



Final Year Project

AI Companion

Padraig Ó Cosgora

BEng. (Hons.) in Software & Electronic Engineering

Galway-Mayo Institute of Technology

2019/2020

Project Poster



By Padraig Ó Cosgora

Project Description

The AI Companion Robot is a home-assistant robot. The AI Companion will intelligently operate around the house and can serve as your eyes and ears when you're away from home. Equipped with machine learning capabilities for financial news sentiment analysis.

Hardware

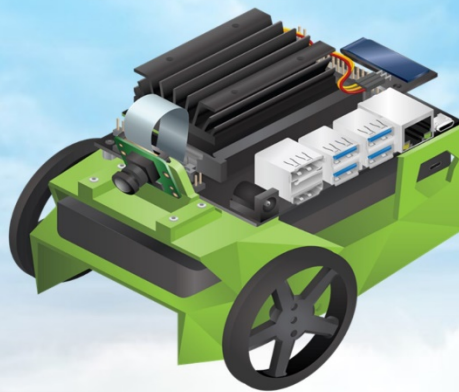
Development Board: Jetson Nano, ESP32

Sensors: CSI camera, OLED Display, IR Sensor, BMP280

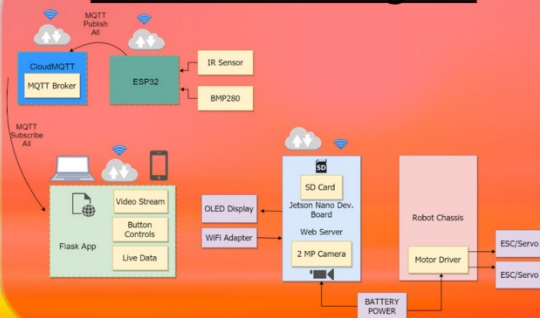
Robot chassis: motor driver (TB6612), 2x servo motor, 3D printed body

AI Companion

BEng in Software & Electronic Engineering



Architectural Diagram



Results

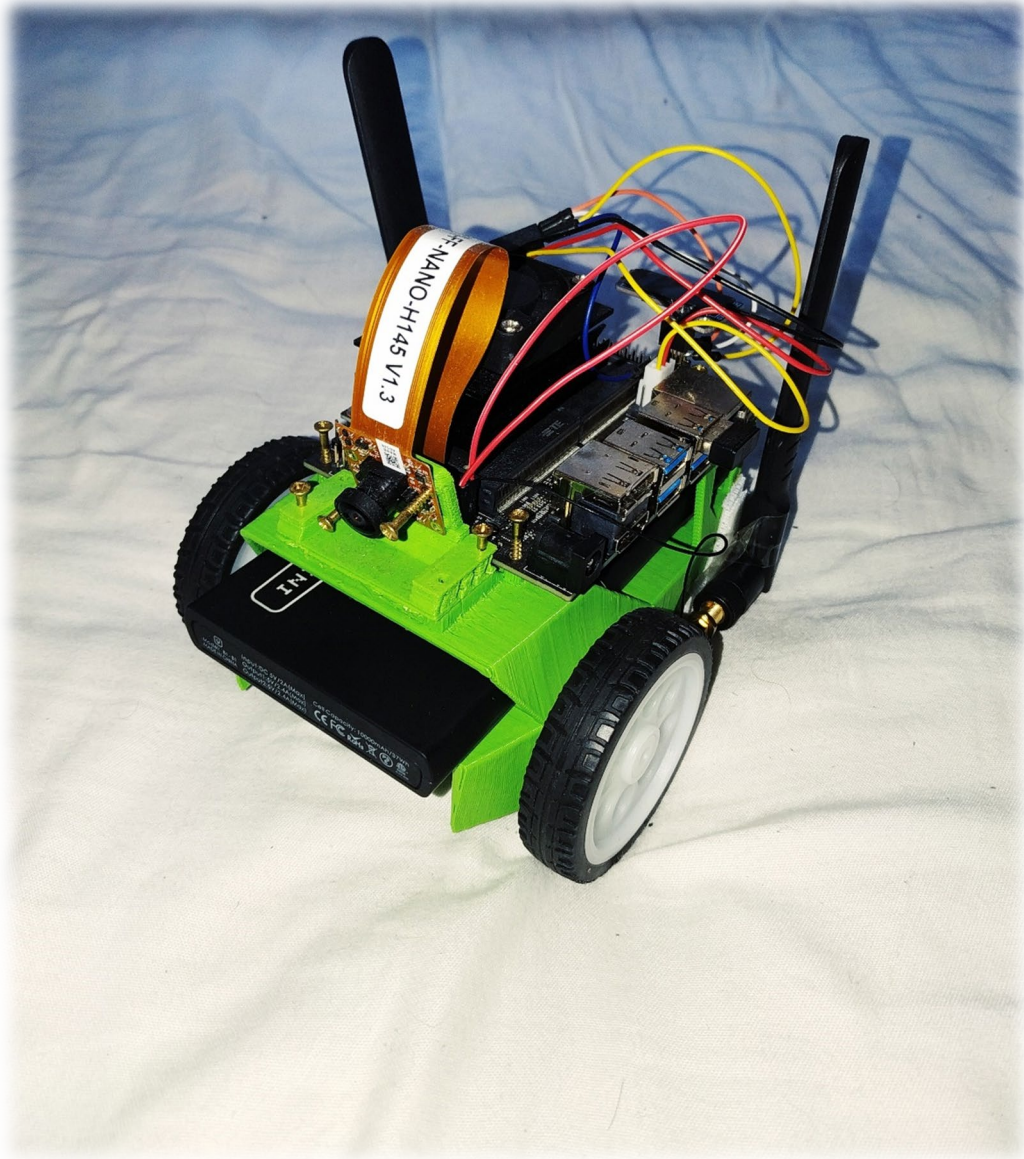
The AI Companion Robot can intelligently operate around the house by avoiding objects and recognising objects or can be controlled remotely for home security. Daily financial news sentiment analysis on any stock.

Skills & Technologies



- Python programming
- Machine Learning—Object Avoidance, Object Detection, Financial Sentiment Analysis
- Servo motor control
- Agile project development
- Flask web server programming
- Problem solving
- Cloud Computing

Project Picture



Abstract

My project aimed to build upon the successes of both the personal assistant robot market and the virtual assistant market by creating an artificially intelligent (AI) personal assistant robot that could act as a medium for virtual assistants, reducing the need for consumers to purchase virtual assistants all over the house.

The robot built was based on Nvidia's fantastic educational DIY AI robot titled 'Jetbot'. [1] Individual components were tested and verified to be operating correctly. Additionally, a web-interface to the robot was created to enable remote robot control and real-time data collection. Collision avoidance, object detection, facial recognition and the Financial News Sentiment Analyser are also implemented.

Declaration

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering in Computer & 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.

Padraig Ó Cosgora

Acknowledgements

I would firstly like to express my highest appreciation to all the mentors, supervisors and technicians involved in ensuring the module ran smoothly throughout the year, amid a global pandemic.

As the first fourth-year students to undertake a Final Year Project, the unwavering support received from all staff has been tremendous.

I'd especially like to thank Michelle Lynch and Brian O'Shea for their supervision and support throughout the year. Throughout the year in weekly lab sessions, I received valuable advice, knowledge and guidance on many areas of my project. I did my best to learn from them and incorporate as much as possible into my project.

I'd also like to thank my classmates for making the weekly labs a fun stress-free learning environment.

Lastly, I would not have been able to complete this project without the support and continued support from my girlfriend, family and friends.

Table of Contents

1	Summary	9
2	Introduction	10
3	The AI, Machine Learning and Deep Learning Overview	12
4	MQTT Overview	14
5	Project Architecture	15
5.1	ESP32	16
6	Development Platform and Tools.....	17
6.1	Arduino IDE.....	17
6.2	PyCharm IDE	18
6.3	Azure Notebooks	19
6.4	JupyterLab	20
7	Sensors.....	21
7.1	IR Sensor	21
7.2	Barometer	22
8	Jetson Nano.....	24
8.1	HTTP Web Server	25
8.1.1	Flask web framework	25
8.1.2	Configuration.....	26
8.2	MQTT Broker	27
8.2.1	CloudMQTT	27
9	Object Detection	29
10	Portfolio Sentiment Analysis	30
11	Collision Avoidance	31
12	Facial Recognition	32
13	Web-Interface	33
13.1	Homepage.....	34
13.2	MQTT Connection	34
13.3	Canvas Graphs API	37
13.4	HTML & CSS.....	38
13.5	GUI Page	39
13.6	Flask Video Streaming.....	41

13.7	Portfolio Page	42
14	Conclusion.....	45
15	References	46

1 Summary

My motivation for choosing this project stems back to when I was at Intel and took part in a Kaggle competition; Kaggle is a Machine Learning website with learning resources and competitions. I wanted to expand my knowledge base in this area by developing and deploying my own ML algorithms. The goal of the project was, in the end, to have a live web-interface, displaying a video-feed of the AI Companion alongside data from the sensors, running my ML algorithm and for it to be accessible by any smart device. I was able to do this by creating a flask web server interface for the Nvidia Jetbot open-source robot. The web server hosted by Portfolio Sentiment Analyser algorithm.

The external hardware consisted of an ESP32 microcontroller, one Jetson Nano and one each of the following sensors - 2MP camera, IR sensor and barometer. Additional features added on the web interface were graphs/charts of the incoming sensor data using Canvas JavaScript(JS) API. The leading technologies used include MQTT, Flask webserver, Web-sockets, JS and HTML/CSS, machine learning.

The Jetson Nano hosts the Flask HTTP server for the interface. The user (client) of the interface connects to via web-sockets to the MQTT broker (CloudMQTT) within the web interface. The micro-controller also connects to the MQTT broker, thus enabling live data transmission. This data is then showcased using the dynamically updating graphs. The web interface also has a robot live video stream and remote controls, as well as the portfolio analysis built-in. The AI Companion also has separate functionality of collision avoidance, object detection and facial recognition.

2 Introduction

Personal assistant robots can support you with everyday household activities, making life more comfortable. These robotic companions can serve a variety of purposes: from education to home automation, with the overall aim being less work for the robot's owner. [2] According to the latest robotics forecast by Tractica, worldwide consumer robots sales reached \$5.6 billion in 2018 and Tractica expects the market to continue to expand rapidly over the next few years, hitting a market value of \$19 billion by the end of 2025 [3]. However, these are not the same as virtual assistants.

