face and the pi then runs the code using OpenCV libraries to detect when the eyes are closed. The raspberry pi is also running all the other mirror code simultaneously and all code is written in python.

**Python**

The programming language I am using for all my code is python. It is an efficient object-oriented programming language. The IDE I am using to write the code is called Thonny. Thonny is a lightweight but very useful tool for running python code which works well on a raspberry pi. Using Thonny I can create the GUI using Tkinter which is Thonny’s GUI development tool.

**OpenCV**

OpenCV is an open source computer vision library of programming functions mainly aimed at real time computer vision. It has a very wide range of functionality in the areas of image processing, facial recognition, and object detection. In my project I am using this facial recognition to detect when the user’s eye are open or closed.

**JSON**

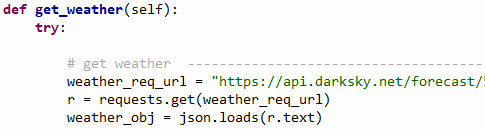
Using JSON to retrieve parse and load JSON objects that are retrieved from the API requests

**API**

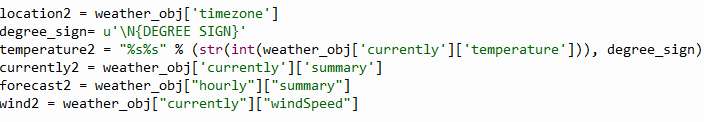
I am using various API’s and get requests to acquire the information that is displayed on the GUI

# Weather API

The weather API is acquired by first sending a request to the website by importing the requests library and sending a get request which will return a nested JSON object. Then I am using JSON to load the object into a text file that can be read and parsed. As seen in the below picture.

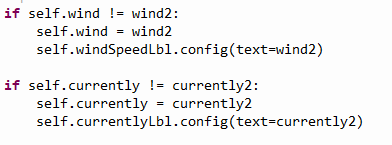


By running this code, I can view the object I will be returned with a full weather forecast for the day. From looking at the object I can find out what the specific information that I want to use instead of printing the entire list and by using its key value I can extract this information and print only this.



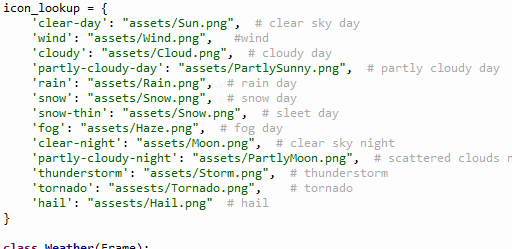
Using these variables which will now store the result of the returned object I can then begin to design the GUI and use the values to be printed in the frame of the GUI.



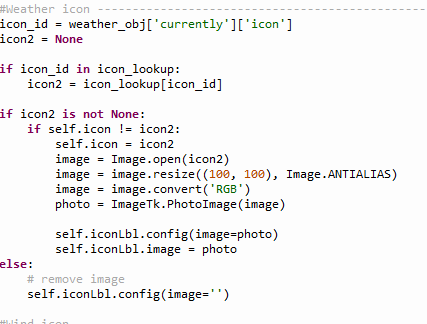


The above images show the construction of the GUI frame and the second photo shows how I am assigning the values of the object into the frame by updating the values of the variables.

I am also loading images of weather icons and the current moon phase in this class. To do this I am storing the file paths of the images in a python dictionary which uses key value pairs, the key being the type of weather e.g. sunny or cloudy, and the value being the file path to the image, then after retrieving the JSON object I am using this dictionary and some logic statements to find the image that corresponds to the type of weather in the JSON object and will attach and print that image on the frame of the GUI.



Above is the image of the dictionary containing images.

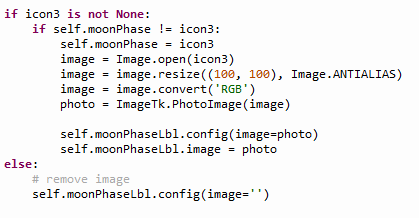


Above is the code that looks in the dictionary and attaches the image to the result of the API call. It also resizes the image to a 25\*25 size so it will fit beside the text on the GUI frame.

