PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)

100-ft Ring Road, Bengaluru – 560 085, Karnataka, India

Report on

‘AUGMENTED REALITY ENABLED IoT WAREHOUSE’

Submitted by

TEJAS RAO (PES12017010106)
VAISHAK KRISHNA (PES1201701678)
RUTVIK AJIT JERE (PES12017010833)

Aug 2020 - May 2021

under the guidance of

Internal Guide

Prof. SUNIL KUMAR
PROFESSOR
Department of ECE
PES University
Bengaluru -560085

FACULTY OF ENGINEERING

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGG

PROGRAM B. TECH

CERTIFICATE

This is to certify that the Report entitled

‘AUGMENTED REALITY ENABLED SMART HOME’

is a bonafide work carried out by

TEJAS RAO (PES12017010106)

VAISHAK KRISHNA (PES1201701678)

RUTVIK AJIT JERE (PES12017010833)

In partial fulfillment for the completion of 8th semester course work in the Program of Study B.Tech in Electronics and Communication Engineering, under rules and regulations of PES University, Bengaluru during the period Aug 2020– May 2021. It is certified that all corrections/suggestions indicated for internal assessment have been incorporated in the report. The report has been approved as it satisfies the 7th and 8th semester academic requirements in respect of Capstone project work.

Signature with date & Seal
(Prof Sunil Kumar)
Internal Guide

Signature with date & Seal
Dr. Anuradha M
Chairperson

Signature with date & Seal
Dr. B. K. Keshavan
Dean - Faculty of Engg. &Technology

Name and signature of the examiners:

1.

2.

DECLARATION

We, Tejas Rao, Rutvik Ajit Jere and Vaishak Krishna**,** hereby declare that the report entitled, '***AUGMENTED REALITY ENABLED IoT WAREHOUSE,*** is an original work done by us under the guidance of **Prof. SUNIL KUMAR**, *PROFESSOR,* ECE Department and is being submitted in partial fulfillment of the requirements for completion of 7th and 8th Semester course work in the Program of Study, B.Tech in Electronics and Communication Engineering.

PLACE:
DATE:

NAME AND SIGNATURE OF THE CANDIDATES

1. Tejas Rao

2. Rutvik Ajit

3. Vaishak Krishna

SYNOPSIS

The Internet of Things (IoT) has urged in a new era of automation, controllability and has redefined Machine - to - Machine (M2M) communication. An IoT network refers to a mesh of devices all connected and communicating through the internet in real time. The key feature of IoT being the fact that it can be leveraged to make routine operations ‘smart’ coupled with a large scope of applications has made it a widely discussed topic in technology today.

On the other hand, augmented reality is technology that allows us to augment virtual assets in real time to provide a simulation effect and enhance the physical world. Through this project, our goal is to integrate these two fore technologies together and apply the same in a warehousing environment.

As a use case of IoT and AR in a warehouse we have found that it is vital to keep certain environment variables in check in a such an environment and thus we have implemented a sensor circuit to monitor and update these variables in real time and using AR to display this information in through AR.

Similar we have made an effort to espouse AR in the process of diagnosis and maintenance of equipment, in this case a VFD. AR can be used to intuitive make information more accessible than traditional ways thus improving efficiency of the entire process.

Finally we look to develop an augmented reality application for measuring distances given that this is a common task and can be made easier through the use of image targets and anchors in unity and Vuforia.

ACKNOWLEDGEMENT

This project has enabled us to learn vast knowledge in the fields of Iot and Augmented Reality, it would not have been possible without the kind support and help of many individuals and organizations. We would like to extend our sincere thanks to all of them.

We are highly indebted to Prof Sunil Kumar, PES University for his guidance, constant supervision, co-operation and support throughout the course of this project.

We express our gratitude towards Dr Anuradha M, (HoD) of the ECE department, PES University for providing us with this opportunity.
We would also like to express special gratitude and thanks to our friends and colleagues for their inputs and encouragement.
Finally, our thanks and appreciations also go to the Electronics and Communications Department of PES University and their faculty for allowing us to use department resources.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1 – Introduction

With the world moving towards a consumer age, warehousing has become an integral part for all companies. A warehouse can be defined as a building for storing materials, equipment and manufactured goods. The need for better technology increased manifold with the scale of the environment and leveraging this technology to improve efficiency and reduce physical effort is the need of the hour.

IoT has become commonplace in today’s world. IoT forms the backbone of any automation process and can improve efficiency and reduce error in a warehousing environment. IoT can be leveraged for inventory, tracking, remote control of machinery and equipment and even be used in data analytics to collect and transmit data from one node to another.

Augmented Reality is an up-and-coming technology which can cater to a large set of needs but is yet to be explored to its full potential. The major strength of AR is the visual stimulus it provides, making for an immersive experience to the user.
Augmented reality combined with a wearable head – mounted display can redefine the way information and data are visualized.

Through this project our intent is to combine these two technologies and apply it in a warehousing environment. Data available through a visual stimulus eliminated the need for screen display devices and using IoT for machine-to-machine communication ensures a seamless experience. Thus, the idea is to connect devices through IoT and provide the user with AR experiences such as AR manual, training or control and provide a comprehensive platform where they can engage with this data or information in the through the physical world.

1.1 Problem Statement

We propose the integration of AR and IoT in a complementary way, connecting devices and collecting data through IoT and making AR scalable to cover objects everywhere with an acceptable level of performance for interacting with IoT devices and enable an immersive visualization of the data.

For this project we have identified three areas where IoT can be integrated within a house. These areas being -

1. ***Environment Monitoring*** - A system to monitor and track the ambient parameters like temperature, humidity and air quality. This is used to ensure the environment is optimal and safe.

2. ***Appliance Diagnostics*** – To leverage AR technology to aid the process of diagnostics of warehouse machinery. In this project we have implemented a Variable Frequency Drive (VFD) and use AR to make diagnosing and fixing problems easier.

3. ***RulAR*** – An Augmented Reality distance measurement application allowing distances to be measured between two points virtually.

CHAPTER 2 – Literature Survey

We conducted a computerized search for publications in the online digital libraries of the Association for Computing Machinery (ACM), Institute of Electrical and Electronics Engineers (IEEE) and Multi-Disciplinary Publishing Institute (MDPI) and identified the following papers to base our project on.

1. A paper published by Dongsik Jo, Department of Digital Contents Engineering, Wonkwang University, Jeonbuk 54538, Korea proposes the integration of AR and IoT in a complementary way, making AR scalable to cover objects everywhere with an acceptable level of performance and interacting with IoT in a more intuitive manner.
   Link To paper on the MDPI website

This paper outlines three key components to combining AR and IoT which are

- Distributed Object Centric Management
- IoT Object-guided tracking
- Seamless Interaction

Augmented reality is useful for displaying data through a tangible interface making comprehension easy and thus helpful in day-to-day environments. Furthermore the ability to be able to control IoT devices through this tangible interface is vital.
Further the paper outlines simple use cases where IoT and AR can be married together for better experiences.