Virtual assistants are usually cloud-based programs which require devices and applications connected to the internet to function. Siri on Apple devices, Cortana on Microsoft devices and Google Assistant on Android devices are three such apps [4]. As of 2019, there are an estimated 3.25 billion digital voice assistants around the world. Forecasts suggest that by 2023 the number will rise to around eight billion units – outnumbering the human population. They are making the adoption of digital assistants, one of the fastest-growing trends in consumer technology [5]. Added to that, the sales of these smart devices is set to increase by as much as 30% as a result of the coronavirus outbreak, according to the global technology market firm ABI Research [6].

The goal of this project was to build a home-assistant AI robot based upon an open-source robot design. The goal was to have the robot autonomously move around the home and be able to avoid obstacles, recognise faces and recognise objects by only using a single camera attached to the robot and machine learning algorithms running locally on the board.

As the project progressed, I also wanted the home-assistant robot to have the capability of being controlled remotely via web-interface by the owner, therefore acting as a home security robot by providing live data and a live video stream of the robots surroundings as well as control of the robot. As well as that, I wanted the home assistant robot to offer a portfolio analysis tool via a locally running machine learning (ML) algorithm analysing financial news sentiment in real-time – to showcase the potential of the personal assistant robot. The robot could then be paired with any virtual assistant and become a smart personal assistant robot, capable of operating around the house – and capable of remote web-interface control for added security.

3 The AI, Machine Learning and Deep Learning Overview

Artificial Intelligence (AI) has been around since the 1950's when the term was coined at the Dartmouth Conferences in 1956. Artificial Intelligence can be simply defined as human intelligence exhibited by machines. Subsets of AI include Machine Learning (ML) and Deep Learning.

ML can be defined as computers with the ability to learn without being explicitly programmed. Deep Learning can be defined as a network capable of adapting itself to new data [7].

In the decades since its creation, AI has been heralded as the future of our civilisations intelligence enhancements. Since 2012, AI has seen mass growth and has become a part of our everyday lives. A lot of that growth stems from the full availability of Graphical Processing Units (GPUs) that make parallel processing cheaper, faster and more powerful. Alongside the hardware enhancements, however, is the explosion in digital data – the Big Data movement. Digital data is growing at an exponential rate, the internet at its crux – images, text, transactions, mapping data, etc.

Technologies that can perform specific tasks as well as, or better than humans can – are defined as the concept; Narrow AI. Narrow AI will be utilised in this project for obstacle avoidance, object recognition, facial recognition and portfolio sentiment analysis. The "intelligence" of the above technologies will come from ML. ML, at its core, is the practice of using algorithms to parse data, learn from it, and then make a determination or prediction about something [8]. The ML algorithm Random Forrest is utilised for the portfolio sentiment analysis in this project.

Deep Learning is an advanced technique for implementing ML. Deep Learning is the term coined to the algorithmic approach of utilising artificial neural networks. Artificial neural

networks have discrete layers, connections, and directions of data propagation. Each neuron assigns a weighting to its input — how correct or incorrect it is relative to the task being performed. The final output is then determined by the total of those weightings. Within the scope of this project, object detection, and collision avoidance utilise machine learning models that are trained, artificial neural networks [9].

4 MQTT Overview

MQTT (*Message Queuing Telemetry Transport*) is a machine-to-machine connectivity protocol. It was designed as an extremely lightweight publish/subscribe messaging transport between client and broker. [10]

An **MQTT client** is defined as any device from a microcontroller up to a fully-fledged server, that has an MQTT library running and is connecting to an MQTT broker over any network.

The **broker** is primarily responsible for receiving all messages, filtering them, deciding who is interested in it and then sending the message to all subscribed clients. Another responsibility of the broker is the authentication and authorisation of clients.

The MQTT protocol is based on top of TCP/IP, and both client and broker need to have a TCP/IP stack.

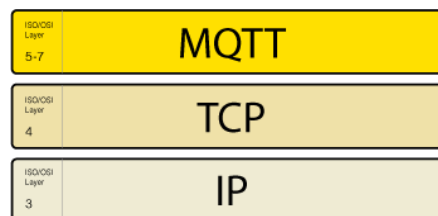


Figure 1 TCP/IP Stack

As MQTT is data-centric, it is more suitable when it is referred to as IoT development.

Based on a report by **Jan Bartnitsky** for flespi.com **MQTT is 20 times faster** and requires **50 times less traffic** on the task of posting consistent time-valuable data. [11]

5 Project Architecture

The project architecture consists of two major development boards. There's a single ESP32 operating as the MQTT Client, uploading all valuable sensor data. This MQTT Client connects to a cloud MQTT Broker, hosted by CloudMQTT. The CloudMQTT Broker is connected to by the web-interface as well as the ESP32 and ensures all data is sent and received by the publishers and subscribers of data. Added to that, there's the Jetson Nano, which is hosting a Flask HTTP server for the web-interface, as well as hosting the Jupyter Notebook and handling the Motor driver, which controls two servo motors. It also handles the 2 MP Camera.

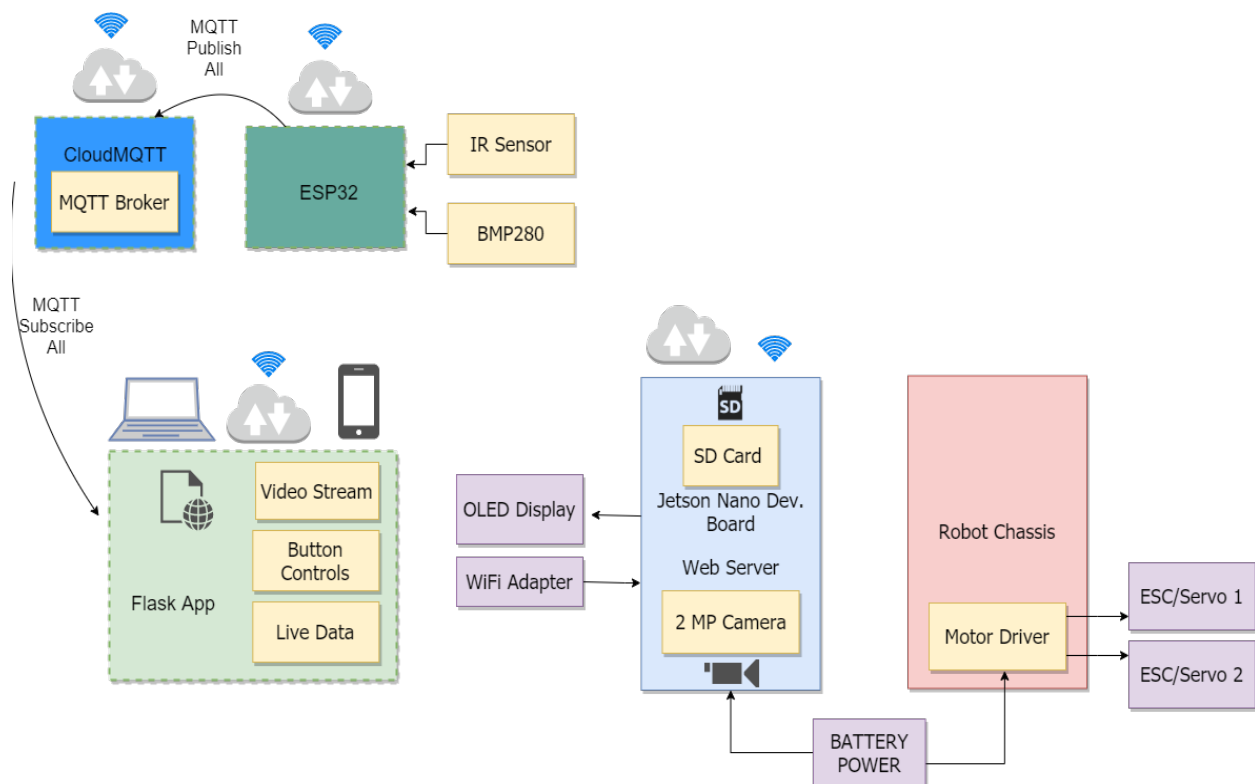


Figure 2 Architecture Diagram

5.1 ESP32

ESP32 is a powerful, generic Wi-Fi+BT+BLE microcontroller module that *"targets a wide variety of applications, ranging from low-power sensor networks to the most demanding tasks, such as voice encoding, music streaming and MP3 decoding."* It's an ideal solution for IoT applications [12].

The board itself features 10 GPIO ports, 2 analogue-to-digital converters, 3 UART interfaces, 3 SPI interfaces, 2 I2C interfaces, 2 I2S interfaces, micro USB-Serial interface, 16 PWM output channels and it's compatible with the Arduino Integrated Development Environment(IDE).