Also, for the moon phase there are eight general shapes or phases the moon will take. And the result of the API call will always return a value between 0 and 1. Using this I can use some simple if else statements and by dividing the value up into eights I can set the corresponding image of the moon to each if else statement.

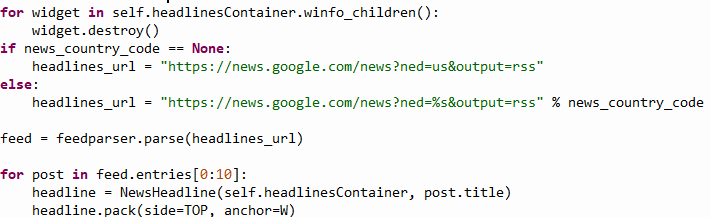


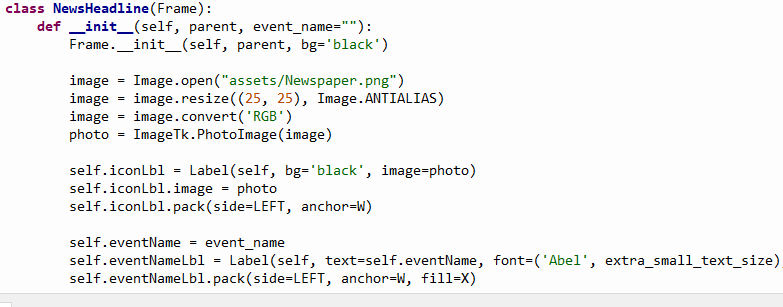
Then assigning these results into variables which will then be reassigned into the frame labels of the GUI



# News Headlines

The next API I am using is the news headlines. To retrieve this, I am sending a request to google news and using feed parser to extract the news headlines and place them inside the frame. I then iterate through the list and show only ten by using a for loop.

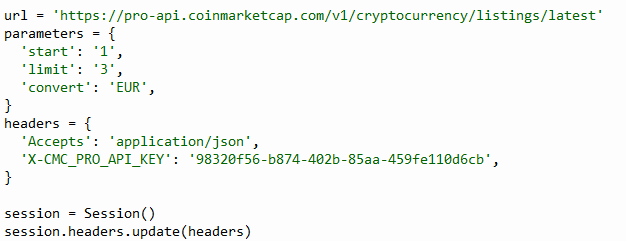




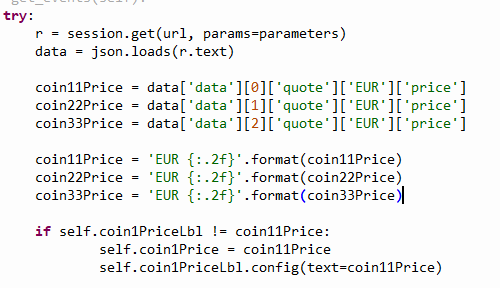
Here I am also attaching an image of a newspaper and showing it beside each headline when it gets printed.

# Crypto Currency Prices

I am also showing the top three most valuable crypto currencies in real time on the frame of the GUI. To do this I am using an API that contains such a list. I am calling this API and limiting the number of returned values as shown below. Where url is the website containing the API. Start and limit is the number of returned results.



This information then gets parsed and formatted using JSON and assigned to variables. Which are then used to be printed on the GUI as shown below.



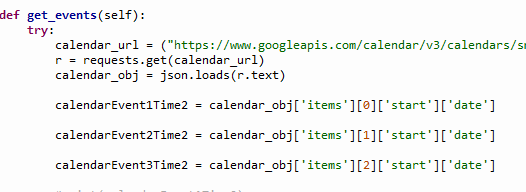
The if statement is taking the current value of the frame and reassigning it to the new value containing the result. The reason for this is when creating the frame an empty value must be instantiated in order for the frame to be created and then reassigned to the value with the result in it.

# Google calendar events

The google calendar events works by logging into a Gmail account and using this account along with an API from google that is designed to show upcoming calendar events. To do this I had to create events on the google account first. I also had to initialise the API by running code from the google api website. Once this was done fetching the results of the api call will return the list of events which can be parsed using JSON and displayed on the GUI frame.



