Oli Thompson

#Dyson School of Design Engineering | MEng Design Engineering

Module Exam

Module code and Name	**DE4-SIOT Sensing & IoT**
Student CID 01228536	
Assessment date	**10th Jan 2020**

Presentation URL (publicly accessible link):

Presentation for Mood Tracking Smart Mirror:
https://youtu.be/v26QM182Wlo

Backup link Presentation for Mood Tracking Smart Mirror:
https://imperiallondon-my.sharepoint.com/:v:/g/personal/ot316_ic_ac_uk/EV114BXF3rhElbOUsoqaXwQBdBNAbihL9ju6pHHqzciGaQ?e=PSXd0Y

Presentation for ISS Tracker:
http://www.olithompson.com/ISS_tracker.php

Code & Data (publicly accessible link):

Code for Mood Tracking Smart Mirror:
https://github.com/ot316/Mood-Detection-Smart-Mirror-data-logging-and-Web-Server

Data for Mood Tracking Smart Mirror:
https://oli-thompson-siot.s3.eu-west-2.amazonaws.com/Mood_Recognition_Dataset.csv

Code for ISS Tracker:
https://github.com/ot316/ISS-Tracker

Oli Thompson

ISS tracker and Mood Tracking Smart Mirror

Coursework 1: Sensing

Introduction and Objectives

This sensing and internet of things coursework project has been split into 2 sub-projects; The ISS tracker and the Mood Tracking Smart Mirror. For each section of this report I will examine both sub-projects in sequence. For readability, the ISS tracker sections are on a grey background. I have included minimal code in this report and will instead link to the relevant files on Github, which have been structured as separate files for readability.

ISS Tracker

The first sub-project involves retrieving the position of the International Space Station (ISS), displaying the positional data on a world map and displaying numeric values such as altitude and speed. The display is configured within a page of my existing portfolio web app. The project also includes a Raspberry Pi computer running a script that sends me an email whenever the ISS is directly over London.

The main objectives of this sub-project were to create a robust visualisation of ISS data, display it on a publicly accessible webpage and to verify its accuracy by comparing it to NASA's own implementation on their website. I wanted to implement polling of the API in both Python and Javascript, to build experience with lower level languages, and to experiment with Python's smtp email library.

Mood Tracking Smart Mirror

The second sub-project is more complex. I designed a neural network and trained it on a dataset of faces, each labelled with 5 categorical emotions. With this model saved, I designed and built a raspberry-pi powered "smart mirror" that uses a camera to retrieve image data. This data is processed and fed into the neural network model and should it detect an emotion, the data is timestamped and saved to a csv file. The file is backed up to an AWS S3 bucket every hour.

The Raspberry Pi runs a webserver that visualises the data on a locally accessible website that is updated automatically. The webpages are interactive and allow the number of displayed data-points to be set by the user to allow insights to be gathered more easily. The data is processed in real time on the client device using Javascript, allowing the user to choose the timescale over which to view the data, for example average readings per second, minute, hour or day in order to give the user a better idea of their changing mood. The website includes a download link to retrieve the raw CSV data. The Raspberry Pi also runs a popular smart mirror software called MagicMirror [link], this displays useful information through an LCD panel from an old laptop. I setup an account with openweather [link] to retrieve weather information for London, set the calendar module up with my imperial college webcal and configured it to read headlines from the BBC News RSS feed. The mirror also displays the London tube service status from the TFL API [link], the bitcoin value from the coinbase API [link], information from my spotify account using the spotify API [link] and finally information about my 3D printer running octoprint using the local octoprint API. The screen also came in useful for debugging.

The objectives of this sub-project were to become familiar with the python libraries typically used for convolutional neural networks; to pre-process data, train a network and apply it to real-world data in order to verify the networks accuracy; to build an interactive Javascript powered web page to display real-time data in a useful and aesthetically pleasing way; and finally to gather meaningful insights from the data. The idea behind using a smart mirror to collect data is based of the premise that a user would stand infront of the smart mirror each morning, and this would provide a non-obtrusive method of data collection that would not interrupt the morning routine.

Data Sources and Setup

ISS Tracker

Web visualisation and Javascript Implementation

The web visualisation uses an API available at http://api.open-notify.org/iss-now.json later replaced by https://api.wheretheiss.at/v1/satellites/25544, because it was not possible to load the unsecured http request inside an SSL secured webpage without it being blocked by Google Chrome's Cross Origin Read Blocking. (This is to stop a client device from loading a malicious resource) [see code]
The secured API is read inside a Javascript file using the ajax request function of the jQuery library. The data is returned as a json file which can be read natively in Javascript. The altitude, velocity, latitude and longitude are saved as variables [see code].

Python Implementation and Email Bot

A Raspberry Pi zeroW was setup and installed into a 3D printed case. It runs a continuous script on boot with the intention of 24/7 operation (Figure 1).
The unsecured API was read inside the Raspberry Pi script using Python's built in 'urllib' library. The data is returned as a json file and is parsed using Python's json library before being returned as a dictionary object. The longitude and latitude are saved as variables. [see code]
The Gmail email is setup with its own Gmail account credentials using the 'smtplib' Python library, with TLS encryption. [see code]

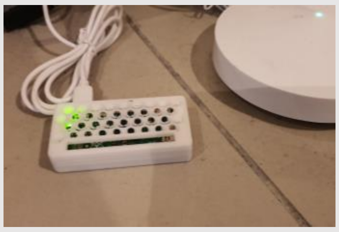


Figure 1: Raspberry Pi ZeroW

Smart Mirror

Hardware build

The smart mirror is built from laser cut 5mm acrylic with the front face made from a 3mm 2-way mirrored acrylic that allows light from a screen to pass through from behind, whilst maintaining a mirrored surface on the front. A Raspberry Pi camera is mounted behind the 2-way mirror as well as a 17" HD LCD screen that was upcycled from an old laptop. The LCD screen is driven by M.NT68676 driver board (Figure 2) that provides an HDMI input. The camera is plugged into a Raspberry Pi 4 that is mounted behind the mirror, its micro HDMI output is plugged into the LCD driver board. To power the mirror a 100W LED 12V power supply is fixed behind the mirror, allowing the smart mirror to be powered directly from the mains. The 12V rail supplies the LCD driver board directly, and a 12V to 5V step down converter can deliver up to 3A to the Raspberry Pi 4 (Figure 3). The Raspberry Pi was initially installed with passive cooling achieved with small heatsinks attached to the SOC with thermally conductive tape. Later in the project it became apparent that active cooling was required, as the Raspberry Pi would regularly display high temperature warnings. A small 5V fan was attached to the Raspberry Pi's enclosure.

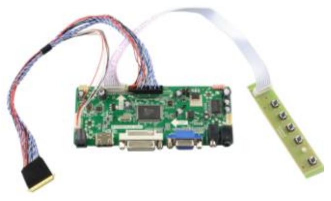


Figure 2: LCD Inverter and Driver Board

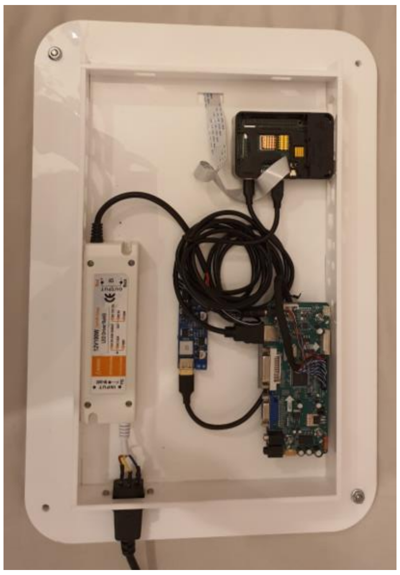


Figure 3: Electronic Components

Software setup

The smart mirror's neural network is trained from a CSV dataset from Kaggle.com [link]. The dataset contains just under 36000 data points, each one consisting of a string of 1600 pixel values, representing a 40*40 pixel grayscale image (Figure 3) and tagged with one of the emotions of neutral, fear, anger, happy, sad and disgust.