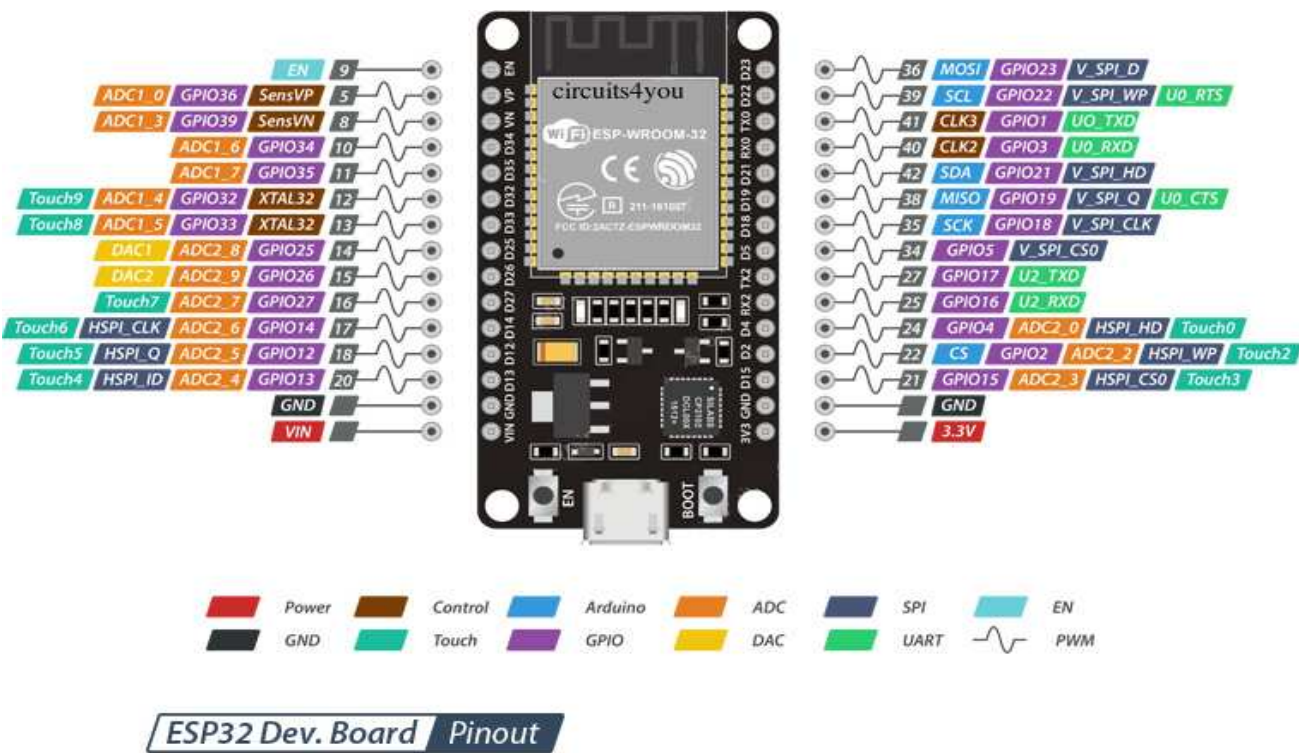


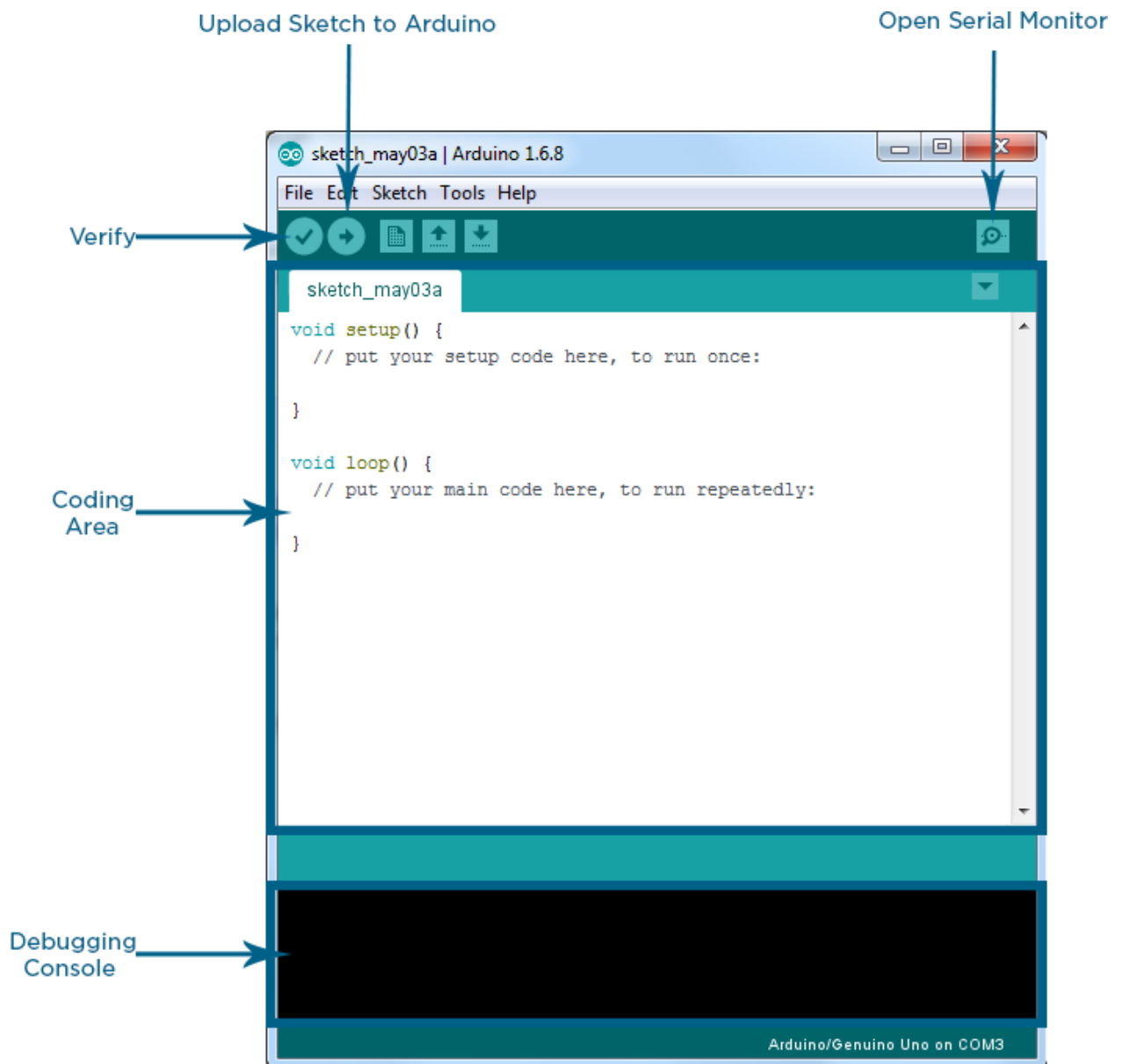
Figure 3 ESP32 Pinout

6 Development Platform and Tools

6.1 Arduino IDE

The Arduino Integrated Development Environment - or Arduino Software (IDE) - contains a text editor for writing code, a message area, a text console, a toolbar with buttons for standard functions and a series of menus. It connects to compatible hardware to upload programs and communicate with them. This IDE was used to program the ESP32. [13]

Figure 4 Arduino IDE



6.2 PyCharm IDE

PyCharm is a powerful IDE for modern Python development. PyCharm provides full support for JavaScript, TypeScript, HTML, CSS as well as for frameworks such as Flask, Django, and Jupyter Notebooks either right out of the box or with easily installed plugins. [14] In addition to supporting versions 2.x and 3.x of Python, PyCharm is also compatible with Windows, Linux, and macOS.

PyCharm was used mainly for the Flask web server, including the Python, JavaScript, HTML & CSS files hosted by the Jetson Nano.

It has 'deployment' software built within the IDE, enabling me to write code that automatically uploads to my (Flask) HTTP server(via FTP) location on the Jetson Nano, over Wi-Fi.

It also has an SSH Terminal built-in to the IDE, enabling me to execute commands such as launching the (Flask) HTTP server on the Jetson Nano.

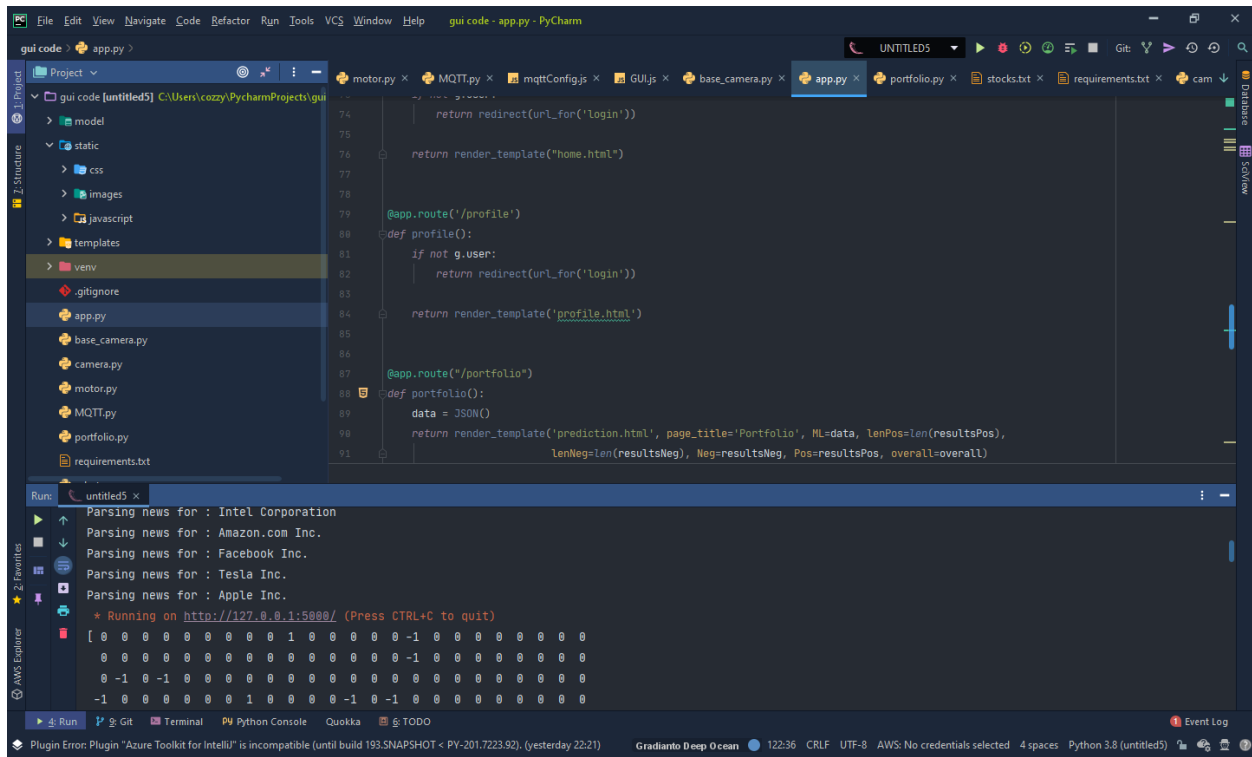


Figure 5 PyCharm IDE

6.3 Azure Notebooks

Azure Notebook is a free service for anyone to develop and run code in their browser using Jupyter. Jupyter is an open-source project that enables combining markdown prose, executable code, and graphics onto a single canvas: A Jupyter notebook shows Python code, markdown and interactive graphics [15].

Azure Notebooks was used in the development of the Portfolio Sentiment Analyser. It was used to import the dataset, import all required python packages, train/test a Random Forrest model, and to save the model.