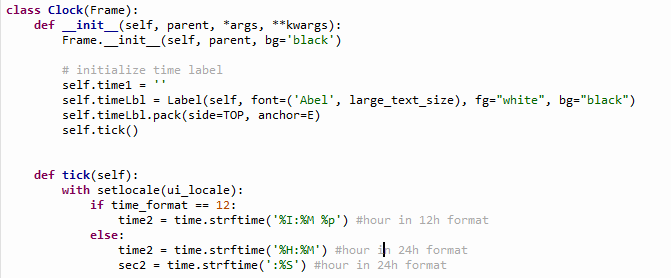
To display the time and date of the event I am using the same process and parsing the information from the object using JSON, but for layout purposes I had to put the time and date of the calendar event into a separate class so that I could have them printed on the same line and I had to use the grid layout using x and y coordinates to achieve this. The reason being is that if they were in the same class the time and date of the event would be printed on the line underneath the name of the event.



The above image is taken from the calendar time class and prints the time and date of the event.

# Time

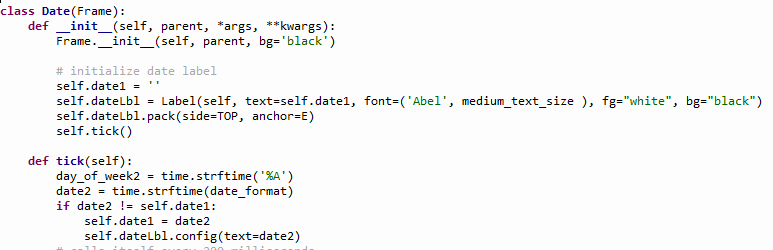
For displaying the time, I am importing the time class and setting variables to contain the time by calling the time function. It uses the time from the time on the raspberry pi. I then create the GUI by initialising a frame with the labels in and assigning the time variables to this frame of the GUI.



Also, for printing the seconds on the frame and having them increment dynamically I am using the same method although I had to put this into a separate class. to make the time tick I created a tick function that contains the time variables for the time and then within the frame function I am calling the tick function that updates the time every second.

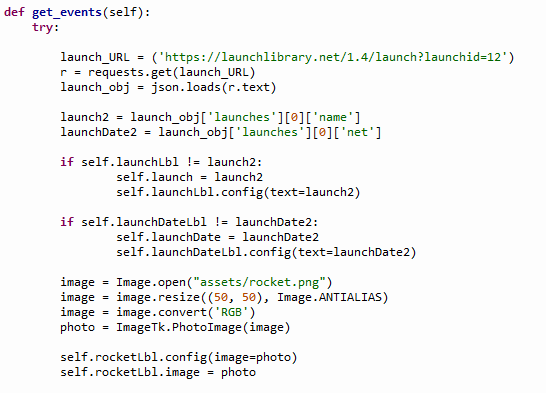
# Date

The date label is done the same way as the time function by importing the datetime class, creating a frame label, and then assigning the values containing the date into the frame and calling the tick function that will update the date when it carries over to the next day



# Rocket launches

The rocket launch is shown by calling an API from a website that contains all upcoming rocket events from all over the world, this returns a JSON object which can be then loaded into a text file using JSON and parsed, in my case I am only showing the next most upcoming event that is going to happen. The exact time and date of the event is shown along with an image of a rocket. Using the same methods as before I am storing the results of the API call into variables and reassigning these values into the frame labels of the GUI.

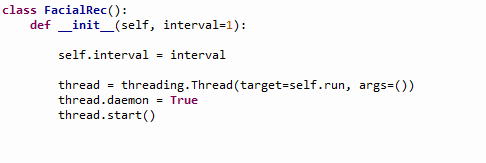


Above image showing the api call for the rocket launches.