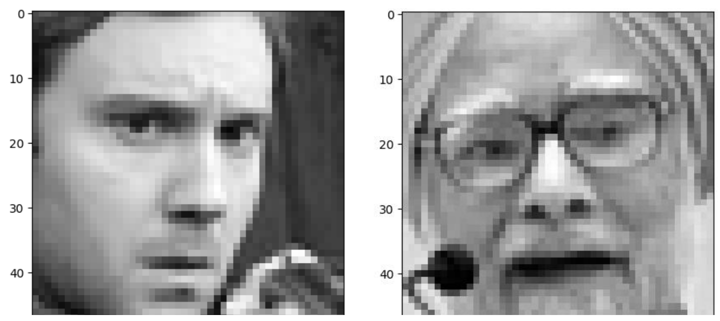


Figure 3: Random Rendered Examples from Dataset

The dataset was loaded into the first python script (Preprocess_DataSet.py) using the 'Pandas' library. The first column was loaded into the 'Pandas' dataframe object, however the column containing the pixels needed to be parsed manually as the values were separated by spaces rather than columns. I split the dataset into 2 parts and used multithreading to process both halves concurrently to save time.

Once the data has been split into training and testing data using the 'train test split' function from the Python library 'sklearn', It is normalised by subtracting the mean and dividing by the standard deviation. The first script concludes by saving the 'numpy' arrays to the root directory. [see code]

The second script 'Mood_Recognition_train_CNN.py' loads the arrays and defines the number of features and labels. The sequential neural network is built using the Python libraries 'tensorflow' and 'keras'. It is comprised of a linear stack of 4 layers: Convolutional kernels of 3*3 pixels are applied to the input layers in sequence, batch normalisation is used to normalise the output of each layer, and the max pooling function samples and reduces the data to reduce computational complexity. Finally, the model is flattened, compiled and saved as json file, with the respective weights saved in an h5 format. [see code].

MagicMirror

Installing the MagicMirror software was simple, it uses Javascript and npm. Adding weather data was achieved by creating an account on openweather, finding my location ID and configuring the MagicMirror config file with my API key. After disabling some unnecessary modules I retrieved my Imperial calendar .ics file from the Imperial College London website and installed it, and copied across the url of the UK BBC news RSS feed [link] to the news module. I configured the TFL API to display the London tube service and used the coinbase API to show the fluctuation in bitcoin prices. I then setup a new app in my spotify account and generated an API key which I used to visualise live information from my spotify account. I generated an API key for my octoprint instance running on a separate Raspberry Pi connected to my 3D printer. This allowed me to display the video stream of my printer on the mirror using by pointing it to the IP address of my octoprint instance. Finally I configured the HDMI display to be controllable with an Amazon Alexa, by emulating a Wemo device.

Data collection and storage process

ISS Tracker

The data returned from the API is in the format of latitude and longitude. In order to display the data on a 2D plane this data must be transformed. It is not possible to accurately map a spherical surface into a 2D plane, and as such different approximate projections exist [1], for example the mercator projection (Figure 4), which preserves the bearing of lines but heavily distorts area, the Azimuthal equidistant (Figure 5) that preserves distances from the North Pole or the Goode homolosine (Figure 6) that preserves area but has discontinuity between the continents.

I have chosen to use the Mercator projection because the transformation from polar (latitude and longitude) to cartesian is mathematically simpler and it is a commonly observed projection.

Figure 4: Mercator Projection

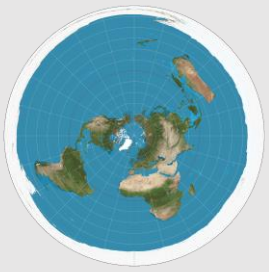


Figure 5: Azimuthal Equidistant Projection

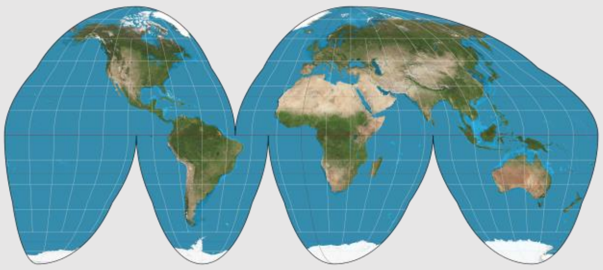


Figure 6: Goode Homolosine

The equation for the transformation [2] is shown in equation 1 and 2, where λ = longitude and φ = latitude:

Equation 1: $$x = \lambda$$

Equation 2: $$y = \ln(\tan(\frac{\pi}{4} + \frac{\varphi}{2}))$$

The formulas above are implemented in Javascript to scale the values dynamically inside the dimensions of the map. [see code]. The transformation is not necessary for the python implementation, as its purpose is to compare the current latitude and longitude to a predefined longitude and latitude interval over London. [3]

Smart Mirror

Python

The 3rd python script 'main.py' runs on the Raspberry Pi. Firstly, it loads the neural network model and initialises the video stream from the attached camera using the 'OpenCV' library. It then sets up a classifier built into OpenCV and uses the 'haar_cascades_frontal_face' XML file to identify faces [link]. This function is used to crop the video stream to contain only the face, otherwise the image passed to the neural network would not be in the same format as the test data. The script first tries to open the existing csv in order to append data to it. If an error is thrown, the script will create a new CSV instead and open it.

The script now enters a while loop where it saves a frame from the camera, crops it to contain only the face, converts it to grayscale and down-samples it to 40*40 pixels. The purpose of this process is to regularise the data to match the test data. Each pixel was originally scaled by dividing it by 255, in order to normalise it between 0 and 1. This assumed that the brightest pixel had a value of 255 and I later changed this to a min-max scaling algorithm, as this worked better in low light, where the brightest pixel may have a lower value. The data is now fed into the neural network and 6 numbers are returned. These represent the network's confidence that the frame should be labelled with each emotion: The largest of these values is the predicted emotion. The script discards any neutral classifications as this data was not particularly useful. A new data-point is generated as well as the date and time (using Python's 'datetime' module) and the value of each emotion. After 10 successful readings, the data point is appended to the dataframe object and saved as a csv, overwriting the previous file. Saving data is restricted to once every 10 readings to improve code efficiency. The data is saved in the 'emotion_data.csv' file in the 'webserver' directory. This isolates the Python back end from the webserver's front end.

For debugging purposes, the script can be run with a '-show' argument from the command line, this was achieved using Python's 'sys' library. This opens an OpenCV video window that shows the resized live image from the camera with the subject's face identified by a square and the predicted emotion written over the top (Figure 7). The labelled emotion will update immediately and proved useful for tuning the performance of the smart mirror. To close the window a keyboard interrupt is assigned to the letter 'q' [see code].