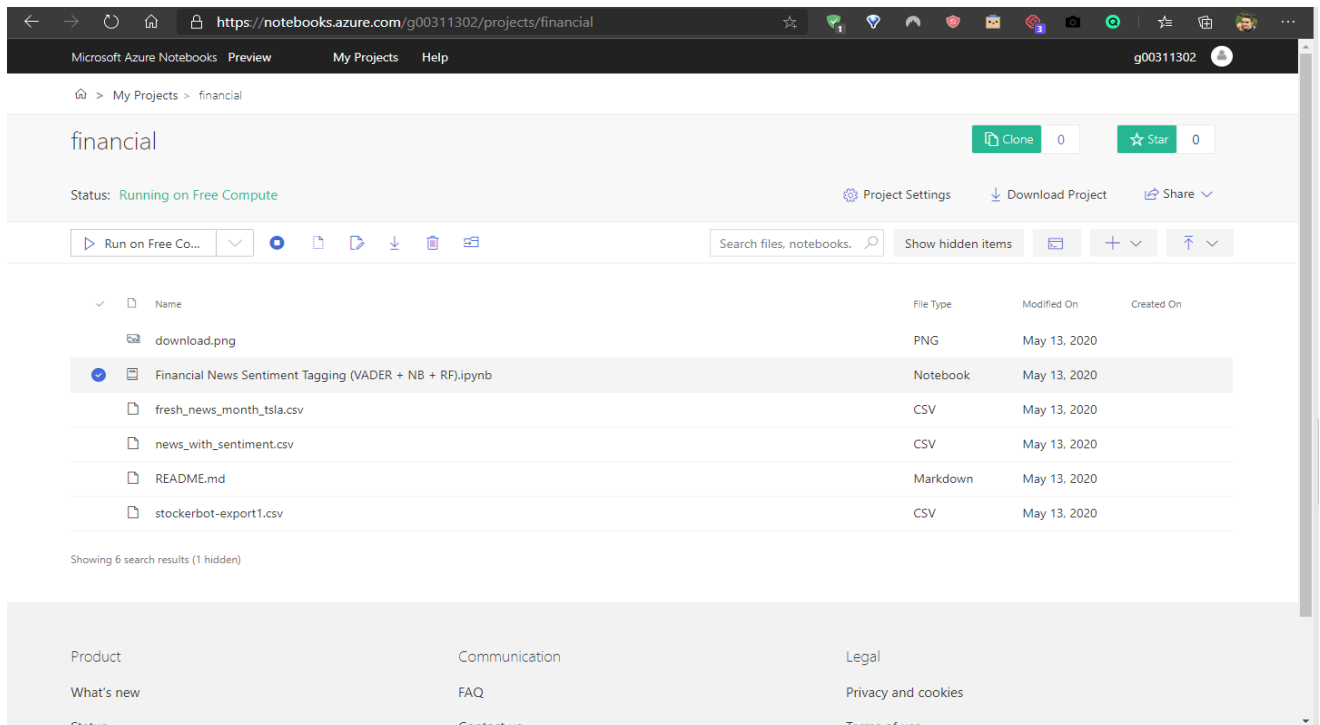


Figure 6 Azure Notebooks Platform

6.4 JupyterLab

JupyterLab is a web-based interactive development environment for Jupyter notebooks, code, and data. JupyterLab is flexible: configure and arrange the user interface to support a wide range of workflows in data science, scientific computing, and machine learning [16].

JupyterLab was utilised as a server on the Jetson Nano, to interact with the Nano when the Nano was configured in "headless" mode. i.e. with no monitor display. The Jupyter Notebook enabled the running of interactive python notebooks (.ipynb file extensions) which were used in the Jetbot learning tutorials, specifically the collision avoidance tutorial.

7 Sensors

The ESP32 was used for gathering data from sensors and publishing that data to the MQTT Broker.

7.1 IR Sensor

The Flying Fish IR Sensor recognises objects within an angle of 35° and a distance between 2 and 30 cm. It works via an infrared LED lighting up and the photodiode next to it, measuring the reflection, then the module can calculate the distance [17].

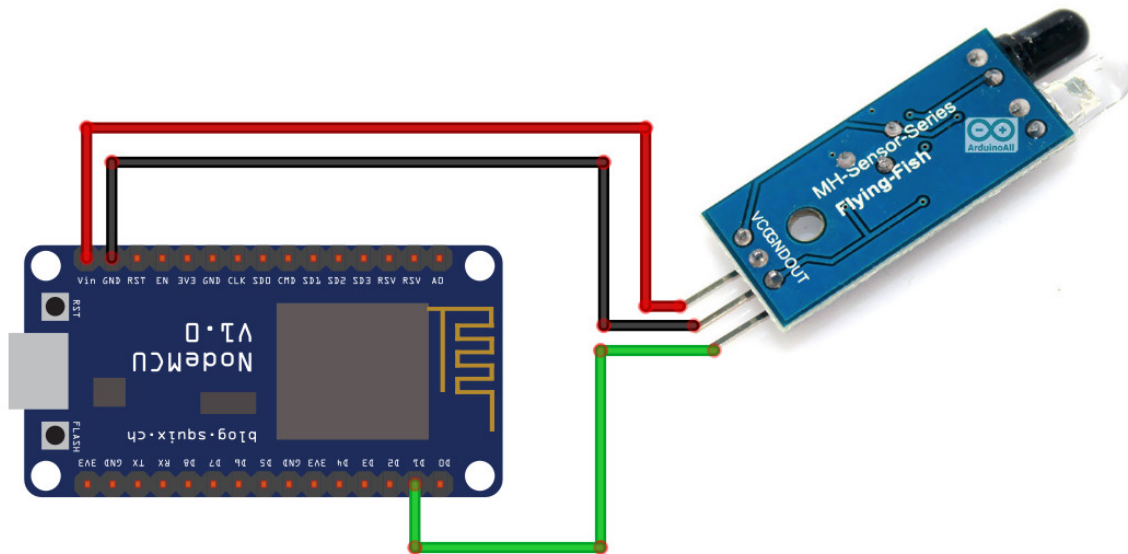


Figure 7 IR Sensor

7.2 Barometer

The BMP280 is a precision sensor used for measuring barometric pressure with ± 1 hPa absolute accuracy, temperature with ± 1 degree Celsius accuracy and altimeter with ± 1 m accuracy [18].

The sensor is communicating with the micro-controller via I²C and publishing the data to the CloudMQTT broker every second.

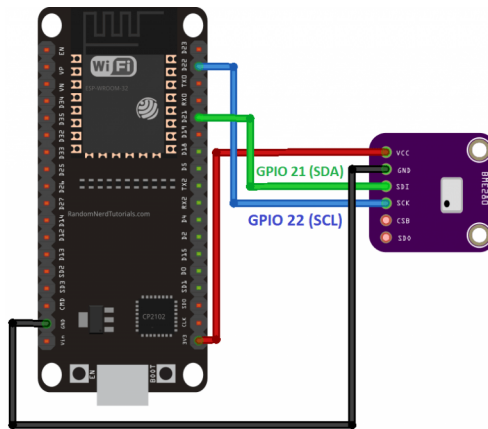
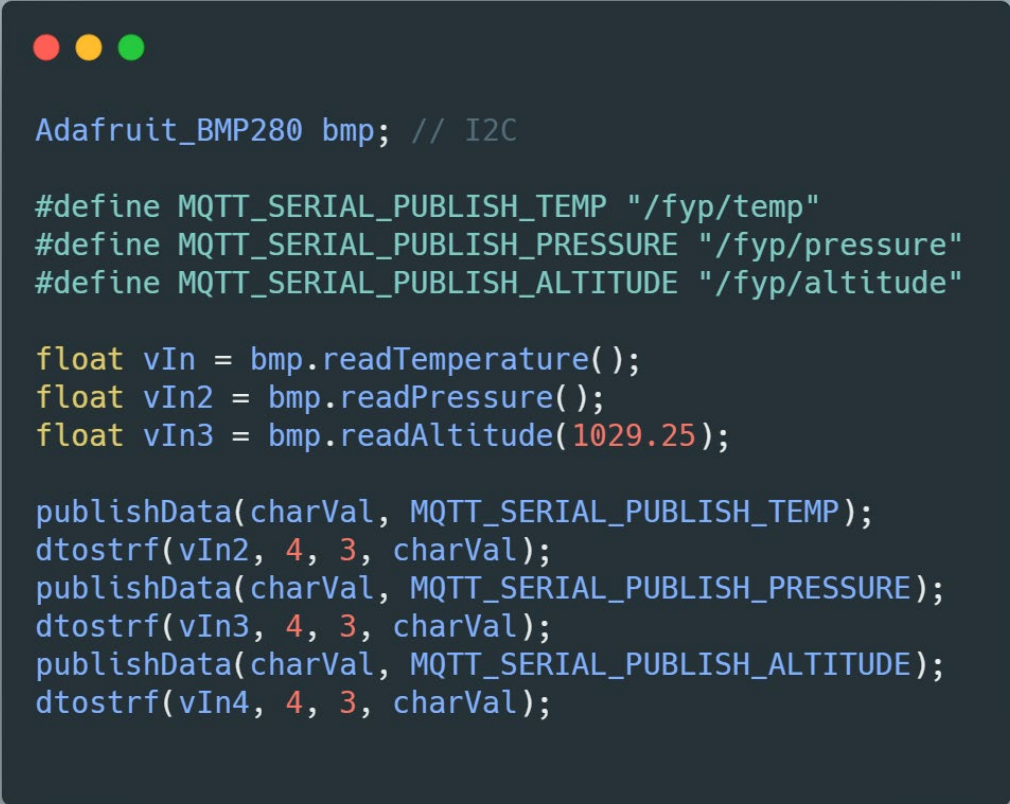


Figure 8 BMP280 circuit

The below code used the Adafruit_BMP280 library. The data is received using the `bmp.readTemperature()` function, stored as float variable, then it gets converted to a character array and then published to the broker under the correct topic name.



```
Adafruit_BMP280 bmp; // I2C

#define MQTT_SERIAL_PUBLISH_TEMP "/fyp/temp"
#define MQTT_SERIAL_PUBLISH_PRESSURE "/fyp/pressure"
#define MQTT_SERIAL_PUBLISH_ALTITUDE "/fyp/altitude"

float vIn = bmp.readTemperature();
float vIn2 = bmp.readPressure();
float vIn3 = bmp.readAltitude(1029.25);

publishData(charVal, MQTT_SERIAL_PUBLISH_TEMP);
dtostrf(vIn2, 4, 3, charVal);
publishData(charVal, MQTT_SERIAL_PUBLISH_PRESSURE);
dtostrf(vIn3, 4, 3, charVal);
publishData(charVal, MQTT_SERIAL_PUBLISH_ALTITUDE);
dtostrf(vIn4, 4, 3, charVal);
```

Figure 9 Barometer code

8 Jetson Nano

The Nvidia Jetson Nano is a developer kit, which consists of an SoM (System on Module) and a reference carrier board. It is targeted for creating embedded systems that need high processing power for machine learning applications.