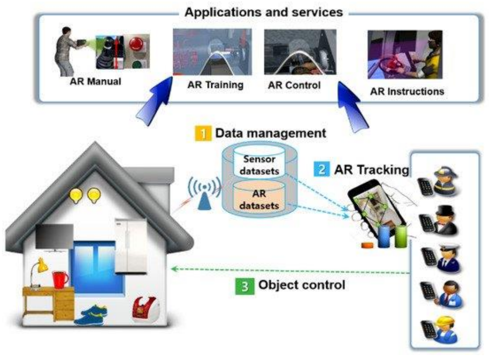


Fig 1.1.1 – Taken from 'AR Enabled IoT for a Smart and Interactive Environment: A Survey and Future Directions

Further the paper also outlines the importance of human – real world interaction through augmented information adding a new dimension to the common everyday interactions.
An in-situ experiment was also concluded where small lamps are controlled through augmented reality without ever physically touching the lamps themselves as an example to how IoT and AR can make everyday interactions immersive.
Finally the paper discusses the various ways of AR interactions such as virtual buttons, virtual copies of real world objects etc.

2. Gary White, Christian Cabrera, Andrei Palade, Siobhan Clarke in their paper "***Augmented Reality in IoT"*** have elaborated on the rise of integrating immersive environments into IoT to provide contextual information to service users and providers in context aware applications.
   Link to paper on Semantic Scholar site

This paper describes the use of augmented reality in IoT to provide contextual information to users and also discusses a AR application and ways in which users can interact with AR.

Other notable work going in integrating AR and IoT :

1. Amazon has come out with Amazon Sumerian - its own software for developing and deploying AR and VR mobile applications.
AR enabled IoT networks for retail shopping called **Amazon GO** already deployed in cities like Seattle and San Francisco.
2. Google Lens, one of Google's flagship products incorporates mixed reality and IoT to provide smart solutions in a number of fields.
3. Tesla is conducting research into converging AR and IoT within cars to allow the driver to visualize various sensor data, in real time.

CHAPTER 3 - Project Fundamentals

3.1 Project Fundamentals

3.1.1 ESP8266

The ESP8266 Wi-Fi Module is a self-contained SOC with integrated TCP/IP protocol stack that can give any microcontroller access to your Wi-Fi network. The ESP8266 is capable of either hosting an application or offloading all Wi-Fi networking functions from another application processor. In this project we will use the ESP8266 as the main microcontroller for all circuits.

3.1.2 AdaFruit IO

AdaFruit IO is a cloud service where users can create and customize IoT dashboards to send and receive data from. All IoT devices in this project are controlled through this platform.

3.2 Augmented Reality

Augmented Reality is the superimposition of the digital world over the real physical world. It is achieved through the visual digital assets overlayed on our field of view of the physical world enabling for an enhanced and interactive visual stimulus for comprehending any sort of data or information.

3.2.1 Vuforia

Vuforia Engine is a software development kit (SDK) to provide a platform for developing augmented reality applications for mobile devices. Vuforia leverages advanced computer vision to enable object tracking and recognition and provides the user with a simple and less complex wrapper class with these functionalities to be used and a robust AR experience. Thus, allowing for users to develop application with little prior knowledge in image processing.

Vuforia is widely popular due to it’s large, unparalleled support for devices including many lower spec devices.

3.2.2 Unity Engine

Unity is 3D/2D game engine which provides a powerful cross-platform IDE to develop AR applications. Unity provides a framework for features such as rigid body physics, collision and 3D rendering which are useful in defining interaction between digital assets amongst each other and with the user as well. Unity also allows applications to be developed across multiple operating systems such as android, iOS and even TvOS.

3.3 Message Queueing Telemetry Transport (MQTT)

MQTT is an OASIS standard messaging protocol for the Internet of Things (IoT). It is designed as a lightweight publish/subscribe messaging model for connecting remote devices. Its main advantages are that it has minimal network bandwidth due to the minimal amount of data being transmitted and a small code footprint and provides resource-constrained network clients with a simple way to distribute telemetry information, thus making it an ideal choice for M2M communication and IoT.

The MQTT protocol defines two types of network entities: a message broker and a number of clients. An MQTT broker is a server that receives all messages from the clients and then routes the messages to the appropriate destination clients. In this project the Adafruit IO server acts as the mqtt broker and the ESPs act as clients.

3.3.1 Publish/ Subscribe Model

MQTT protocol segregates data into ‘topics’, e.g., a topic called temperature. Clients can ‘publish’ or upload data under that topic on the cloud platform.
Concurrently, clients can ‘subscribe’ to various topics and in lieu of this subscription the broker automatically forwards any change in data under that topic to the subsequent clients who have subscribed to it. This allows for a way for the clients to receive data without having to periodically request for it. Fig 3.3.1 depicts a simple publish/subscribe architecture.

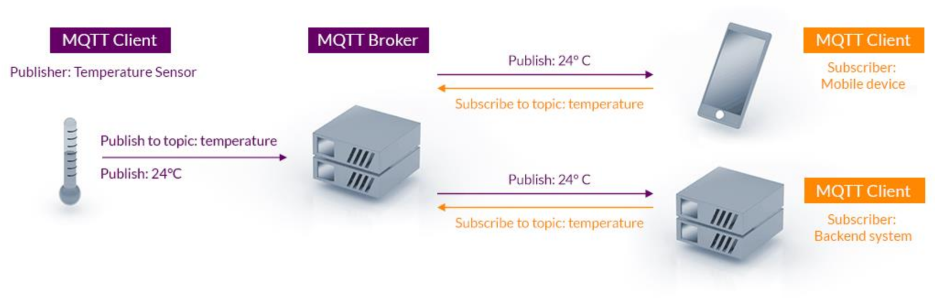

MQTT Client
Publisher: Temperature Sensor
MQTT Broker
Publish to topic: temperature
Publish: 24°C
Publish: 24° C
Subscribe to topic: temperature
MQTT Client
Subscriber: Mobile device
Publish: 24° C
Subscribe to topic: temperature
MQTT Client
Subscriber: Backend system

Fig 3.3.1 - Pub/sub model for a topic ‘temperature’

3.4 Methodology - IoT Client

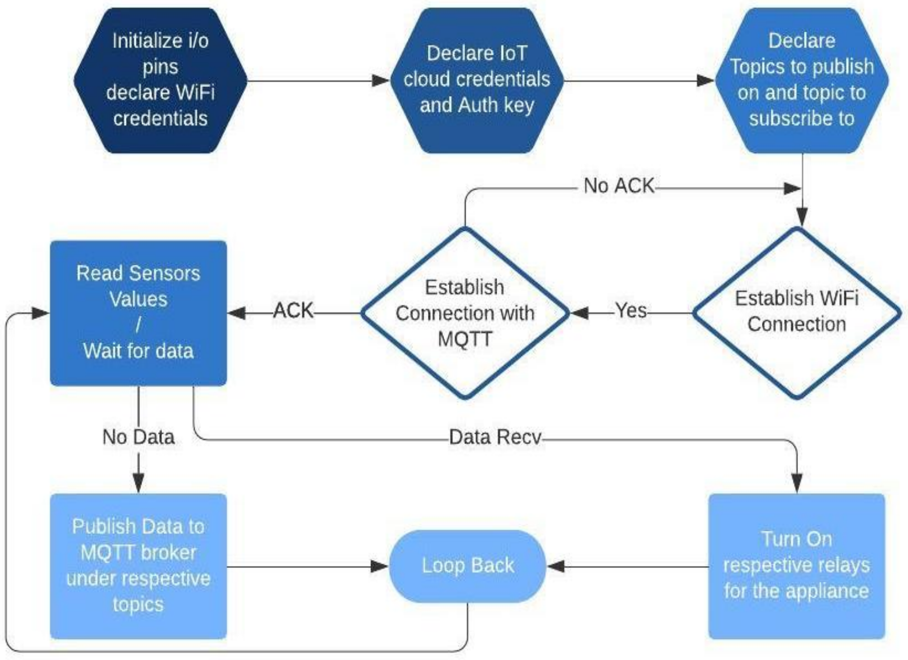

Initialize i/o pins declare WiFi credentials
Declare IoT cloud credentials and Auth key
Declare Topics to publish on and topic to subscribe to
No ACK
Establish WiFi Connection
Yes
Establish Connection with MQTT
ACK
Read Sensors Values / Wait for data
No Data
Data Recv
Publish Data to MQTT broker under respective topics
Loop Back
Turn On respective relays for the appliance