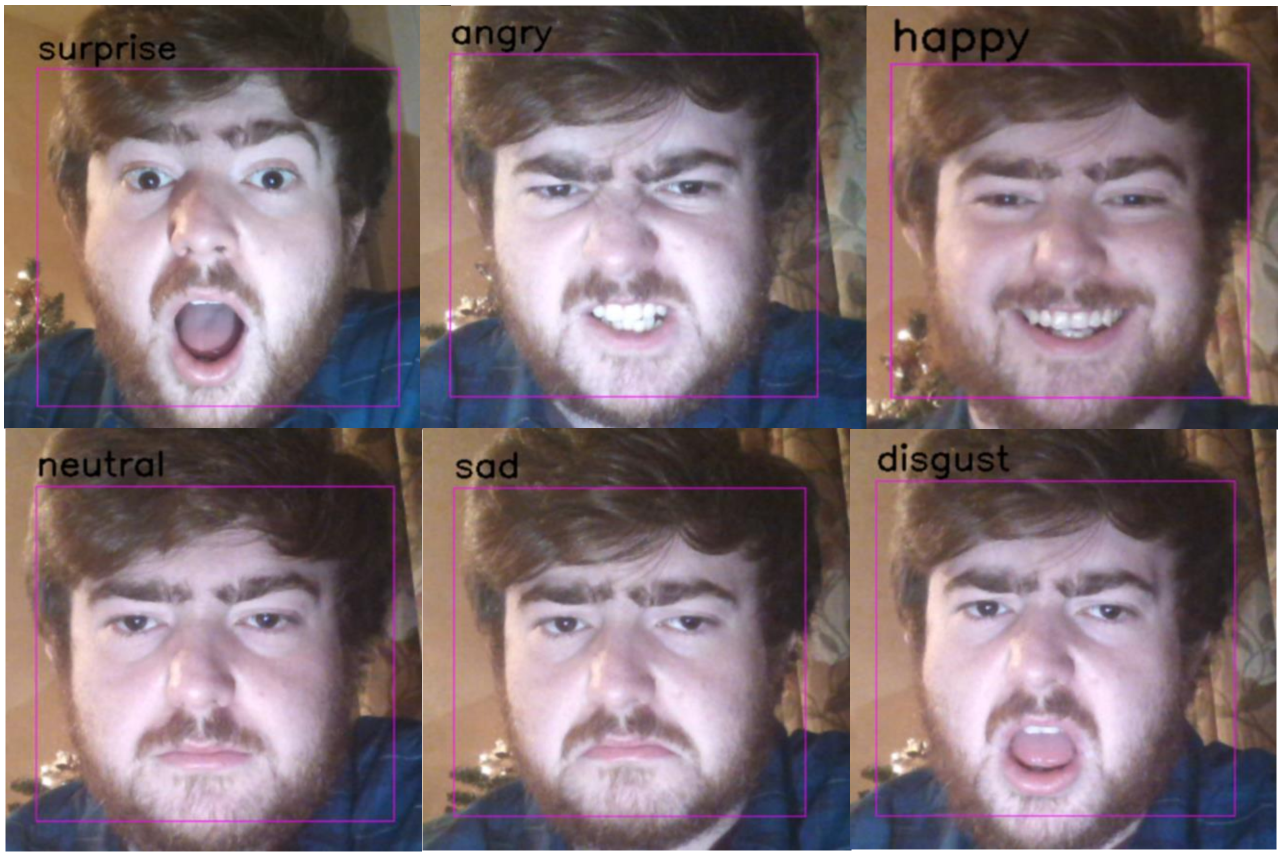


Figure 7 Emotion Recognition Test

I created a new IAM user in my AWS account, that was given access to an AWS S3 bucket that I setup previously. I copied the credentials of the account into the environment variables of the AWS CLI, allowing me programmatic access to my S3 bucket. I wrote a Python script that uploaded the dataset to the S3 bucket, which I later made publicly accessible. To automate this process, I setup the Raspberry Pi to execute the script every hour using a cron job by adding it to the crontab file.

Javascript

With the CSV file of emotion data saved by the Python script, further data processing must be carried out before it can be displayed in a useful way. Rather than processing this data in the continuously running Python 'main.py' and adding more computational strain on the Raspberry Pi, I chose to carry out the data processing in Javascript, that would run on the client-side device in the web browser. This proved to be much more challenging as it lacked the advanced libraries supported in Python, and I had to write the functions from scratch. I learnt that the reason for this is because Javascript is not usually used for numeric, scientific or data applications, although it is significantly faster than Python as it is a lower level language.

The Python script continuously saves datapoints in quick succession as long as a face is in view. It is not particularly useful to observe emotion data changing every few milliseconds and as such the data must be averaged over a certain timescale.

The script contains a 'graph2' function, that begins by parsing the csv file into a 2-dimensional Javascript array. Javascript does not natively support 2-dimensional arrays, but an array of arrays can be constructed with nested for loops. I used the 'Papaparse library' [link] to parse the CSV, which is well suited for large files. Once each array had been pushed to the larger database array, I stripped the datetime string of the un-needed milliseconds value, by discarding the last 7 characters of the string. This later became the method by which I achieved variable averages over different timescales; days, hours, minutes, seconds: I simply removed 16, 13, 10 or 7 characters from the datetime string, essentially labelling datapoints that shared a timestamp with the same index, depending on what the value of the 'timescale' variable was when the graph function was run.

The function iterated through the datetime column of the database and created a new array of each unique value, called ‘uniquetimes’. I then created an array for how many values were labelled for each unique datetime named ‘countarray’ as well as a ‘cumulativecountarray’ that was generated by performing a cumulative addition on ‘countarray’. This was used for indexing the database. The code segment below shows how these arrays are used to average all the sets of values accordingly and save them to a new array. This process works regardless of which timescale is specified previously.

```
for (var i = 0; i < uniquetimes.length; i++) {
      for (var j = cumulativecountarray[i]; j < cumulativecountarray[i + 1]; j++) {
            for (var k = 1; k < 6; k++) {
                  average[i][k] += parseFloat(array[j][k]);
                  average[i][k] = average[i][k] / countarray[i + 1];
```

Before the data is plotted, each array is normalised between 1 and 0 with an element-wise subtraction of the minimum value and division by the range. [see full code]