JETSON NANO SPECIFICATIONS	
GPU	128 Core Maxwell 472 GFLOPs (FP16)
CPU	4 core ARM A57 @ 1.43 GHz
Memory	4 GB 64 bit LPDDR4 25.6 GB/s
Storage	16 GB eMMC
Video Encode	4K @ 30 4x 1080p @ 30 8x 720p @ 30 (H.264/H.265)
Video Decode	4K @ 60 2x 4K @ 30 8x 1080p @ 30 16x 720p @ 30 (H.264/H.265)
Camera	12 (3x4 or 4x2) MIPI CSI-2 DPHY 1.1 lanes (1.5 Gbps)
Display	HDMI 2.0 or DP1.2 eDP 1.4 DSI (1 x2) 2 simultaneous
UPHY	1 x1/2/4 PCIE 1 USB 3.0
SDIO/SPI/SysIOs/GPI Os/I2C	1x SDIO / 2x SPI / 5x SysIO / 13x GPIOs / 6x I2C

Figure 10 Jetson Nano diagram

The system is built around a 128 CUDA core Maxwell GPU (the same as the Nintendo Switch) with a quad-core ARM A57 CPU running at 1.43 GHz and coupled with 4GB of LPDDR4 memory. Jetson Nano is NVIDIA's take on "AI embedded platform for the masses" [19].

The Jetson Nano is utilised to host the Flask HTTP server, as well as to host the Jupyter Notebook server, and running inference on all the machine learning algorithms involved in collisions avoidance and object detection.

8.1 HTTP Web Server

A server that responds to HTTP requests to deliver content and services. The HTTP Web Server in my project was implemented using the Flask web framework.

8.1.1 Flask web framework

Flask is a web application framework written in Python. It is developed by Armin Ronacher, who leads an international group of Python enthusiasts named Pocco. Flask is based on the Werkzeug WSGI toolkit and Jinja2 template engine. The advantage of using Python for this project is that all the Jetbot robot modules and the Portfolio Sentiment Analyser were written in Python [20].

8.1.2 Configuration

```

from flask import (
    Flask,
    g,
    redirect,
    render_template,
    request,
    session,
    Response,
    url_for
)
from portfolio import JSON, resultsPos, resultsNeg, overall
from robot import Robot
from camera import Camera

class User:
    def __init__(self, id, username, password):
        self.id = id
        self.username = username
        self.password = password

    def __repr__(self):
        return f'<User: {self.username}>'

users = [User(id=1, username='Padraig', password='password'), User(id=2, username='Jane',
    password='password'), User(id=3, username='Joe', password='password')]

app = Flask(__name__)
app.secret_key = 'thisisasecret'
robot = Robot()

@app.before_request
def before_request():
    g.user = None

    if 'user_id' in session:
        user = [x for x in users if x.id == session['user_id']][0]
        g.user = user

@app.route('/GUI')
def GUI():
    return render_template('GUI.html')

@app.route('/')
def homepage():
    print("Re-directed user to login page")
    return redirect(url_for('login'))

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        session.pop('user_id', None)
        username = request.form['username']
        password = request.form['password']
        user = [x for x in users if x.username == username][0]
        if user and user.password == password:
            session['user_id'] = user.id
            return redirect(url_for('home'))
        return redirect(url_for('login'))
    return render_template('login.html')

@app.route('/home')
def home():
    if not g.user:
        return redirect(url_for('login'))
    return render_template("home.html")

@app.route('/profile')
def profile():
    if not g.user:
        return redirect(url_for('login'))
    return render_template('profile.html')

@app.route("/portfolio")
def portfolio():
    data = JSON()
    return render_template('prediction.html', page_title='Portfolio', ML=data, lenPos=len(resultsPos),
        lenNeg=len(resultsNeg), Neg=resultsNeg, Pos=resultsPos, overall=overall)

@app.route('/companion/<move>')
def companion(move):
    if move == 'forward':
        robot.forward()
    elif move == 'backward':
        robot.backward()
    elif move == 'left':
        robot.left()
    elif move == 'right':
        robot.right()
    elif move == 'stop':
        robot.stop()
    return '', 204

def gen(camera):
    """Video streaming generator function."""
    while True:
        frame = camera.get_frame()
        yield (b'--frame\r\n'
            b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n')

@app.route('/video_feed')
def video_feed():
    """Video streaming route. Put this in the src attribute of an img tag."""
    return Response(gen(Camera()),
        mimetype='multipart/x-mixed-replace; boundary=frame')

if __name__ == '__main__':
    app.run(debug=True, threaded=True)

```

Figure 11 Server configuration

This code defines all the routes and modules involved in the Flask web server and hosts the server on the default host 127.0.0.1 with default port 5000.

8.2 MQTT Broker

The MQTT protocol is based on the principle of publishing messages and subscribing to topics, or "pub/sub". Multiple clients connect to a broker and subscribe to topics that they are interested in. Clients also connect to the broker and publish messages to topics. Many clients may subscribe to the same topics and do with the information as they please. The broker and MQTT act as a simple, standard interface for everything to connect to.

8.2.1 CloudMQTT

CloudMQTT has managed Mosquitto servers in the cloud.

Mosquitto is an open-source (EPL/EDL licensed) message *broker* that implements the *MQTT* protocol versions 3.1 and 3.1.1. *Mosquitto* is lightweight and is suitable for use on all devices from low power single board computers to full servers. [20]

8.2.2 Configuration

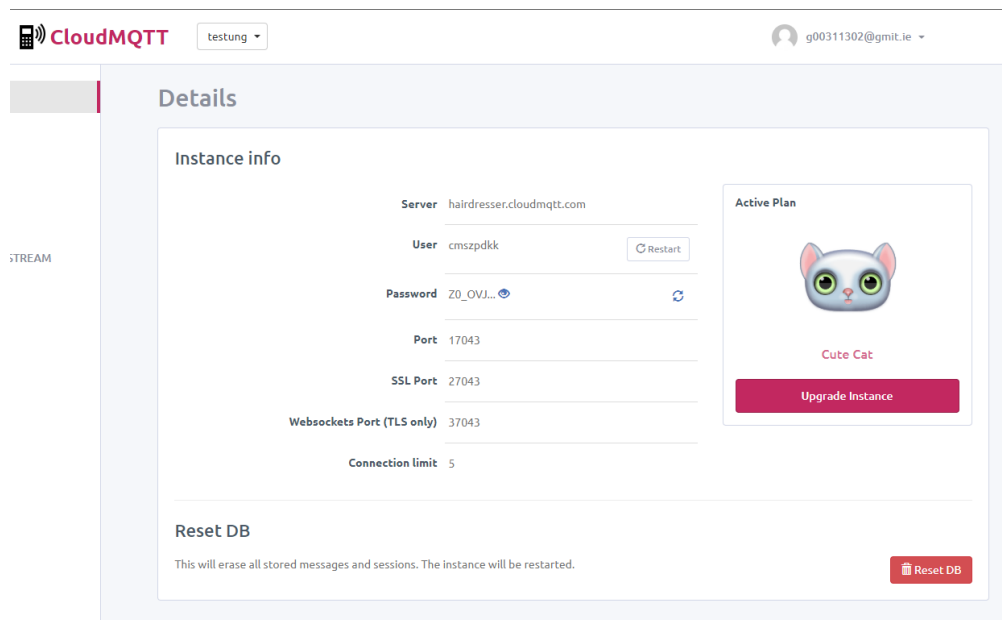


Figure 12 Broker Configuration

This configuration sets up a broker at the above server address on the default MQTT port 17043 (for communication with the ESP32). It also listens for WebSocket connections on port 37043 to enable the web-interface (web client) to display live incoming data by maintaining a constant connection with the page.

9 Object Detection

The SDD-MobileNet-V2 model pretrained on the 90-class MS-COCO dataset deep learning neural network is utilised for real-time object detection.



```
import jetson.inference
import jetson.utils

net = jetson.inference.detectNet("ssd-mobilenet-v2", threshold=0.5)
camera = jetson.utils.gstCamera(1280, 720, "0")
display = jetson.utils.glDisplay()

while display.IsOpen():
    img, width, height = camera.CaptureRGBA()
    detections = net.Detect(img, width, height)
    display.RenderOnce(img, width, height)
    display.SetTitle("Object Detection | Network {:.0f} FPS".format(net.GetNetworkFPS()))
```

Figure 13 Object Detection Code

10 Portfolio Sentiment Analysis

Random Forest classifier trained on data containing headlines and sentiment, returned an accuracy of ~93% on test data. This model is then used on the latest news to provide an overall summary of the sentiment of an entire stock portfolio. This is done by saving the model and storing it on the Flask web server.

```
In [15]: 1 # Read in 20,000 headlines
2 news_pd = pd.read_csv("../news_with_sentiment.csv")
3 news_pd = news_pd[:20000] # 20,000 rows will use more RAM than is available. Truncation required.
4 y = news_pd['sentiment']

In [16]: 1 from nltk.corpus import stopwords
2 from sklearn.feature_extraction.text import TfidfVectorizer
3
4 # Remove stopwords and vectorize the dataset
5 #TfidfVectorizer converts a collection of raw documents to a matrix of TF-IDF features.
6 vectorizer = TfidfVectorizer(max_features=2500, min_df=7, max_df=0.8, stop_words=stopwords.words('english'))
7 processed_features = vectorizer.fit_transform(news_pd['text']).toarray()

In [ ]: 1 # 80/20 data split
2 from sklearn.model_selection import train_test_split
3 X_train, X_test, y_train, y_test = train_test_split(processed_features, y, test_size=0.2, random_state=0)
4
5 # Fit our model with split data, starting with 450 estimators (450 decision trees)
6 from sklearn.ensemble import RandomForestClassifier
7
8 text_classifier = RandomForestClassifier(n_estimators=450, random_state=0)
9 text_classifier.fit(X_train, y_train)

/home/nbuser/anaconda3_420/lib/python3.5/site-packages/sklearn/ensemble/weight_boosting.py:29: DeprecationWarning: numpy.core.umath_tests is an internal NumPy module and should not be imported. It will be removed in a future NumPy release.
  from numpy.core.umath_tests import inner1d

In [ ]: 1 # Predicting the sentiment of our test data
2 predictions = text_classifier.predict(X_test)
3
4
5 # Checking our accuracy
6 from sklearn.metrics import accuracy_score
7 print(accuracy_score(y_test, predictions))

93.57% accuracy
```

Figure 14 Training Random Forrest Classifier

11 Collision Avoidance

The robot can be configured to learn it's new surroundings and any obstacles that may lie in its path – this can be done by taking a series of images of the obstacle's and classifying those images as blockages. Images will also be taken on the AI Companion when it's free to roam, and these two datasets are used as classifiers to apply transfer learning to a neural network that will train a machine learning model to recognise those obstacles in real-time going forward [21].