Fig 3.4.1- Flowchart - IoT client

Fig 3.4.1 depicts the methodology of any IoT client to send/ receive i.e publish/ subscribe to data on the cloud platform

- ❖ The client must first connect to the local Wi-Fi network
- ❖ Topics which the particular client should publish to and subscribe must be declared explicitly.
- ❖ Establish connection with the MQTT broker i.e. AdaFruit IO using the user's authentication ID and password.
- ❖ Ping the connection periodically to ensure the ESP is always connected to the broker, if the ping does not receive an ACK, re-establish connection.
- ❖ Once connected, the client can write data to the topics defined earlier under publish.
- ❖ Any updation of data under a subscribed topic is forwarded to the client by the broker without the client having to request for that data.

3.5 Methodology – AR Application

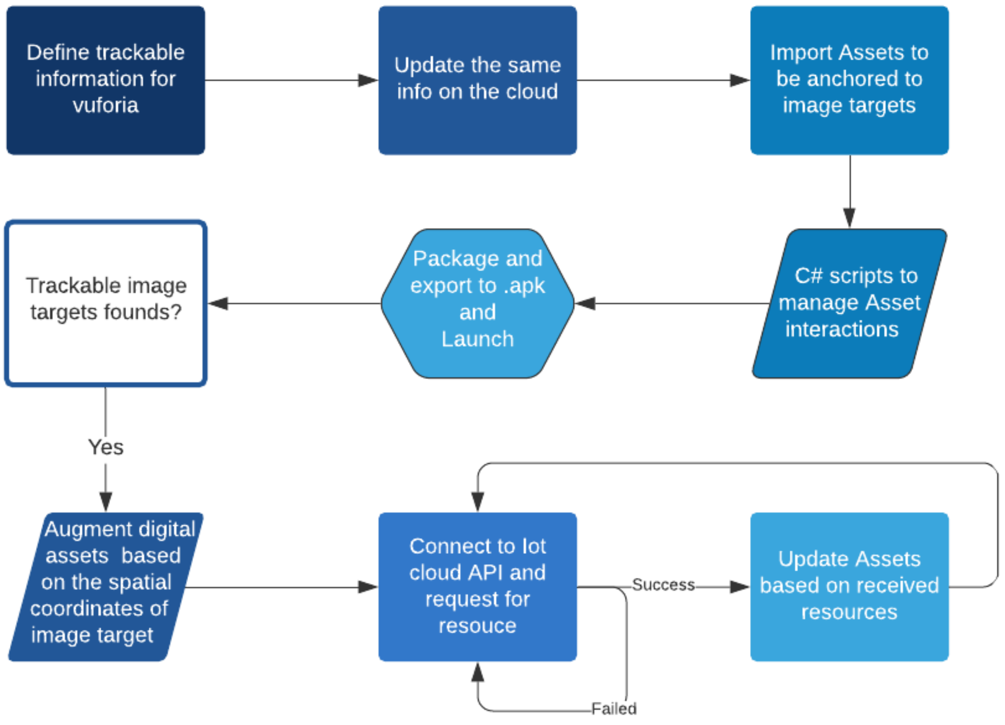

Define trackable information for vuforia
Update the same info on the cloud
Import Assets to be anchored to image targets
C# scripts to manage Asset interactions
Package and export to .apk and Launch
Trackable image targets founds?
Yes
Augment digital assets based on the spatial coordinates of image target
Connect to Iot cloud API and request for resouce
Success
Update Assets based on received resources
Failed

Fig 3.5.1- Flowchart – AR Application

Fig 3.5.1 depicts a high-level program flow of the mobile application.

1. Instantiate an image target object and tag an image from your vuforia cloud account to it. This will be the image that the application will recognize.
2. Add/import digital assets into unity to be augmented upon recognition of the image target and ensure their spatial arrangement with respect to the image target matches requirements.
3. Writing C# scripts to define and control the behavior of digital assets amongst themselves and with the user. This includes function definitions for each asset, update functions and event triggers.
4. Declare the API credentials to be connected to and define function definitions for GET, POST, PUT and DELETE requests to be sent over HTTP.

5. Update assets and interactions based on the API responses received to reflect the data on the cloud platform.

3.6 Application UI

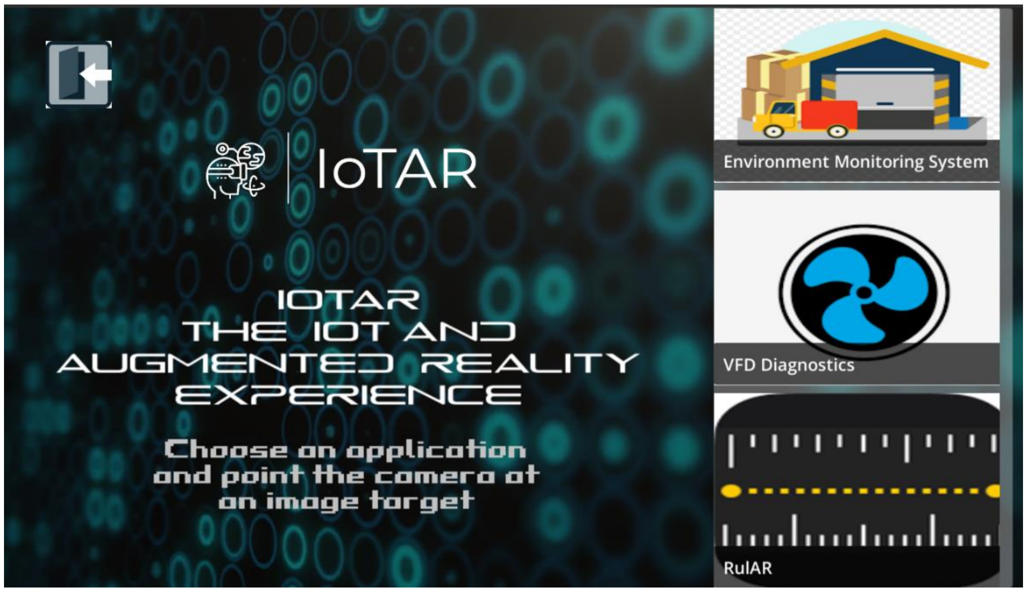

IoTAR
IOTAR
THE IOT AND
AUGMENTED REALITY
EXPERIENCE
Choose an application and point the camera at an image target
Environment Monitoring System
VFD Diagnostics
RulAR

Fig 3.6.1 – App UI

The Intro Scene UI of the mobile application

The three buttons on the right lead to the respective scenes where each scene fulfills one use case which are –

1. Environment Monitoring System
2. VFD Diagnostics
3. RulAR (Distance Measurement using AR)

These use cases will be covered in detail in the subsequent chapters.