Basic characteristics of the end-to-end systems set up and data

Smart Mirror

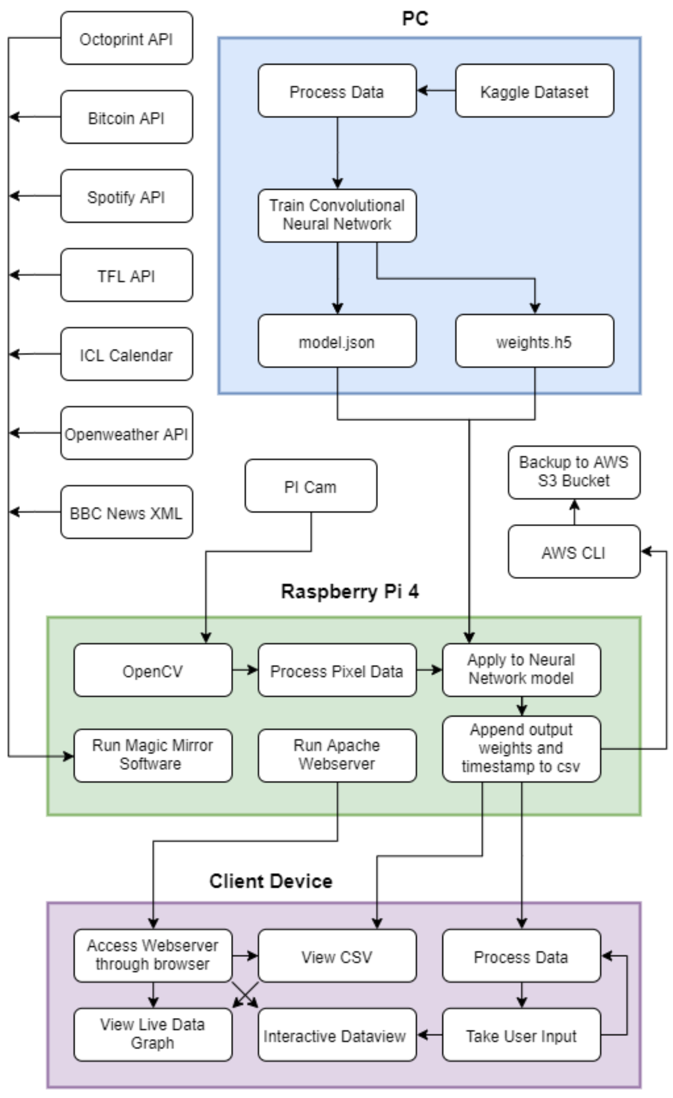


Figure 8: Flow Diagram of the Smart Mirror System

ISS Tracker

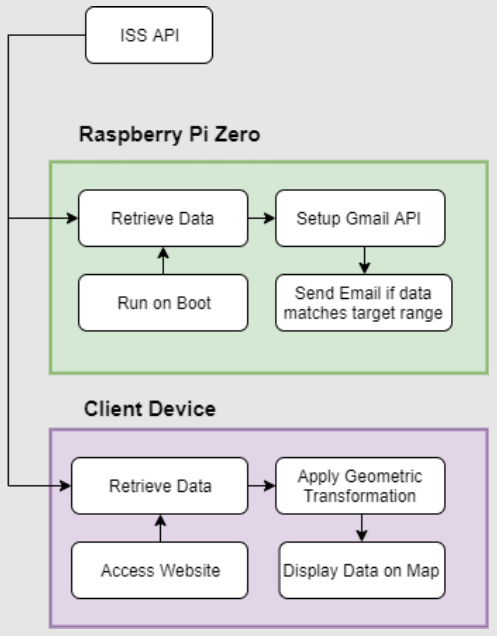


Figure 9: Flow Diagram of the ISS Tracker System

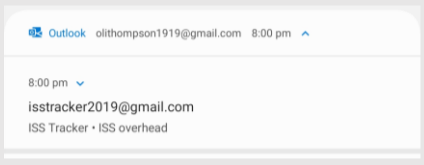
Coursework 2: Internet of Things

Data Interaction/visualisation/actuation platform

ISS Tracker

Python Implementation and Email Bot

The data actuation for the Raspberry Pi first involves comparing the constantly updated latitude and longitude data from the API to a region over London [3]; this is done with a simple if statement. If both the latitude and longitude are within the defined interval an email is sent to my personal email address with the message 'ISS Overhead' (Figure 10). [see code]


Outlook olithompson1919@gmail.com 8:00 pm

8:00 pm

isstracker2019@gmail.com

ISS Tracker • ISS overhead

Figure 10: Email Notification from ISS Tracker

Web visualisation and Javascript Implementation

The webpage for the visualisation of the ISS position is built into a page on my existing web-app. The PHP file contains mostly static HTML and some PHP code for templating. (please note the ISS tracker GitHub repository contains only the HTML, CSS and Javascript to allow it to be viewed independently to the WebApp). The HTML links to a CSS style sheet, some necessary Javascript Libraries and the 'ISStracker.js' Javascript file. [see code] The ISS tracker file firstly retrieves the map image and graphic of the ISS via an https request to an AWS S3 bucket inside the 'preload' function. The images are then saved as static variables. this function is called when the page is loaded. Next the setup function defines the dimensions of the map and uses the p5 library to create a canvas on the page. The function also centres the canvas on the webpage.

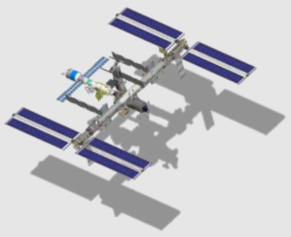
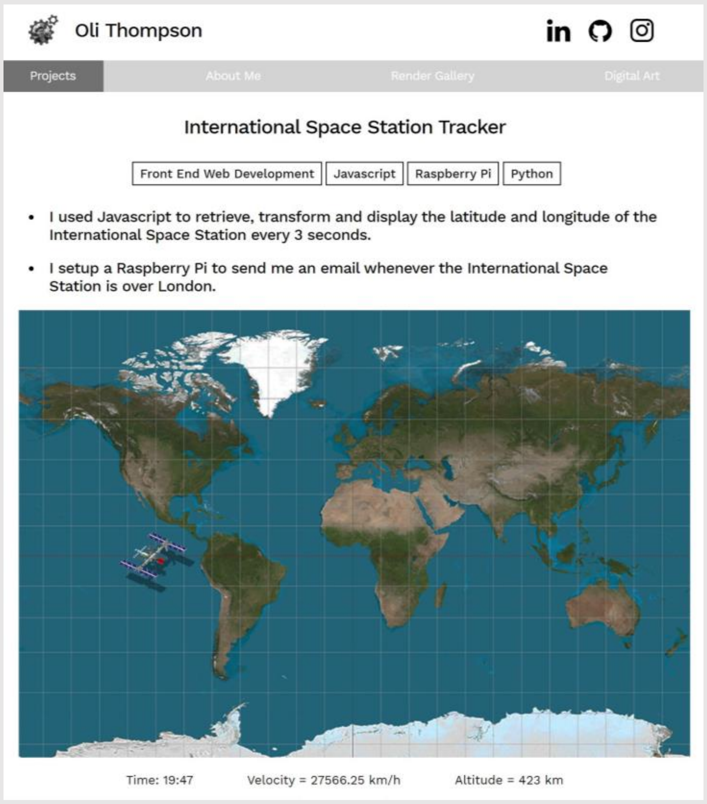


Figure 11: ISS Graphic



International Space Station Tracker

Front End Web Development | Javascript | Raspberry Pi | Python

- I used Javascript to retrieve, transform and display the latitude and longitude of the International Space Station every 3 seconds.
- I setup a Raspberry Pi to send me an email whenever the International Space Station is over London.

Time: 19:47 Velocity = 27566.25 km/h Altitude = 423 km

Figure 12: ISS Web Interface

The plot function is called within setup and processes the data from the API as explained previously. It then clears the previous picture and sets the background of the canvas to the map picture obtained in 'preload'. The variables x and y that were returned from the transformation code are used as coordinates to 'draw' a small red circle on the canvas, this is because the ISS graphic (Figure 11) is quite large and it is difficult to see the precise location. These same variables are used to place the ISS image onto the canvas, offset by half its width and height in order to centre it.