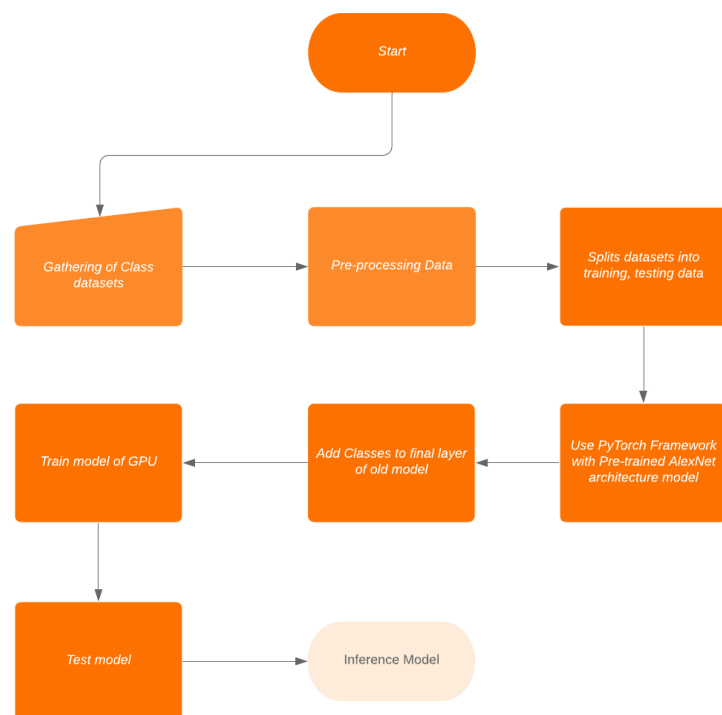


Figure 15 Transfer Learning: Collision Avoidance

12 Facial Recognition

Facial recognition was implemented, utilising the `face_recognition` module alongside OpenCV. However, while running this script, the robot camera appears to be too close to the ground to pick up faces. The code works by loading the facial encodings of 'known' people and comparing them to images of people to decipher if they're known or not. The implementation of this code would be to recognise intruders [22].

```
import face_recognition
import cv2
import os
print(cv2.__version__)

image_dir = './faces/known/padraig.jpg'

Encodings = []
Names = []

for root, dirs, files in os.walk(image_dir):
    print(files)
    for file in files:
        path=os.path.join(root, file)
        print(path)
        name=os.path.splitext(file)[0]
        print(name)
        person=face_recognition.load_image_file(path)
        encoding=face_recognition.face_encodings(person)[0]
        Encodings.append(encoding)
        Names.append(name)
    print(Names)

font=cv2.FONT_HERSHEY_SIMPLEX

testImage=face_recognition.load_image_file('./faces/unknown/u12.jpg')
facePositions=face_recognition.face_locations(testImage)
allEncodings=face_recognition.face_encodings(testImage, facePositions)
testImage=cv2.cvtColor(testImage, cv2.COLOR_RGB2BGR)

for(top, right, bottom, left), face_encoding in zip(facePositions, allEncodings):
    name='Unknown Person'
    matches=face_recognition.compare_faces(Encodings, face_encoding)
    if True in matches:
        first_match_index=matches.index(True)
        name=Names[first_match_index]
        cv2.rectangle(testImage,(left,top),(right,bottom),(0,0,255),2)
        cv2.putText(testImage, name, (left,top-6),font,.75,(0,255,255),2)
cv2.imshow('Picture', testImage)
cv2.moveWindow('Picture', 0, 0)
if cv2.waitKey(0)==ord('q'):
    cv2.destroyAllWindows()
```

Figure 16 Facial recognition

13 Web-Interface

When the server is loaded up, the user must have access credentials to access the robot's web-interface.

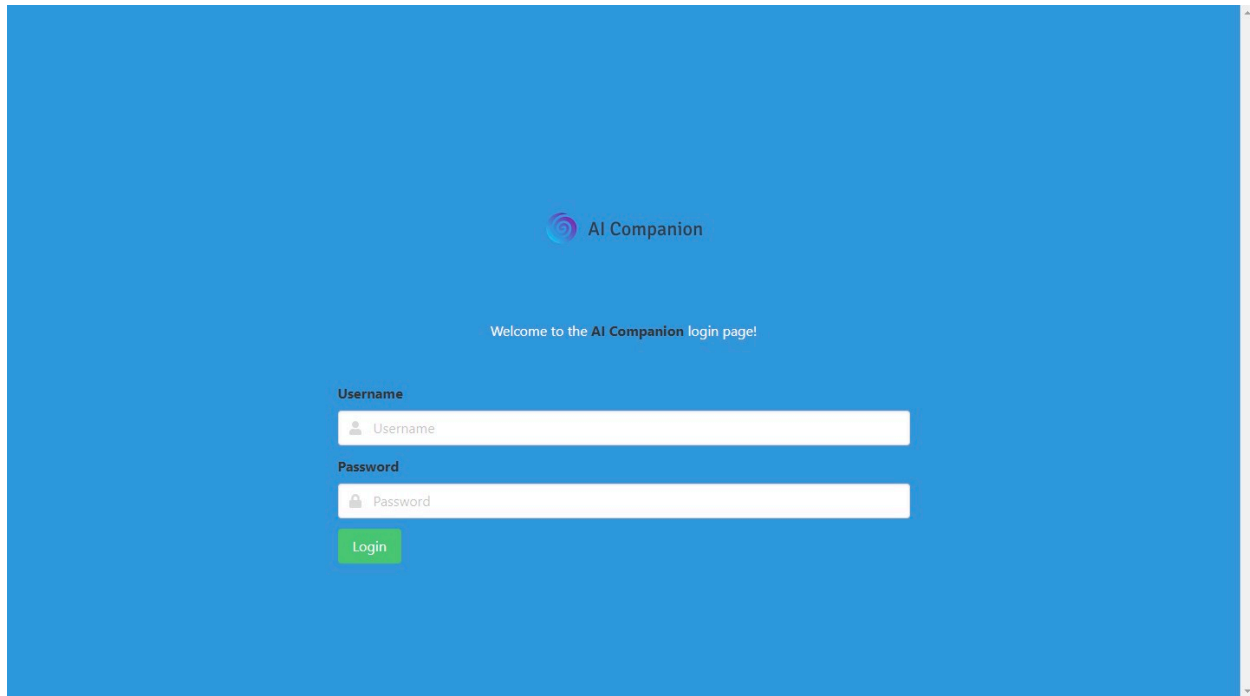


Figure 17 Login to interface dashboard

13.1 Homepage

After a successful login, the web-interface home page brings together elements of HTML, CSS & JavaScript. The elements are all described below.