CHAPTER 4 - Design

4.1 Environment Monitoring:

With the environment monitoring use case, we aim to measure physical quantities such as temperature, humidity and the air quality of the home. Further, these measurements shall be displayed on the cloud using MQTT protocols and subsequently on the mobile app through augmented digital assets.

4.1.1 Sensors

1. DHT11 sensor:

   DHT11 is a temperature and humidity.

   To measure humidity, it contains a humidity sensing component which contains two electrodes and a moisture holding substrate between them. As the humidity changes, the resistance between the electrodes changes. This signifies that there has been a change in humidity.

   To measure temperature, the DHT11 uses a thermistor which changes its resistance according to the temperature. The resistance varies with the temperature inversely.

2. MQ 135 gas sensor:

   MQ 135 sensor like the DHT11 sensor is also an electrochemical sensor. It contains tin dioxide substrate (SnO_2) which has a higher resistance in clear air. However when there is an increase in a hazardous gas the resistance of the substrate decreases, showing us that there is a change in the air quality. The gases which the MQ135 sensor can detect are:

   1. Ammonia (NH_3)
   2. Carbon Dioxide (CO_2)
   3. Oxides of nitrogen (NO_x)
   4. Benzene (C_6H_6)
   5. Carbon Monoxide (CO)

To measure the presence of a particular gas in parts per million (ppm), we need to make use of the graph given below. This is a graph of Rs/R0 in the x-axis and gas measured in ppm in the y-axis. Also, this graph is a log-log graph.

To measure Rs (sensor resistance in at various concentrations of gases)

To measure Rs (sensor resistance in at various concentrations of gases)

```
RS = [(VC x RL) / VRL] - RL
```

Vc is the sensor voltage. RL is the load resistance and VRL is the analog output converted to Volts.

R0(sensor resistance at 100ppm of NH3 in clear air) is measured similarly as Rs however, V is taken as an average of a certain amount of values.

The ratio = Rs/R0

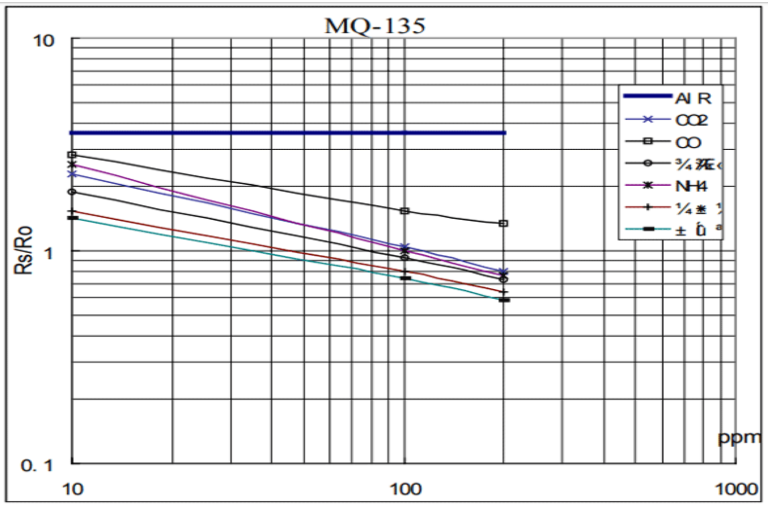


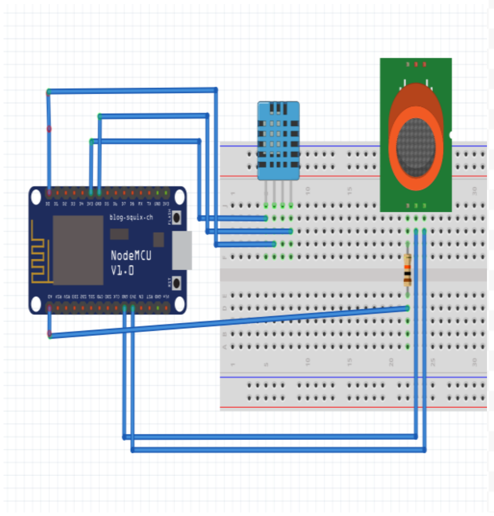
Fig 4.1.1

Log graph for PPM for MQ135 sensor

Initial conditions: Temp = 20C, Humidity = 65%, O2 concentration = 21%, RL = 20kohms

Image: https://components101.com/sensors/mq135-gas-sensor-for-air-quality

4.1.2 Schematic and Flow chart



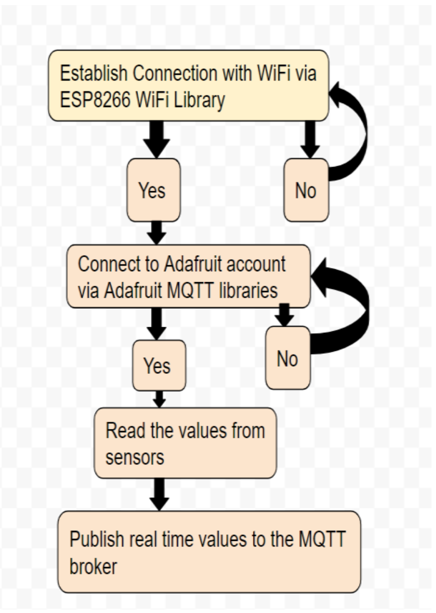

Establish Connection with WiFi via ESP8266 WiFi Library
Yes
No
Connect to Adafruit account via Adafruit MQTT libraries
Yes
No
Read the values from sensors
Publish real time values to the MQTT broker

Fig 4.1.2 – Env. Monitoring Schematic *Fig 4.1.3 – Env. Monitoring Program Flow*

4.1.3 Methodology:

- Initialize the Wi-Fi SSID and its password into variables. Followed by the MQTT brokers IP address, port and account details.
- We then set up the class Client and call the functions to publish the sensor variables we wish to see on the cloud.
- During the setup phase of the code, we to establish connection with the Wi-Fi, failing which we try again
- We then call the recursive function connect which establishes connection with the MQTT broker. If the function returns 0, we have established the connection. If not, the code goes back into the loop.
- We then read the values received by the sensors and use the Publish function to publish it onto the cloud.

4.2 VFD Diagnostics

4.2.1 Introduction

In this section we will outline the working of a Variable Frequency Drive or VFD which is commonly used in warehousing to drive large motor loads.
We have implemented a VFD and propose the use of augmented reality to help in the process of diagnosing and rectifying problems that might occur and believe this process can be made simpler by leveraging the visual stimulus offered by augmented reality.

4.2.2 Variable Frequency Drive(VFD)

A VFD is a type of motor controller that drives an electric motor by varying the frequency and voltage of the AC input supplied to the electric motor.

4.2.2.1 Advantages Of VFDs

1. Slow start capability - A load can be started at a desired low voltage thus allowing the appliance to gradually attain full load reducing wear
2. Low power demand on startup
3. Controlled Acceleration - The load is allowed to gradually attain speed from zero thus increasing longevity.
4. Adjustable operating speed - By controlling the frequency of the driving signal a VFD can operate at variable speeds according to the need.
5. Energy savings - By not running the appliance at full load, power consumed is reduced.