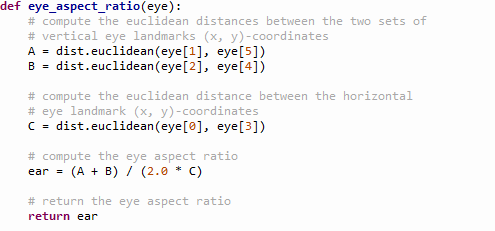
# Drowsiness Detection using open cv

This part of the code is the other aspect of the project and runs on a separate thread to the mirror frame code. What this code does is it uses the Pi camera as the input stream and begins looking for a face, once a face is found it then monitors only the eye regions and tracks whether or not the eyes are open or closed, if they are closed for a long enough period of time indicating that the user might be falling asleep, it will then play the sound of an alarm that would wake them up.

This works by using OpenCV for facial recognition and by using dlib to apply facial landmarks and find the eyes. Then using SciPy the calculate the Euclidean distance between the 2 eyes which can be used to calculate the eye aspect ratio. The eye aspect ratio is what indicates if the eyes are open as the value of the EAR will be constant when the eyes are open and will rapidly fall to zero when they are closed.

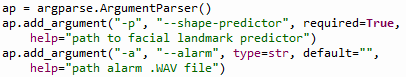


The first part of the code was to start a new thread as shown above so it could run simultaneously with the mirror code.



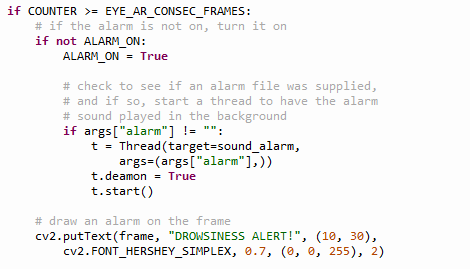
Next using the SciPy package to calculate the EAR. The relation between the vertical eye landmarks and the horizontal landmarks is related to the EAR.

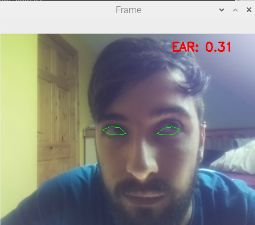
I also must use command line arguments when running the code to specify the file path for dlib’s shape predictor and the sound of the alarm. Shown below.





Once the code is running and the shape predictor is loaded, I can use dlib and the utils package to start looking for a face and apply facial landmarks to measure the eyes. This code will run continuously until shut down, and by using the values now calculated and stored into variables the next part is to play the alarm sound once the eyes are closed for a long enough period of time. To do this I am incrementing a counter any time the eyes remain closed and do not open again. And once the counter reaches or exceeds the threshold variable. A new thread is started that will play the alarm sound





Above is the output of the code when running with the eyes found and highlighted in green and the eye aspect ratio value shown in red.

# Problem Solving

Some problems I had was first figuring out how to retrieve the JSON objects from API calls and to parse them, I did not know how to do this at first but I learned by doing some research, I then had to look at the nested JSON object to identify the values I wanted.

Another issue was organising the layout of the final Mirror frame as some of the objects were being printed in the incorrect place so I had to adapt to using a grid system for the layout and also separating some of the API’s into subclasses so I could get them to print in the same area.

I also ran into trouble when trying to run the Pi camera and the mirror frame at the same time so I had to incorporate threading and run the code on separate threads so it could all function together.

# Conclusion

In conclusion the project has been completed successfully and is fully functional. A prototype model has been built and could potentially be a product that customers might want to buy, the functionality of the OpenCV code is also highly adaptable and reusable and could be used for other purposes. I have learned a lot from undertaking this project and I feel it was very worthwhile doing it and I enjoyed working on it.

# References

[1] DarkSky.net. Weather API [Online]. Available: <https://darksky.net/dev/docs>

[2] Dr. A. Rosebrock, “*Drowsiness Detection with OpenCV”*, [Online], May 8, 2017. Available. <https://www.pyimagesearch.com/2017/05/08/drowsiness-detection-opencv/>

[3]coinmarketcap.com, “*Crypto currency API*” [Online], Available <https://coinmarketcap.com/api/documentation/v1/>

[4] Google Inc, “*Google calendar API Python Quickstart”*, [Online], 21 April 2020. Available <https://developers.google.com/calendar/quickstart/python>

[5] Original example code[online] Available <https://github.com/HackerShackOfficial/Smart-Mirror>

[6] My source code <https://github.com/ECARR22/Project_Y4>