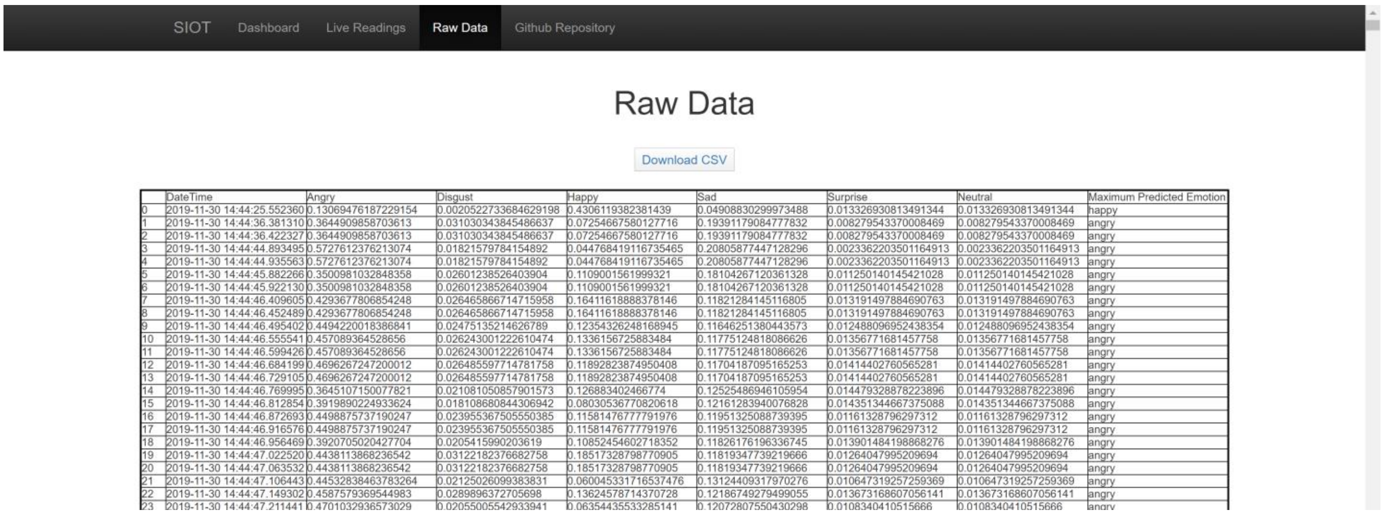
The current date and time are called using the 'Date()' function and passed to a string variable before being concatenated to the altitude and speed data. This string is then set to an HTML tag using the built in 'innerHTML' function.

The plot function is executed continuously at intervals of 3 seconds using the 'setInterval' function. If the window is resized at any time a the 'windowResized' function will be called, which simply repositions the Canvas to the middle of the page as standard CSS would not work. The finished web interface is shown in Figure 12. [see code]

Smart Mirror

The smart mirror hosts an apache2 webserver accessible over the local network at ‘smartmirror.local’. The webserver hosts a small static local website with simple CSS style sheet and 3 HTML pages that can be accessed using the navigation bar (this also has a link to the GitHub repository). I used a bootstrap template to save time with making the HTML responsive to different devices. [see code]

Going through the pages in order of increasing complexity and following on from the Javascript data collection and storage subsection: The 3rd page simply converts the whole CSV database into an html table for viewing (Figure 13). There is also a download button that allows the entire database to be downloaded to the client device; this was useful for backing it up because GitHub was unable to handle the file once it became larger than 100mb [see code]. Later when the AWS S3 automatic backup method was implemented, this was no longer an issue.



Raw Data

	DateTime	Angry	Disgust	Happy	Sad	Surprise	Neutral	Maximum Predicted Emotion
0	2019-11-30 14:44:25.552360	0.13069476187229154	0.0020522733684629198	0.4306119382381439	0.04908830299973488	0.013326930813491344	0.013326930813491344	happy
1	2019-11-30 14:44:36.381310	0.3644909858703613	0.031030343845486637	0.07254667580127716	0.19391179084777832	0.008279543370008469	0.008279543370008469	angry
2	2019-11-30 14:44:36.422327	0.3644909858703613	0.031030343845486637	0.07254667580127716	0.19391179084777832	0.008279543370008469	0.008279543370008469	angry
3	2019-11-30 14:44:44.893495	0.5727612376213074	0.01821579784154892	0.044768419116735465	0.20805877447128296	0.0023362203501164913	0.0023362203501164913	angry
4	2019-11-30 14:44:44.935563	0.5727612376213074	0.01821579784154892	0.044768419116735465	0.20805877447128296	0.0023362203501164913	0.0023362203501164913	angry
5	2019-11-30 14:44:45.882266	0.3500981032848358	0.02601238526403904	0.1109001561999321	0.18104267120361328	0.011250140145421028	0.011250140145421028	angry
6	2019-11-30 14:44:45.922130	0.3500981032848358	0.02601238526403904	0.1109001561999321	0.18104267120361328	0.011250140145421028	0.011250140145421028	angry
7	2019-11-30 14:44:46.409605	0.4293677806854248	0.026465866714715958	0.16411618888378146	0.11821284145116805	0.013191497884690763	0.013191497884690763	angry
8	2019-11-30 14:44:46.452489	0.4293677806854248	0.026465866714715958	0.16411618888378146	0.11821284145116805	0.013191497884690763	0.013191497884690763	angry
9	2019-11-30 14:44:46.495402	0.4494220018386841	0.02475135214626789	0.12354326248168945	0.11646251380443573	0.012488096952438354	0.012488096952438354	angry
10	2019-11-30 14:44:46.555541	0.457089364528656	0.026243001222610474	0.1336156725883484	0.11775124818086626	0.013567716814577758	0.013567716814577758	angry
11	2019-11-30 14:44:46.599426	0.457089364528656	0.026243001222610474	0.1336156725883484	0.11775124818086626	0.013567716814577758	0.013567716814577758	angry
12	2019-11-30 14:44:46.684199	0.4696267247200012	0.026485597714781758	0.11892823874950408	0.11704187095165253	0.01414402760565281	0.01414402760565281	angry
13	2019-11-30 14:44:46.729105	0.4696267247200012	0.026485597714781758	0.11892823874950408	0.11704187095165253	0.01414402760565281	0.01414402760565281	angry
14	2019-11-30 14:44:46.769995	0.3645107150077821	0.021081050857901573	0.126883402466774	0.12525486946105954	0.014479328878223896	0.014479328878223896	angry
15	2019-11-30 14:44:46.812854	0.3919890224933624	0.018108680844306942	0.08030536770820618	0.12161283940076828	0.014351344667375088	0.014351344667375088	angry
16	2019-11-30 14:44:46.872693	0.4498875737190247	0.023955367505550385	0.11581476777791976	0.11951325088739395	0.01161328796297312	0.01161328796297312	angry
17	2019-11-30 14:44:46.916576	0.4498875737190247	0.023955367505550385	0.11581476777791976	0.11951325088739395	0.01161328796297312	0.01161328796297312	angry
18	2019-11-30 14:44:46.956469	0.3920705020427704	0.0205415990203619	0.10852454602718352	0.11826176196336745	0.013901484198868276	0.013901484198868276	angry
19	2019-11-30 14:44:47.022520	0.4438113868236542	0.03122182376682758	0.18517328798770905	0.11819347739219666	0.01264047995209694	0.01264047995209694	angry
20	2019-11-30 14:44:47.063532	0.4438113868236542	0.03122182376682758	0.18517328798770905	0.11819347739219666	0.01264047995209694	0.01264047995209694	angry
21	2019-11-30 14:44:47.106443	0.44532838463783264	0.02125026099383831	0.060045331716537476	0.13124409317970276	0.010647319257259369	0.010647319257259369	angry
22	2019-11-30 14:44:47.149302	0.4587579369544983	0.0289896372705698	0.13624578714370728	0.12186749279499055	0.013673168607056141	0.013673168607056141	angry
23	2019-11-30 14:44:47.211441	0.4701032936573029	0.02055005542933941	0.06354435533285141	0.12072807550430298	0.01083404105156666	0.01083404105156666	angry

Figure 13: Raw Data Webpage

The purpose of the ‘live reading’s webpage is to see the graph update in real-time and view each datapoint as it is generated. This serves little purpose for actually gathering useful data trends but is a good visual tool to show how the system works and it is very satisfying to be able to control the graph in real time with your face. The advantage of the graph over the OpenCV test view is that each emotion is always displayed, rather than only showing the single highest predicted emotion.