4.2.3 Working Principle

A typical VFD consists of four sections shown in Fig 3.2.3.1 and Fig 3.2.3.2

Rectifier Circuit:

A Rectifier is used for changing the incoming alternating current (AC) supply to direct current (DC). In our project we are using a bridge rectifier IC to convert input AC power into DC.

Intermediate Circuit:

The intermediate circuit consists of a large capacitor to filter the output from the rectifier and remove AC ripples and store DC power.

Inverter Circuit:

This section comprises electronic switches like transistors, IGBTs, etc. It receives DC power from the DC link and converts into AC which is delivered to the motor. It uses modulation techniques like pulse width modulation to vary output frequency for controlling the speed of the induction motor.

Control Circuit:

It consists of a microprocessor unit to control the switching frequency of the inverter circuit to generate PWM signals.

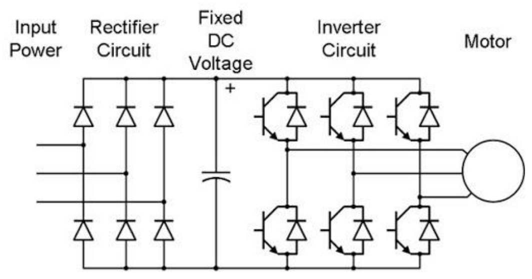


Fig 4.2.1 - VFD high level Schematic

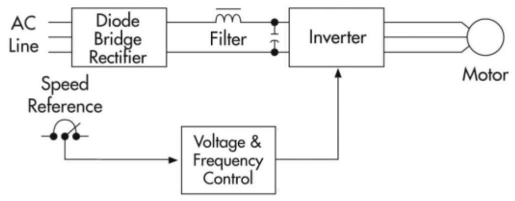


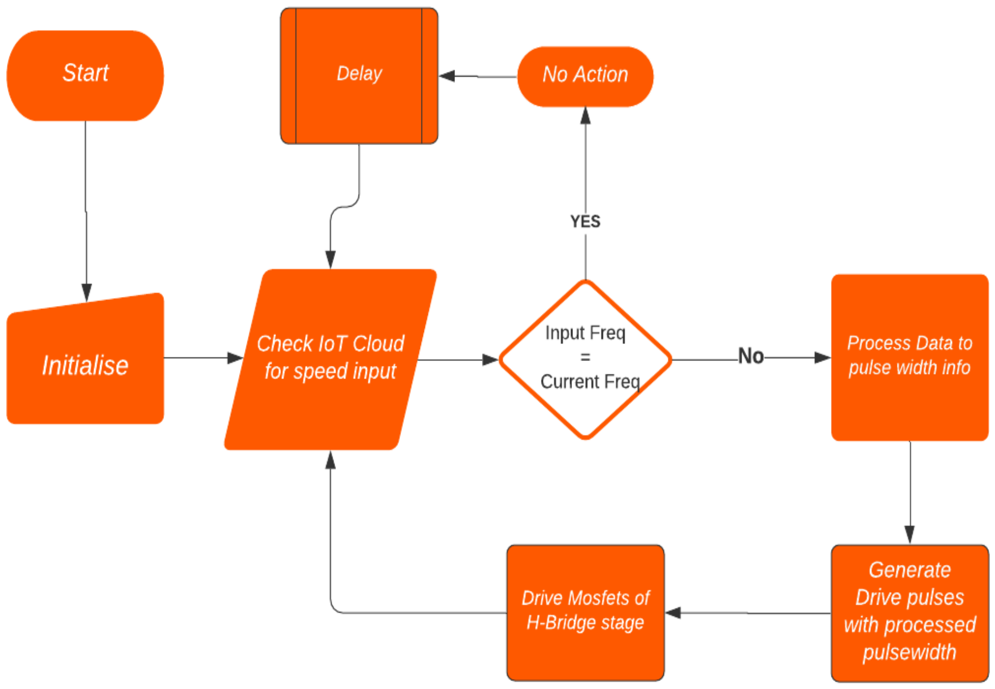
Fig 4.2.2 Equivalent schematic of VFD control unit
Source: Reference [3] - Fig. 4.

4.2.4 VFD to Control Speed

In this project we are using a VFD to control the speed of a fan. This is a typical PWM VFD, like the name suggests, it uses sine PWM waves to control average voltage at the output.

In this section we will cover the basic methodology followed to establish cloud connection, publish/ subscribe values and finally a brief about the working of the circuit and its elements.

Fig 4.2.3 A high level program flow for the VFD circuit.


Start
Delay
No Action
YES
Initialise
Check IoT Cloud for speed input
Input Freq = Current Freq
No
Process Data to pulse width info
Drive Mosfets of H-Bridge stage
Generate Drive pulses with processed pulsewidth

4.2.5 VFD Circuit Diagram

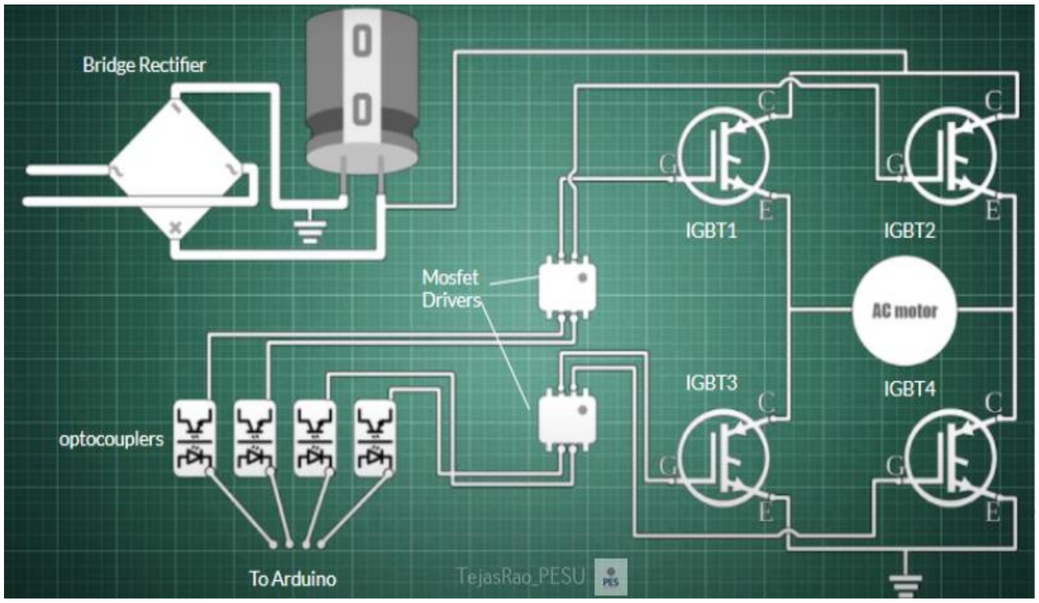


Fig 4.2.4 VFD Circuit Diagram

1. 220v 50Hz AC input is converted into 311v DC by the bridge rectifier IC. A 680uF 480v capacitor is used to filter the DC signal. This forms the rectifier and intermediate circuit.

2. The Arduino. Mosfet drivers and IGBTs form the inverter circuit. The idea is to produce an AC wave from the converted DC but of desired frequency by switching the IGBTs alternatively.