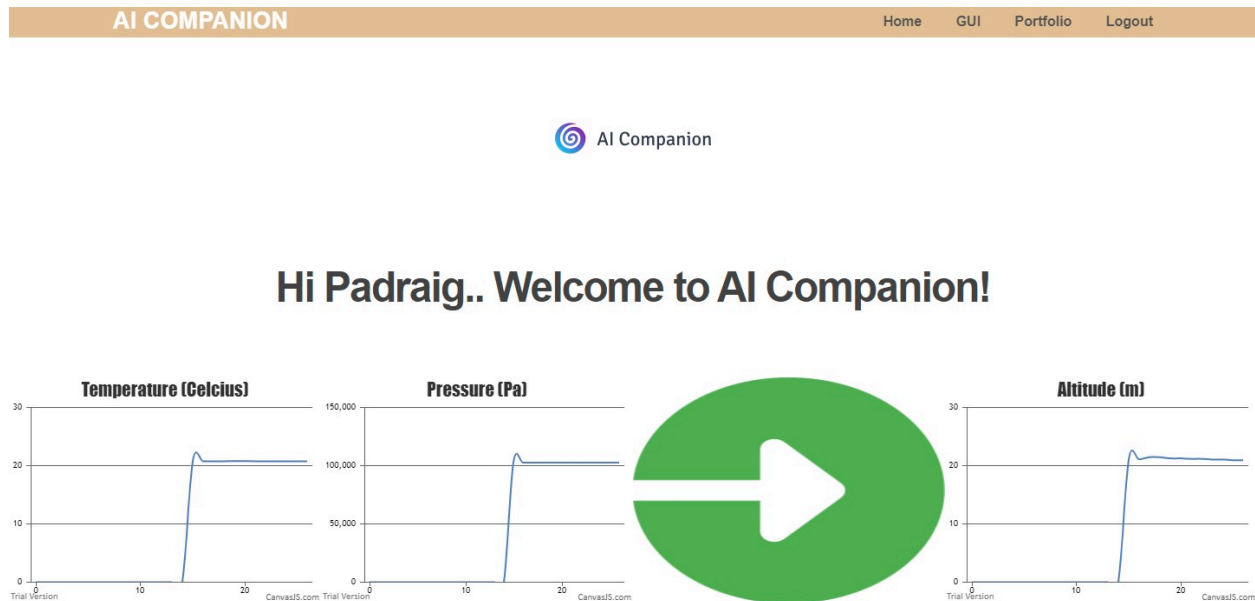


Figure 18 User interface dashboard

13.2 MQTT Connection

The client connects to the broker via WebSocket's. It does this to have live updating data on the webpage without the need to refresh the dashboard. This is done by connecting the client-side JavaScript on the homepage to the MQTT Broker via WebSocket's.

```

// Create a client instance
client = new Paho.MQTT.Client("hairstresser.cloudmqtt.com", 37043, "web_" + parseInt(Math.random() *
100, 10));

var device = {
  temp: 0,
  pressure: 0,
  altitude: 0,
  distance: 0
}

// set callback handlers
client.onConnectionLost = onConnectionLost;
client.onMessageArrived = onMessageArrived;
var options = {
  useSSL: true,
  userName: "cmszpdkk",
  password: "Z0_OVJ09x11Z",
  onSuccess: onConnect,
  onFailure: doFail
}

// connect the client
client.connect(options);

// called when the client connects
function onConnect() {
  // Once a connection has been made, make a subscription and send a message.
  console.log("onConnect");
  client.subscribe("/fyp/temp");
  client.subscribe("/fyp/altitude");
  client.subscribe("/fyp/pressure");
  client.subscribe("/fyp/obstacle");
  message = new Paho.MQTT.Message("Hello CloudMQTT");
  message.destinationName = "/cloudmqtt";
  loadTemp();
  loadAlt();
  loadPressure();
  client.send(message);
}

function doFail(e){
  console.log(e);
}

// called when the client loses its connection
function onConnectionLost(responseObject) {
  if (responseObject.errorCode !== 0) {
    console.log("onConnectionLost:"+responseObject.errorMessage);
  }
}

// called when a message arrives
function onMessageArrived(message) {
  console.log("onMessageArrived:"+message.payloadString);
  console.log("Topic: " + message.destinationName);
  if (message.destinationName === "/fyp/temp") {
    device.temp=parseFloat(message.payloadString);
  } else if (message.destinationName === "/fyp/altitude") {
    device.altitude=parseFloat(message.payloadString);
  } else if (message.destinationName === "/fyp/pressure") {
    device.pressure=parseFloat(message.payloadString);
  } else if (message.destinationName === "/fyp/obstacle") {
    if (parseFloat(message.payloadString) === 1.0) {
      device.distance=parseFloat(message.payloadString);
      document.getElementById('myImage').src='static\\images\\not-clear.png';
    } else {
      device.distance=parseFloat(message.payloadString);
      document.getElementById('myImage').src='static\\images\\clear.png';
    }
  } else {
    console.log("other topic")
  }
}

```

Figure 19 WebSocket connection

WebSocket is a network protocol that provides bi-directional communication between a browser and a web server. It was standardised in 2011, and all modern browsers have support for WebSocket's built-in. Like MQTT, WebSocket's are based on TCP [23].

WebSocket's have very little overhead in terms of bandwidth and latency compared to classic HTTP requests which are needed when using (long) polling. This philosophy of having as little overhead as possible fits very well to MQTT. The CloudMQTT broker has MQTT support [24].

13.3 Canvas Graphs API

HTML5 JavaScript Charts that are built on top of HTML5 *Canvas* Element. Renders across devices & is 10x faster than SVG based Charting Libraries. [25]

Dynamic or Live charts were used for displaying the temperature, altitude and atmospheric pressure data that varies with time. These Charts are interactive, responsive, support animation and live updates. Examples available online were altered to display the correct data incoming from the MQTT Broker. [26]

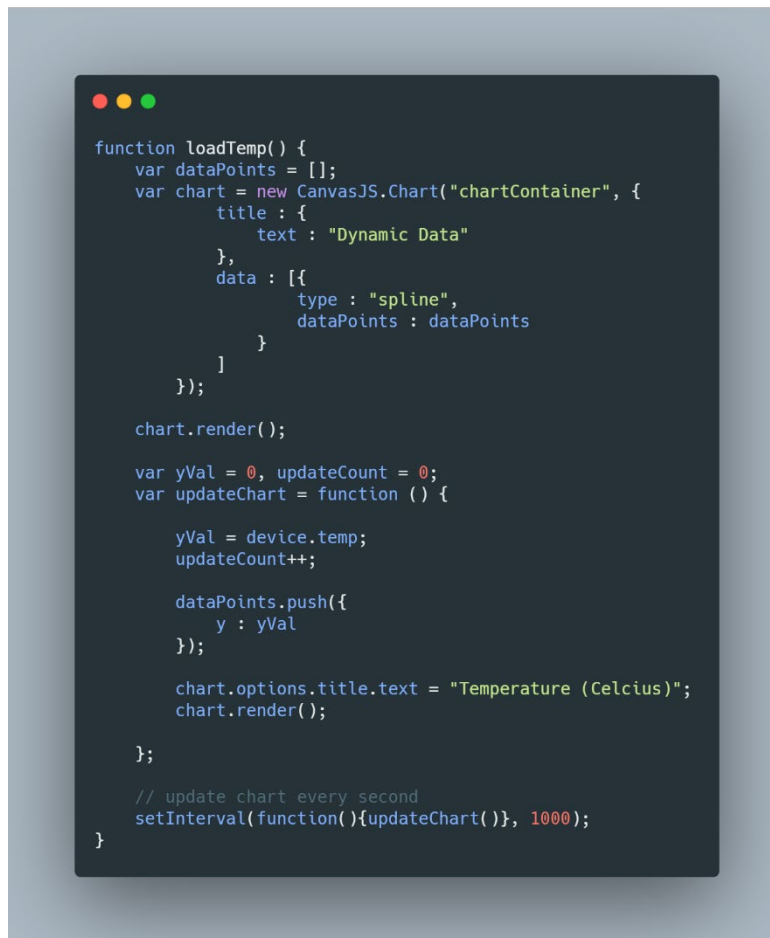


Figure 20 Canvas Graph API

13.4 HTML & CSS

HTML (the Hypertext Mark-up Language) and CSS (Cascading Style Sheets) are two of the core technologies for building Web pages. HTML provides the *structure* of the page, CSS the (visual and aural) *layout*, for a variety of devices. Along with graphics and scripting, HTML and CSS are the basis of building Web pages and Web Applications. The HTML/CSS is rendered using the Jinja2 templating language for Python [27].

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>AI Companion</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/template.css') }}">
    <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.0/jquery.min.js">
    </script>
    <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/paho-mqtt/1.0.1/mqttws31.js">
    </script>
    <script type="text/javascript" src="{{ url_for('static', filename='javascript/mqttConfig.js') }}">
    </script>
    <script type="text/javascript" src="{{ url_for('static', filename='javascript/tempGraph.js') }}">
    </script>
    <script type="text/javascript" src="{{ url_for('static', filename='javascript/pressureGraph.js') }}">
    </script>
    <script type="text/javascript" src="{{ url_for('static', filename='javascript/altitudeGraph.js') }}">
    </script>
  </head>
  <body>
    <header>
      <div class="container">
        <h1 class="logo">AI Companion</h1>
        <strong>
          <nav>
            <ul class="menu">
              <li><a href="{{ url_for('home') }}">Home</a></li>
              <li><a href="{{ url_for('GUI') }}">GUI</a></li>
              <li><a href="{{ url_for('portfolio') }}">Portfolio</a></li>
              <li><a href="{{ url_for('login') }}">Logout</a></li>
            </ul>
          </nav>
        </strong>
      </div>
    </header>
    {% block content %}
    {% endblock %}
  </body>
</html>
```

Figure 21 Jinja Template HTML code

13.5 GUI Page

If the user selects the GUI page in the navigation bar, they'll see a live stream of the robot's camera, along with button controls to control the movement of the robot.

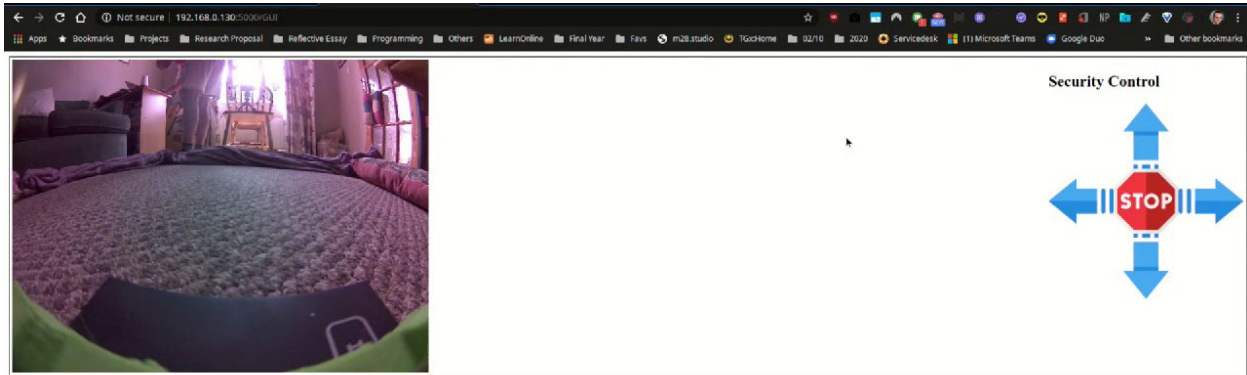


Figure 22 GUI interface

A code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The code is written in a light blue/grey monospace font. It defines five click events for buttons with IDs #forward, #left, #stop, #right, and #backward. Each event triggers an \$.ajax() call with a specific URL and 'get' type.

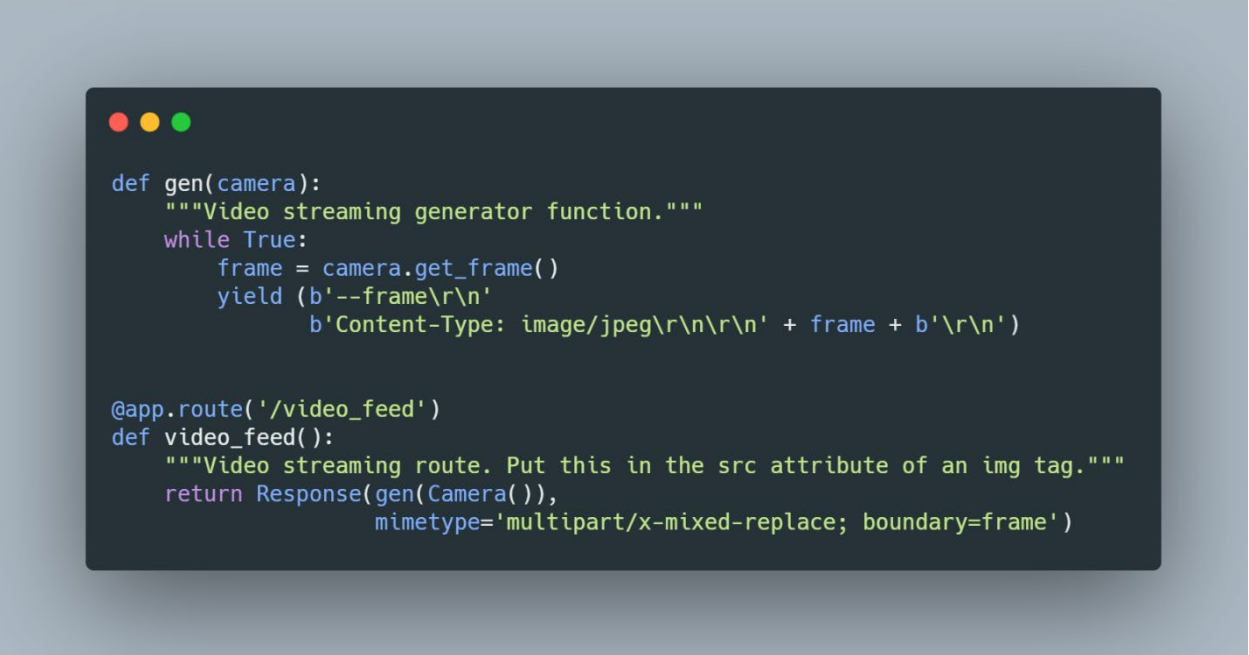
```
$(function() {  
    $('#forward').click(function() {  
        $.ajax({  
            url: '/companion/forward',  
            type: 'get',  
        })  
    })  
  