The user can adjust how many datapoints are displayed by entering a number in a text field, the ‘Chart1.js’ file is responsible for plotting the data, initially a ‘setup1’ function checks the user text field, if it is empty a default variable of datapoints is initialised as 100. This function concludes by calling the ‘graph function’ that takes the datapoints variable as an input. After parsing the csv data and normalising it between 0 and 1; the graph function pushes each emotion into individual arrays, ready to be plotted. I used ‘chart.js’ which provides a good level of customisation with attractive animations and interactive graphs, where the user can hover over datapoints to view them or hide lines by clicking on the legend. I plotted each emotion with a different colour and constrained the axes between 0 and 1. In order to update the graph in real time, the setup function is run every 1.5 second using the built in ‘setinterval’ function. This number was chosen as it was a good balance between low latency and efficiency. As the graph is constantly being redrawn, it will continuously check for user input in the text field using ‘getElementByID’ and will update the number of datapoints displayed automatically, scaling the graph accordingly (Figure 14) [see HTML code] [see Javascript code].

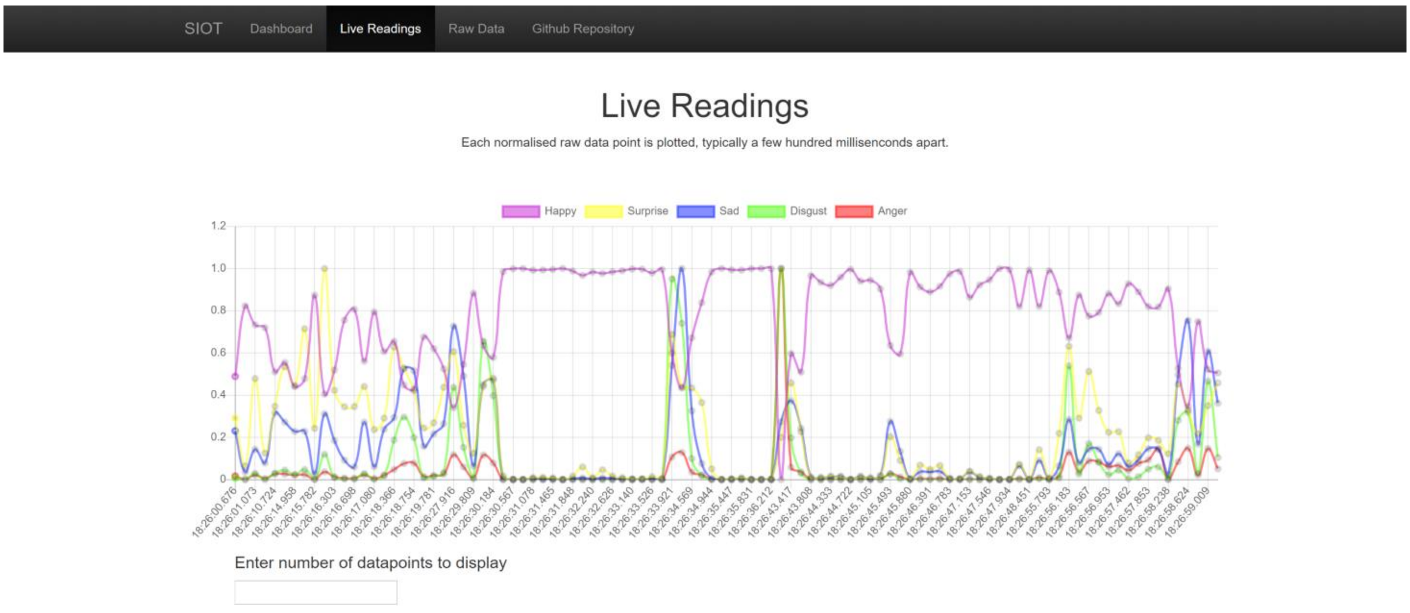

SIOT
Dashboard
Live Readings
Raw Data
Github Repository
Live Readings
Each normalised raw data point is plotted, typically a few hundred millisenconds apart.
Happy
Surprise
Sad
Disgust
Anger
Enter number of datapoints to display

Figure 14: Live Reading Webpage

The final page of the local website is the default page when the user visits the webserver: It is the dashboard where the user can view emotion data over different timescales, to gather insights into their changing mood. Aside from the data processing, the graphing process is similar to what is implemented in the 'live readings' webpage, with the Javascript file renamed as 'Chart2.js'. As explained previously, the data can be averaged over different time scales, including seconds, minutes, hours and days depending on which HTML button Is clicked; the button simply calls the 'graph2' function with an extra integer input argument corresponding to each button. Essentially the script will take averages of all data collected in the same day, hour etc, and display it with the data from other days/hours etc to give the user greater insights into their changing mood. The data is sliced to contain a default of 100 datapoints, or less if 100 data points do not exist for the given timescale (e.g. data has not yet been collected over 100 days) and plotted on the webpage. The input text field allows the user to control how many datapoints are shown, essentially giving them the ability to 'zoom in' on a specific time. Adjacent to the text field is an HTML

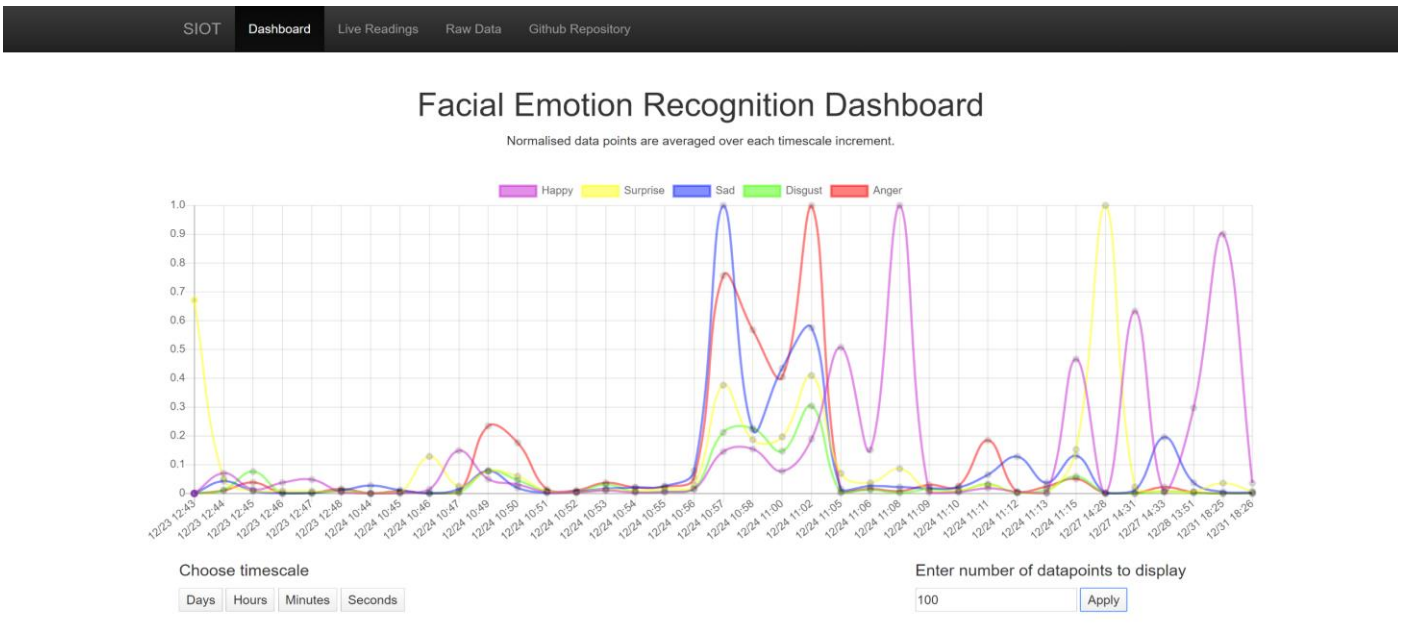

SIOT
Dashboard
Live Readings
Raw Data
Github Repository
Facial Emotion Recognition Dashboard
Normalised data points are averaged over each timescale increment.
Happy
Surprise
Sad
Disgust
Anger
Choose timescale
Days
Hours
Minutes
Seconds
Enter number of datapoints to display
100
Apply

Figure 15: Averaged Datapoints of the last 100 minutes

'Apply' button, which is necessary to trigger the 'graph2' function as this webpage does not update the graph automatically as the 'live readings' webpage does. [see HTML code] [see Javascript code]. Figure 15 shows the averaged datapoints of the readings from the last 100 minutes, and Figure 16 shows the averaged readings of the last 10 hours.