3. Optocouplers are used to isolate low voltage and high voltage in the circuit to protect the Arduino. Mosfet drivers IR2101 are used to scale the Arduino voltage of 5v to the required gate voltage to turn the IGBTs on.

4. To create the positive half cycle IGBTs 1 and 4 are turn ON and the others OFF, current enters the motor through IGBT 1 and exists through IGBT 4. Similarly for the negative half cycle IGBTs 2 and 3 are turned ON and the others OFF, current enters the motor through IGBT 2 and IGBT 3, which is the reverse direction.

5. The IGBTs are driven using Pulse Width Modulated signals from the Arduino to create a sinusoid at the output. Fig 3.5.2 depicts the effect of the pulse widths and switching

delays to create high frequency and low frequency waves while also controlling the average voltage.

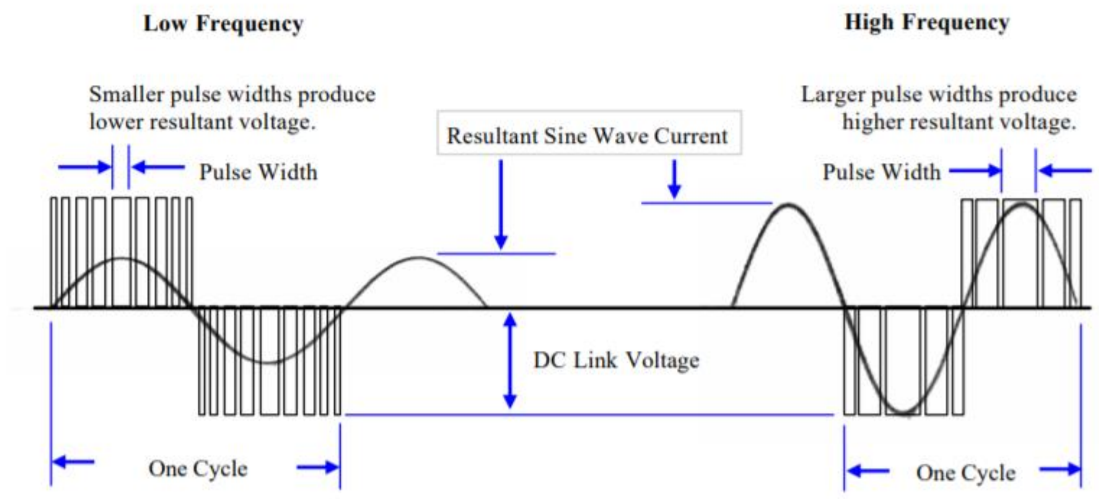

Low Frequency
High Frequency
Smaller pulse widths produce lower resultant voltage.
Larger pulse widths produce higher resultant voltage.
Resultant Sine Wave Current
Pulse Width
Pulse Width
DC Link Voltage
One Cycle
One Cycle

Fig 4.2.5 - PWM Sine Wave Synthesis

1. Smaller pulse widths result in lesser average voltage thus the resulting sinusoid has a smaller peak, larger pulse widths result in a higher average voltage thus the resulting sinusoid has a larger peak.
2. Frequency of the output wave is controlled by the switching speeds of the IGBTs. Hence lesser the duration between switching, higher the out frequency and vice-versa.
3. Thus, we obtain an AC signal of desired frequency and voltage at the output to run the motor and this also allows us to maintain a constant voltage to frequency ratio which is necessary to prolong the life of the motor.

4.2.5.1 Frequency - Speed Relationship

The relationship between speed and frequency of a motor is given by the following equation

$$Motor\ Speed\ (RPM) = \frac{60\ x\ f}{Pairs\ of\ Poles} = \frac{60\ x\ f}{Poles/2} = \frac{120\ x\ f}{Poles}$$

Therefore, an increase in frequency causes the rotor to complete one cycle in between the poles faster, thus increasing the rpm.

4.3 RulAR

4.3.1 Introduction

RulAR is an augmented reality distance measuring app which uses the spatial coordinates of virtual assets to find the physical distance between them along the three coordinates axes.
The idea is to simplify the process of measuring distances between two points using just an application instead of physically measuring using tools. Unity is a comprehensive engine for game development and in any a player's spatial coordinates and movements are constantly tracked. This same technology can be leveraged to measure physical distance through augmented reality assets and Vuforia.

4.3.2 Methodology

1. The logic behind using AR to measure distances is to instantiate two unique image targets and provide an anchor object to both targets.
2. These anchors will provide a reference point for each image targets, as the user moves the image targets physically, we obtain the spatial coordinates of these anchor points.
3. These coordinates are relative to the coordinates of the camera on the mobile device.
4. Now, the difference between these two anchors across the three coordinate axes are calculated and this relative distance between the two points is indeed the actual physical distance as well.
5. This is enabled by unity natively where it can scale the application environment according to the real-world environment and since we use relative distance between the two points, the virtual and physical distances are the same.

4.3.3 Unity Setup

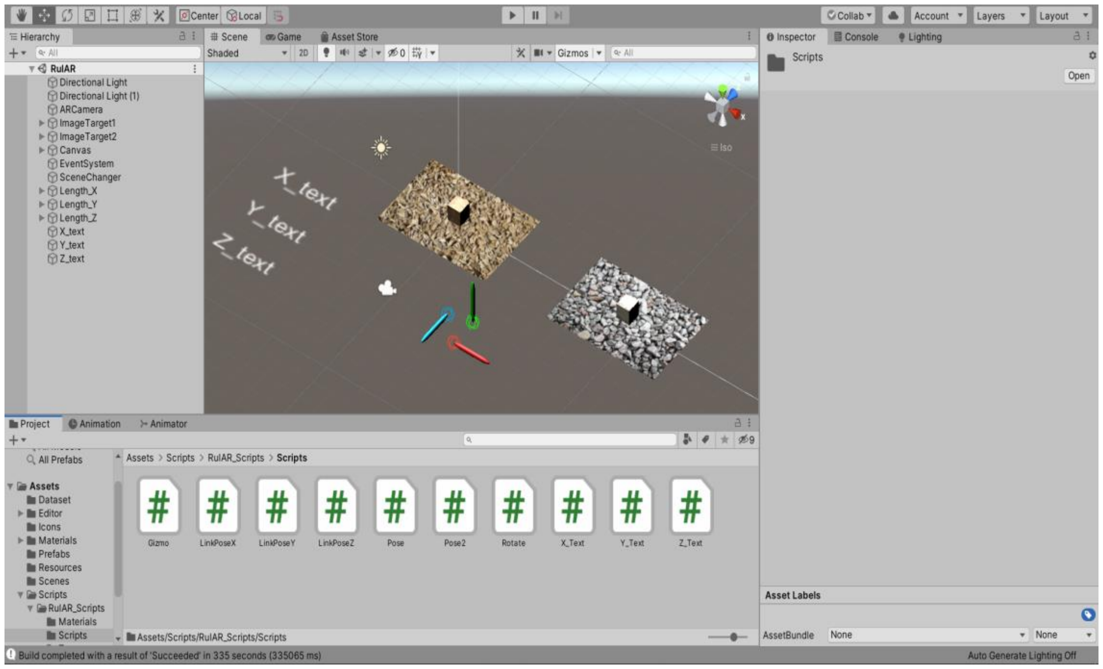


Fig 4.3.1
Unity Environment for RulAR use case

The setup involves two unique image targets. Each configured with a uniquely identifiable image.
A simple cube Game Object is an anchor for each image target.