    $('#left').click(function() {  
        $.ajax({  
            url: '/companion/left',  
            type: 'get',  
        })  
    })  
  
    $('#stop').click(function() {  
        $.ajax({  
            url: '/companion/stop',  
            type: 'get',  
        })  
    })  
  
    $('#right').click(function() {  
        $.ajax({  
            url: '/companion/right',  
            type: 'get',  
        })  
    })  
  
    $('#backward').click(function() {  
        $.ajax({  
            url: '/companion/backward',  
            type: 'get',  
        })  
    })  
})
```

Figure 23 Remote controls

Each button is configured to a dynamic variable route on the Flask server, which performs the required movement on the Robot instance. This operation is carried out using Ajax GET requests.

13.6 Flask Video Streaming

Flask server that uses a streaming response to provide a stream of video frames captured from a camera in Motion JPEG format. This code was adapted from a series of brilliant tutorials and explanations of video streaming on Flask by Miguel Grinberg [28].

A code editor window with a dark background and light-colored text. It contains Python code for a Flask video streaming application. The code includes a generator function 'gen' that yields video frames as JPEG images, and a route 'video_feed' that returns a streaming response using the generator.

```
def gen(camera):  
    """Video streaming generator function."""  
    while True:  
        frame = camera.get_frame()  
        yield (b'--frame\r\n'  
              b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')  
  
@app.route('/video_feed')  
def video_feed():  
    """Video streaming route. Put this in the src attribute of an img tag."""  
    return Response(gen(Camera()),  
                    mimetype='multipart/x-mixed-replace; boundary=frame')
```

Figure 254 Video Streaming



Figure 25 Portfolio Sentiment Analyser Output

The portfolio sentiment analyser works by taking the user's portfolio (located as a text file on the server) and utilising two API's to firstly garner the correct company information, then use a news API to get the latest news mentioning that company in any given timeframe. These news headings are then vectorised (converted from words to integers and floats, needed for an ML algorithm and used to get a prediction of -1(Negative), 0 (Neutral) or +1 (Positive) from my Portfolio Sentiment Analysis algorithm.

```

with open('model/newsSentiment.joblib', 'rb') as f:
    model = load(f)

headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko)
    Chrome/41.0.2228.0 Safari/537.36'
}

with open("stocks.txt", "r") as ins:
    count = 1
    for line in ins:
        ticker = line
        aggregate = "Insufficient Data"

        try:
            # Query for the stock name, for refined news queries.
            resp = requests.get(
                url="https://www.alphavantage.co/query?function=SYMBOL_SEARCH&keywords={}"
                .format(ticker), headers=headers)
            data = resp.json()
            #print(data)
            companyName = data['bestMatches'][0]['2. name']
            print("Parsing news for : " + companyName)

            #Query for news
            resp = requests.get(
                url="https://newsapi.org/v2/everything?"
                'q={}&'
                'from=2020-04-05' # This is the OLDEST date an article can be from, free edition
                'sortBy=popularity&' #Filter by popularity (read the newsapi docs)
                'apiKey=fe0115ceffe418988616191b03e1c74'.format(
                    ticker + " " + companyName), headers=headers) #Add the company name in full after
            # the ticker, for more accurate news queries
            data = resp.json()

            for article in data['articles']:
                articleCount = articleCount + 1
                newsUrl = article['url']
                #print(newsUrl)
                newsArticle = article['title']
                articles.append(newsArticle)
                #print(newsArticle)

            #print("Stock Analysis: " + ticker + " , " + aggregate + " , " + str(int(time.time()))) #unix
            timestamp at the end

            # Prevent throttling. Can make one request per 172.8 seconds as we have 500 requests a day.
            time.sleep(10) #Should be 173 seconds but we don't have time for that

        except Exception as e:
            print("Error: " + str(e))
            time.sleep(10)

    ins.close();

# Create the pandas DataFrame
df = pd.DataFrame({'headlines': articles})
news = df.to_json(orient='index')
#df.to_json(r'model/exportedNews.json', orient='split')

fresh_news = df;
vectorizer_new_data = CountVectorizer(max_features=30, min_df=0)
processed_features_new_data = vectorizer_new_data.fit_transform(fresh_news['headlines']).toarray()

prediction = model.predict(processed_features_new_data)
print(prediction)
lists = prediction.tolist()
my_json_string = json.dumps(lists)

for i in range(len(prediction)):
    if prediction[i] == 1:
        resultsPos.append(fresh_news['headlines'][i])
        overall += 1;
        print("Positive: " + fresh_news['headlines'][i])
    if prediction[i] == -1:
        resultsNeg.append(fresh_news['headlines'][i])
        overall -= 1;
        print("Negative: " + fresh_news['headlines'][i])

```

Figure 26 Portfolio Sentiment Analyser Output

14 Conclusion

The outcome of the project is considered a success. The project has a functional web interface giving the user a live video stream, live incoming data, remote robot controls as well as the portfolio sentiment analyser. The collision avoidance and object detection functionalities are working as intended. The facial recognition is working, yet the location of the camera makes it incredibly difficult to read faces.

All sensor data has been measured and verified to be operating correctly, along with incredibly low latency between readings on the microprocessor being received by the web client.

The user has a stable, easy-to-use, simplistic, yet powerful interface to remotely control the robot.

The project has plenty of scope for future developments, including; turning the sentiment analyser into an API, incorporating the collision avoidance, object detection features into buttons within the web interface, and turning the live video stream into a live facial recognition/ object recognition video stream.

15 References

- [1] Nvidia, "Jetbot", <https://github.com/NVIDIA-AI-IOT/jetbot>, 2019
- [2] Leanna Garfield, "We will all have personal robot assistants within the next decade ", <https://www.businessinsider.com/personal-assistant-robots-are-the-future-2016-3?r=US&IR=T>, 2016
- [3] Tractica, "Consumer Robotics", <https://tractica.ondia.com/research/consumer-robotics/>, 2019
- [4] PCMag, "Virtual Assistant", <https://www.pcmag.com/encyclopedia/term/virtual-assistant>, 2019
- [5] Statista, "Number of digital voice assistants in use worldwide from 2019 to 2023", <https://www.statista.com/statistics/973815/worldwide-digital-voice-assistant-in-use/>, 2019
- [6] Alex Riley, "How your smart home devices can be turned against you", <https://www.bbc.com/future/article/20200511-how-smart-home-devices-are-being-used-for-domestic-abuse>, 2020
- [7] David Thomas, "Artificial Intelligence Investing Gets Ready For Prime Time", <https://www.forbes.com/sites/greatspeculations/2017/10/25/getting-ready-for-prime-time-of-artificial-intelligence-investing/#7b37e1e42010>, 2017
- [8] Michelle Knight, "What is Machine Learning?", <https://www.dataversity.net/what-is-machine-learning/>, 2017
- [9] Michael Copeland, "What's the Difference Between Artificial Intelligence, Machine Learning and Deep Learning?", <https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/>, 2016
- [10] MQTT, "Overview", mqtt.org/, 2018
- [11] Flespi, "MQTT v HTTP", flespi.com/blog/http-vs-mqtt-performance-tests, 2018
- [12] Adafruit, "ESP32", <https://www.adafruit.com/product/3320>, 2020
- [13] Arduino, "Overview", www.arduino.cc/, 2018
- [14] JetBrains, "PyCharm", <https://www.jetbrains.com/pycharm/>, 2020

- [15] Microsoft, "Azure Notebook", <https://notebooks.azure.com/help/introduction>, 2020
- [16] Jupyter, "Jupyter Notebook", <https://jupyter.org/>, 2020
- [17] Danny Jost, "What is an IR Sensor?", <https://www.fierceelectronics.com/sensors/what-ir-sensor>, 2019
- [18] Adafruit, "BMP280", <https://www.adafruit.com/product/2651>, 2020
- [19] Elaine Wu, "NVIDIA Jetson Nano Developer Kit Detailed Review", <https://www.seeedstudio.com/blog/2019/04/03/nvidia-jetson-nano-developer-kit-detailed-review/>, 2019
- [20] TutorialsPoint, "Flask Overview", https://www.tutorialspoint.com/flask/flask_overview.htm, 2019
- [21] Nvidia, "Collision Avoidance Example", <https://github.com/NVIDIA-AI-IOT/jetbot/wiki/examples>, 2019
- [22] Adam Geitgey, "Face Recognition", <https://pypi.org/project/face-recognition/>, 2019
- [23] HiveMQ, "MQTT essentials", <https://www.hivemq.com/blog/mqtt-essentials-special-mqtt-over-websockets>, 2018
- [24] HiveMQ, "MQTT over Websockets", <https://www.hivemq.com/blog/mqtt-over-websockets-with-hivemq>, 2018
- [25] Canvas JS, "Overview", canvasjs.com, 2018.
- [26] Canvas JS, "Live Charts", canvasjs.com/javascript-charts/, 2018.
- [27] w3, "HTML/CSS", [w3.org/standards/webdesign/htmlcss](https://www.w3.org/standards/webdesign/htmlcss), 2018.
- [28] Miguel Grinberg, "Flask Video Streaming Revisited", <https://blog.miguelgrinberg.com/post/flask-video-streaming-revisited>, 2017

16 Appendix

All code available at : https://cozy133.github.io/proj_eng/