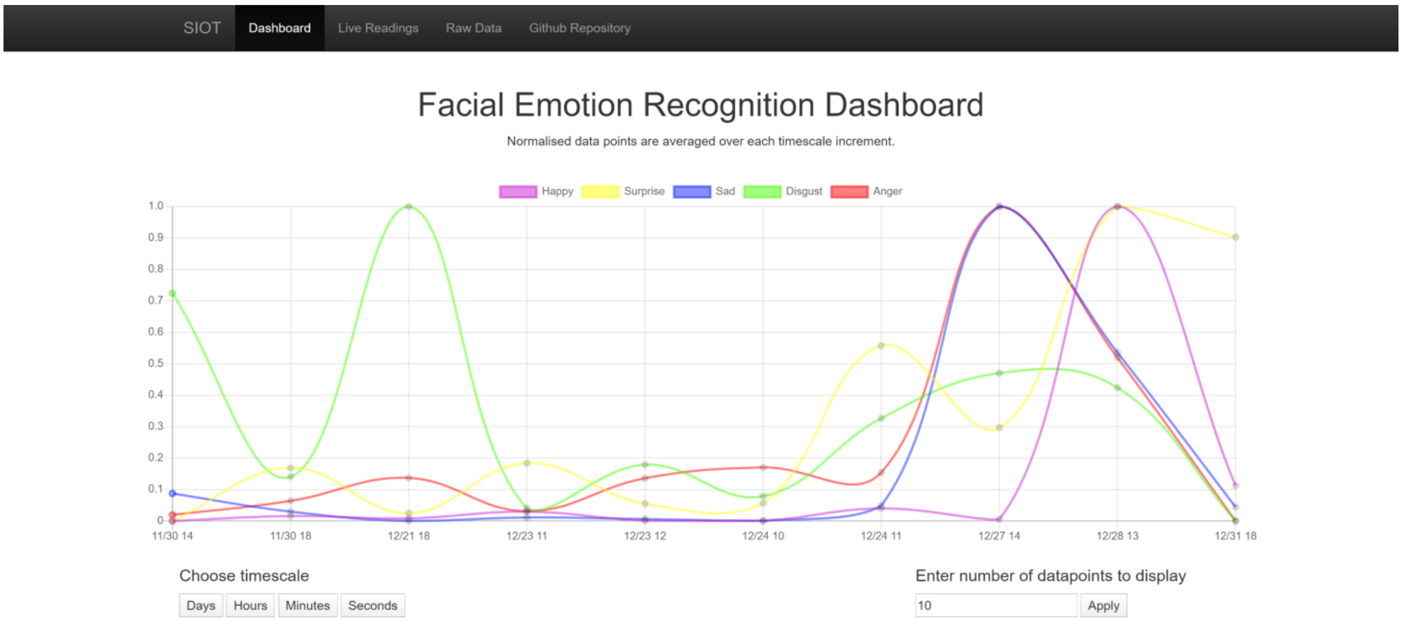


Figure 16: Averaged Datapoints of the last 10 hours

Data analytics, inferences and insights

ISS Tracker

The ISS is considered by some to be one of the most advanced science and engineering projects ever to have been made, and also represents a great political achievement through the cooperation of the space agencies of the United States, Russia, Europe, Japan, and Canada [4]. Whilst it is interesting to see the ISS's sinusoidal trajectory on a 2D map projection as well as it's altitude and speed, the data serves little practical purpose other than personal interest and satisfaction. The project does match up reasonably accurately with NASA's official ISS tracker [link], with a very slight latitudinal offset.

The email alert will hopefully prove genuinely useful for catching sightings of the ISS, however poor weather has prevented this from happening so far.

Smart Mirror

At the time of writing I have collected over a weeks worth of data and intend to continue data collection until the submission date. The neural network works reasonably well and is around 66% accurate; I am able to change the predicted emotion by pulling different faces reasonably reliably, although particular emotions such as 'sad' are difficult to trigger. The winners of the Kaggle competition involving this dataset achieved a similar accuracy, suggesting that this is close to the upper limit attainable with this particular dataset and current algorithms. A better network could possibly be obtained with higher resolution pictures; however, this would greatly increase the training time, which already took a staggering 3 days when run on a high-end GPU and would likely require renting a dedicated server. In addition to this, the dataset has been tagged manually by a human, which means the emotions classification may also be subjective.

Realistically it is difficult to draw useful insights into the emotion data, because unfortunately it does not work particularly well with subtle facial expressions and users are unlikely to have exaggerated facial expressions when standing in front of their mirror in the morning. In addition to this, emotion is a highly subjective and personal experience. However, during testing there were times when I noticed the smart mirror did accurately log genuine emotions! in particular anger during debugging and happiness, both when the project was going well and when testing with friends who were amused by the occasional inaccuracies. This is a promising sign and, in my opinion, proves that the concept does have potential.

Discussions on the important aspects of the project

ISS Tracker

The important aspects for me were the learning experiences about different topics, in particular CORS requests, becoming more familiar with Javascript especially JQuery and completing a project to do with the ISS which is a topic I find interesting. Whilst other websites do exist that show the same information, I was surprised that (whilst they are do include more features) the existing implementations I looked at [link] [link] were not of a high aesthetic quality and both (including NASA's own) displayed a google maps error at the time of writing. Therefore, making it publicly accessible was also important to me in case anyone else was interested or might find the data useful.

Smart Mirror

The important aspects for me were the learning experiences around Python convolutional neural networks, networking, Javascript and general data handling. I feel I have learnt a lot from this project and am in a good position to take this knowledge further.

It is worth touching on the privacy and data protection issues that arise with facial recognition and mood tracking. I have only tested the system on myself and consenting friends and have the camera installed in my room away from the general public. Were this to be a consumer product data privacy and cyber security would be of great concern due to the nature of the data and the non-encrypted data transfer exposure should the network be compromised.

Avenues for future work and potential impact

ISS Tracker

I would like to add a trajectory path to the web visualisation of the ISS, showing the predicted position so that users could see more than the instantaneous data.

I believe the email bot has significant scope for improvement. Firstly, I would like to run the system on a dedicated low spec cloud server, to prevent me from requiring a Raspberry Pi constantly plugged in. I would like to build a simple webapp whereby users submit their email and region of interest to a scalable MySQL database. The server would then send out emails whenever the ISS was above a user's specified location. I would require my own email server because Gmail would likely close the account suspecting It of spam.

Smart Mirror

An initial goal for this project was to be able to activate a 'training mode' through the web interface. This would allow the user to log their own emotion manually and then instead of the camera feed being fed into the neural network model, the picture would be appended to the training dataset, and the model could be retrained. This would allow the user to improve the neural network model to work better for them specifically, I believe this would make a big difference in terms of accuracy, however training the model cannot be done locally on the Raspberry Pi for obvious reasons, and this would therefore require the user to transfer the model over to the Pi to observe any improvements. Unfortunately, I have not yet had time to implement this change.