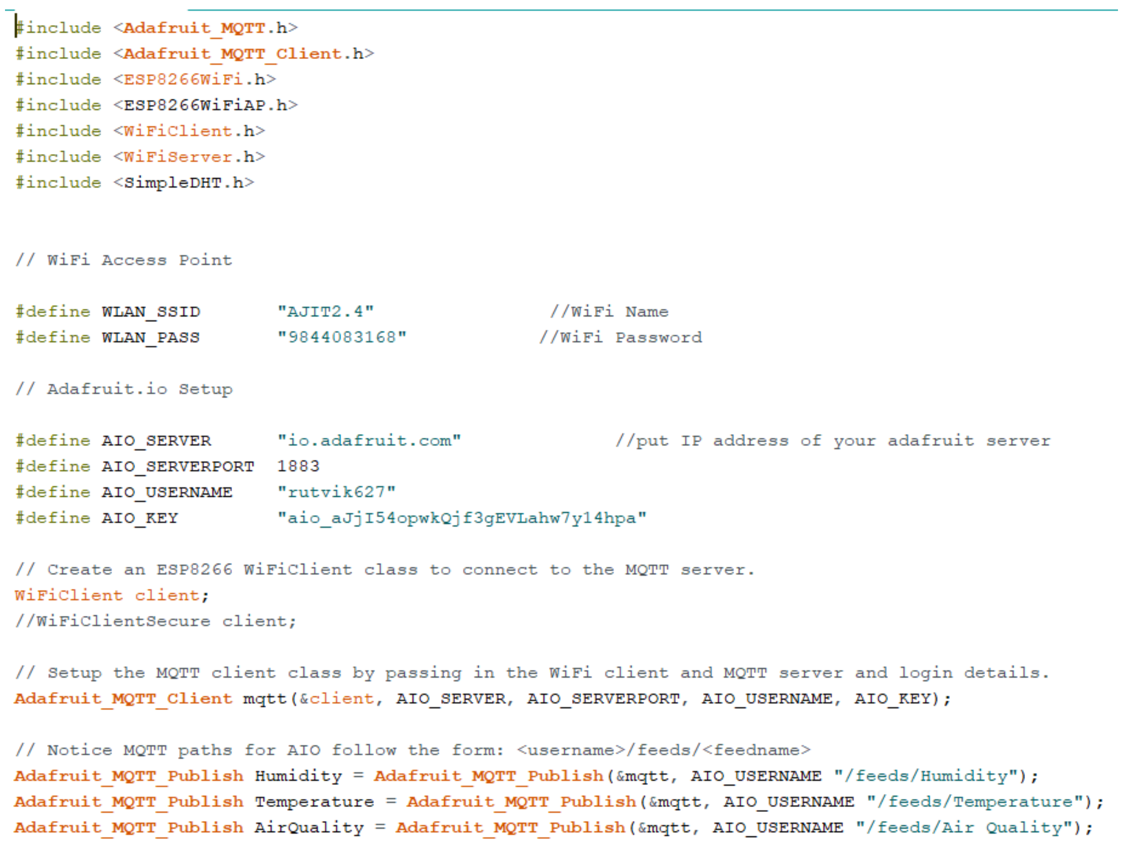
Now, a capsule shaped game object is created, this will be used to visually connect the two image targets and also display the measured value across it. This needs to be scripted in such a way that it’s length is dynamic and always fits the space in-between the cube game objects regardless of the distance between them.
The same process is repeated to create such game objects for all three coordinate axes i.e. X-Axis, Y-Axis and the Z-Axis.
Additionally text variable to display the distance value are created for each axes and scripted to change it’s values dynamically to show the measured distance.

CHAPTER 5 – Implementation and Code

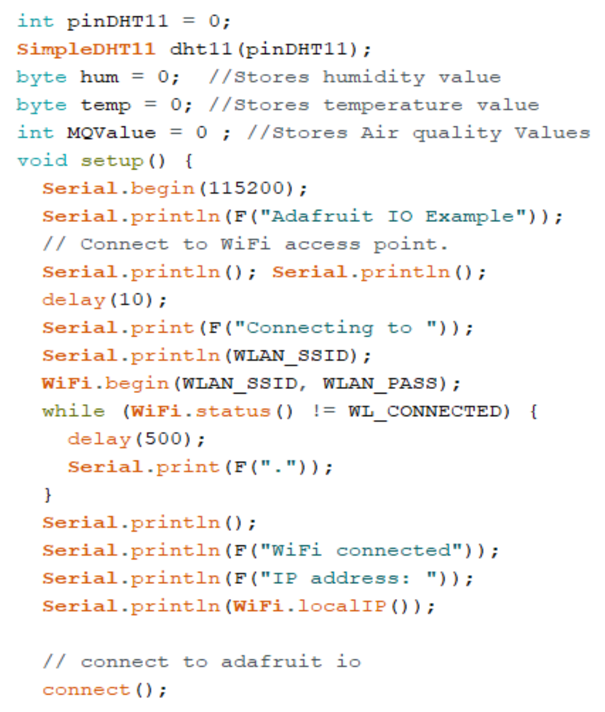
5.1 Environment Monitoring:

5.1.1 Code Implementation:

- Initialize the Wi-Fi SSID and its password into variables. Followed by the MQTT brokers IP address, port and account details.
- In this section of the code, we declare the Wi-Fi and Adafruit credentials. Then we initialize the feeds we wish to publish to, in this case it is the humidity, temperature and air quality.
- We declare these feeds in the code with the respective API URLs along with the Adafruit username.
- During the setup phase of the code, we to establish connection with the Wi-Fi, failing which we try again until connection is successful
- We then call the recursive function **Connect()** which establishes connection with the MQTT broker. If the function returns 0, we have established the connection. If not, the code goes back into the loop while printing the cause and type of error if there is an error
- We then read the values received by the sensors and use the Publish function to publish it onto the cloud.
- For the air quality value, it is first converted to PPM using the graph in Fig 4.4.1 and subsequently publish that value to the cloud.



```
#include <Adafruit_MQTT.h>
#include <Adafruit_MQTT_Client.h>
#include <ESP8266WiFi.h>
#include <ESP8266WiFiAP.h>
#include <WiFiClient.h>
#include <WiFiServer.h>
#include <SimpleDHT.h>


// WiFi Access Point

#define WLAN_SSID       "AJIT2.4"                //WiFi Name
#define WLAN_PASS       "9844083168"            //WiFi Password

// Adafruit.io Setup

#define AIO_SERVER      "io.adafruit.com"              //put IP address of your adafruit server
#define AIO_SERVERPORT  1883
#define AIO_USERNAME    "rutvik627"
#define AIO_KEY         "aio_aJjI54opwkQjf3gEVLahw7y14hpa"

// Create an ESP8266 WiFiClient class to connect to the MQTT server.
WiFiClient client;
//WiFiClientSecure client;

// Setup the MQTT client class by passing in the WiFi client and MQTT server and login details.
Adafruit_MQTT_Client mqtt(&client, AIO_SERVER, AIO_SERVERPORT, AIO_USERNAME, AIO_KEY);

// Notice MQTT paths for AIO follow the form: <username>/feeds/<feedname>
Adafruit_MQTT_Publish Humidity = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/Humidity");
Adafruit_MQTT_Publish Temperature = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/Temperature");
Adafruit_MQTT_Publish AirQuality = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/Air Quality");
```



```
int pinDHT11 = 0;
SimpleDHT11 dht11(pinDHT11);
byte hum = 0;  //Stores humidity value
byte temp = 0; //Stores temperature value
int MQValue = 0 ; //Stores Air quality Values
void setup() {
  Serial.begin(115200);
  Serial.println(F("Adafruit IO Example"));
  // Connect to WiFi access point.
  Serial.println(); Serial.println();
  delay(10);
  Serial.print(F("Connecting to "));
  Serial.println(WLAN_SSID);
  WiFi.begin(WLAN_SSID, WLAN_PASS);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(F("."));
  }
  Serial.println();
  Serial.println(F("WiFi connected"));
  Serial.println(F("IP address: "));
  Serial.println(WiFi.localIP());

  // connect to adafruit io
  connect();
```