Constantly saving over a CSV file is not an ideal solution, I believe using a dedicated database language such as MySQL would be much better, however I have never done this before and did not want to be over ambitious at this stage, but this is certainly an avenue for potential improvement. I would also like to implement reading the database from the AWS S3 bucket, this would allow multiple smart mirrors to retrieve the latest data-set, append data to it and save it once again to the cloud. This would allow the system to be scalable.

Appendix

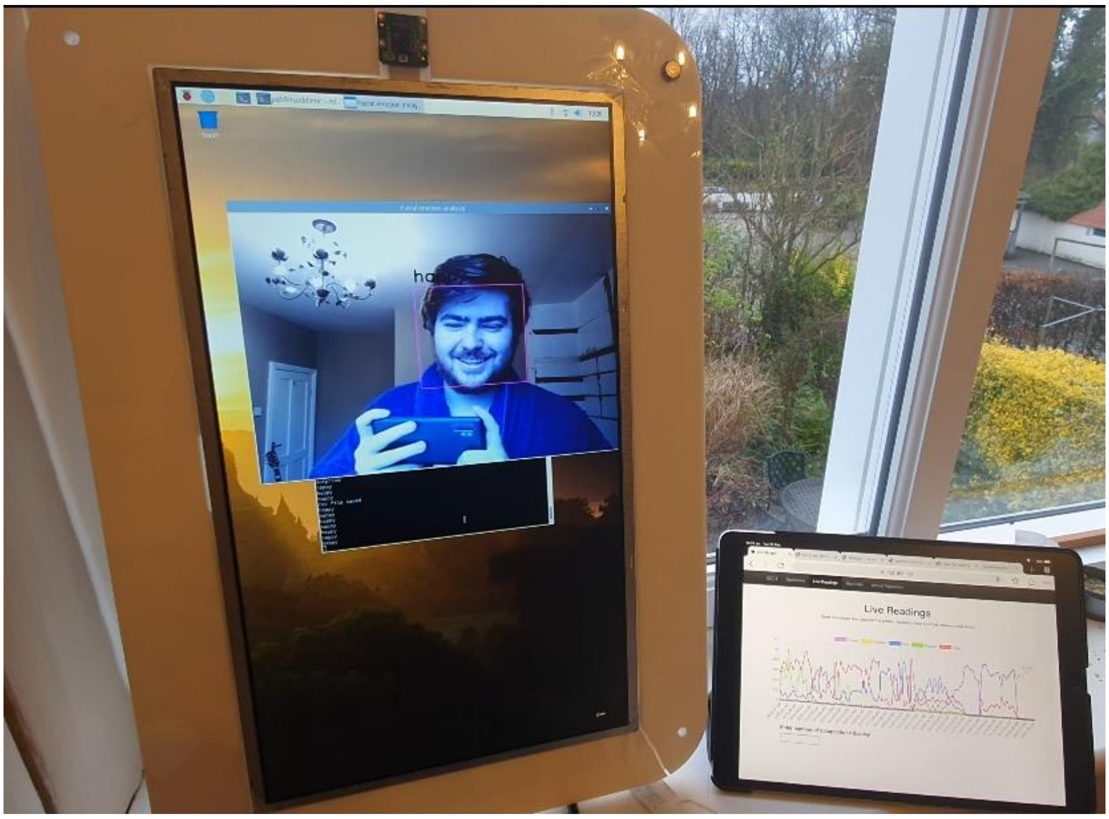
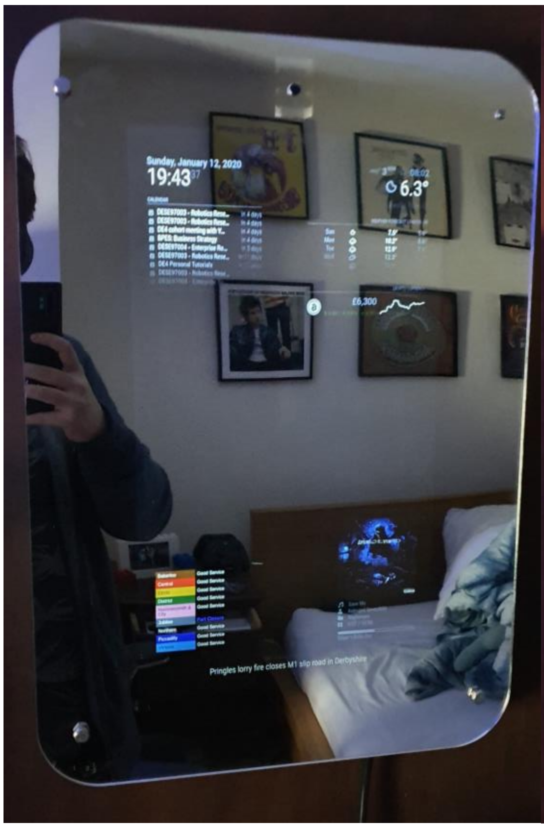


Figure 17: Emotion Recognition Running in 'debug' mode on the Smart Mirror (mirror panel removed), with the Live Readings webpage open on an iPad and Updating in Real Time.



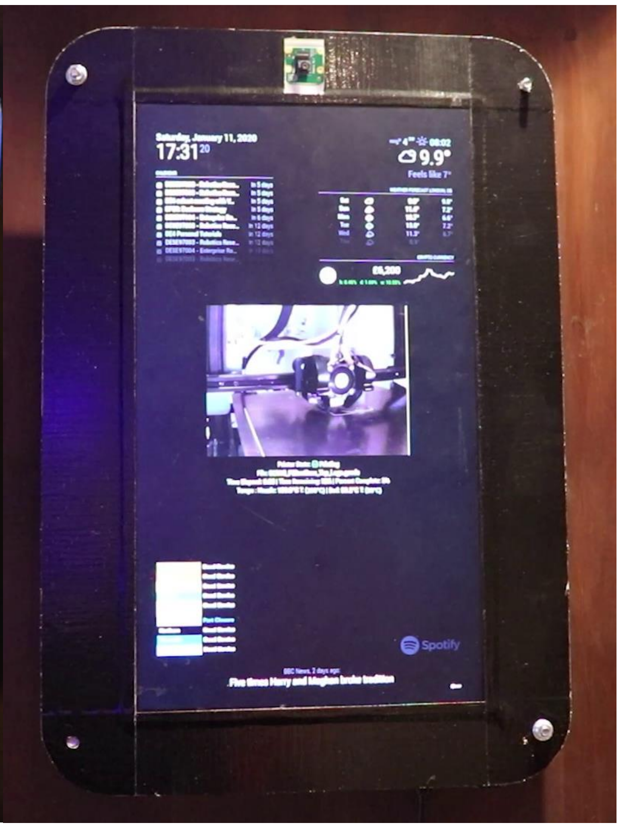


Figure 18: Emotion Recognition Running with the Smart Mirror Interface Turned On. The Spotify API is Showing Track Information in the Bottom Right

Figure 19: Mirror Panel removed, showing the Smart Mirror APIs and the Video Stream from my 3D printer

References

[1] "https://en.wikipedia.org/wiki/List_of_map_projections," [Online].

[2] "Geomatics Guidance Note 7, part 2 Coordinate Conversions & Transformations including Formulas," *international association of oil and gas producers,* August 2018.

[3] "https://latitudelongitude.org/," [Online].

[4] "https://www.nasa.gov/mission_pages/station/cooperation/index.html," [Online].