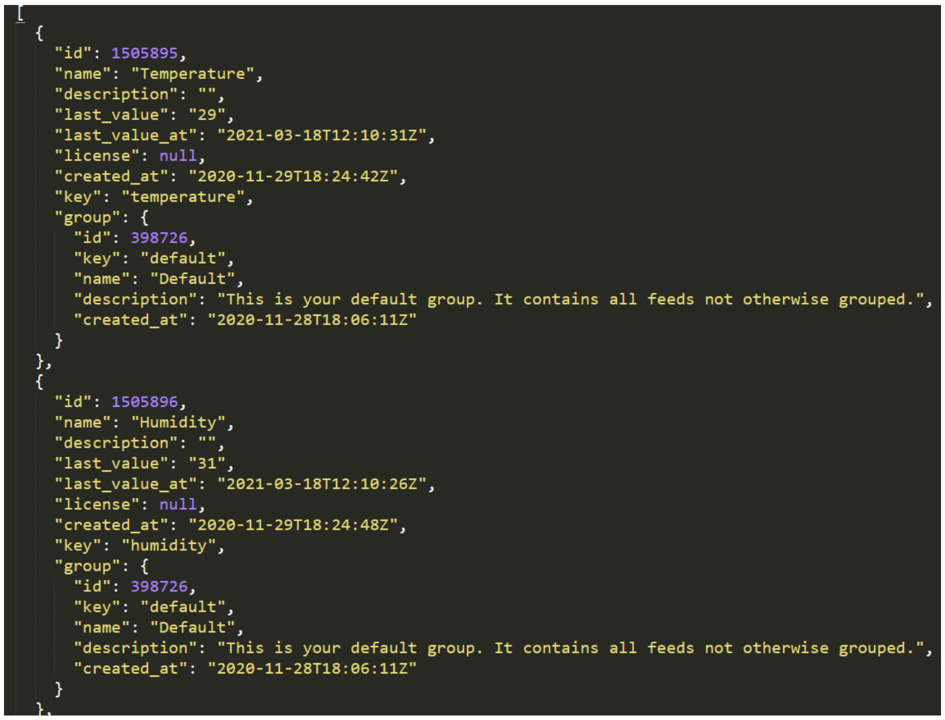
```
// connect to adafruit io via MQTT
void connect() {
  Serial.print(F("Connecting to Adafruit IO... "));
  int8_t ret;
  while ((ret = mqtt.connect()) != 0) {
    switch (ret) {
      case 1: Serial.println(F("Wrong protocol")); break;
      case 2: Serial.println(F("ID rejected")); break;
      case 3: Serial.println(F("Server unavail")); break;
      case 4: Serial.println(F("Bad user/pass")); break;
      case 5: Serial.println(F("Not authed")); break;
      case 6: Serial.println(F("Failed to subscribe")); break;
      default: Serial.println(F("Connection failed")); break;
    }

    if(ret >= 0)
      mqtt.disconnect();

    Serial.println(F("Retrying connection..."));
    delay(10000);
  }
  Serial.println(F("Adafruit IO Connected!"));
}


void loop() {
  // ping adafruit io a few times to make sure we remain connected
  if(! mqtt.ping(30)) {
    // reconnect to adafruit io
    if(! mqtt.connected())
      connect();
  }
  dht11.read(&temp, &hum, NULL);
  Serial.print((int)temp); Serial.print(" *C, ");
  Serial.print((int)hum); Serial.println(" H");
  delay(5000);

   if (! Temperature.publish(temp)) {                    //Publish to Adafruit
      Serial.println(F("Failed"));
    }
      if (! Humidity.publish(hum)) {                      //Publish to Adafruit
      Serial.println(F("Failed"));
    }
    else {
      Serial.println(F("Sent!"));
    }


  //MQ135 sensor
  float m = -0.3376; //Slope
  float b = 0.7165; //Y-Intercept
  float R0 = 9.06;
  float sensor_volt; //Define variable for sensor voltag
  float RS_gas; //Define variable for sensor resistance
  float ratio; //Define variable for ratio
  int sensorValue = analogRead(MQValue); //Read analog v
  sensor_volt = sensorValue*(5.0/1023.0); //Convert anal
  RS_gas = ((5.0*10.0)/sensor_volt)-10.0; //Get value of
  ratio = RS_gas/R0;  // Get ratio RS_gas/RS_air
  float ppm_log = (log10(ratio)-b)/m; //Get ppm value in
  float ppm = pow(10, ppm_log);
  ppm = ppm*100;
  Serial.println(ppm);
  if(! AirQuality.publish(ppm))  {                    //
      Serial.println(F("Failed"));
    }
  else {
      Serial.println(F("AirQuality Data Sent")) ;
    }
}
```

5.1.2 AR Interface For Environment Monitoring

In this section we will discuss the unity and Vuforia environment for the environment monitoring use case.

The goal here is to retrieve data from AdaFruit cloud platform i.e., the temperature, humidity and air quality data and import it into the unity environment and subsequently display the information on a digital asset like a simple panel.
The application must first recognize an image target and on recognition activate the digital assets and establish a connection to the cloud API in the background and retrieve information.

To retrieve data, a GET request must be made to the cloud API under the specific feeds whose data is to be retrieved. The subsequent responses are in JSON format, an example is shown in Fig 4.1.3.1 below.



```
[
  {
    "id": 1505895,
    "name": "Temperature",
    "description": "",
    "last_value": "29",
    "last_value_at": "2021-03-18T12:10:31Z",
    "license": null,
    "created_at": "2020-11-29T18:24:42Z",
    "key": "temperature",
    "group": {
      "id": 398726,
      "key": "default",
      "name": "Default",
      "description": "This is your default group. It contains all feeds not otherwise grouped.",
      "created_at": "2020-11-28T18:06:11Z"
    }
  },
  {
    "id": 1505896,
    "name": "Humidity",
    "description": "",
    "last_value": "31",
    "last_value_at": "2021-03-18T12:10:26Z",
    "license": null,
    "created_at": "2020-11-29T18:24:48Z",
    "key": "humidity",
    "group": {
      "id": 398726,
      "key": "default",
      "name": "Default",
      "description": "This is your default group. It contains all feeds not otherwise grouped.",
      "created_at": "2020-11-28T18:06:11Z"
    }
  },
```

Fig 5.1.2
Response to a GET request containing all Feeds Information

The API credentials are present in the AdaFruit documentation. The response consists of data of all feeds on that particular account.

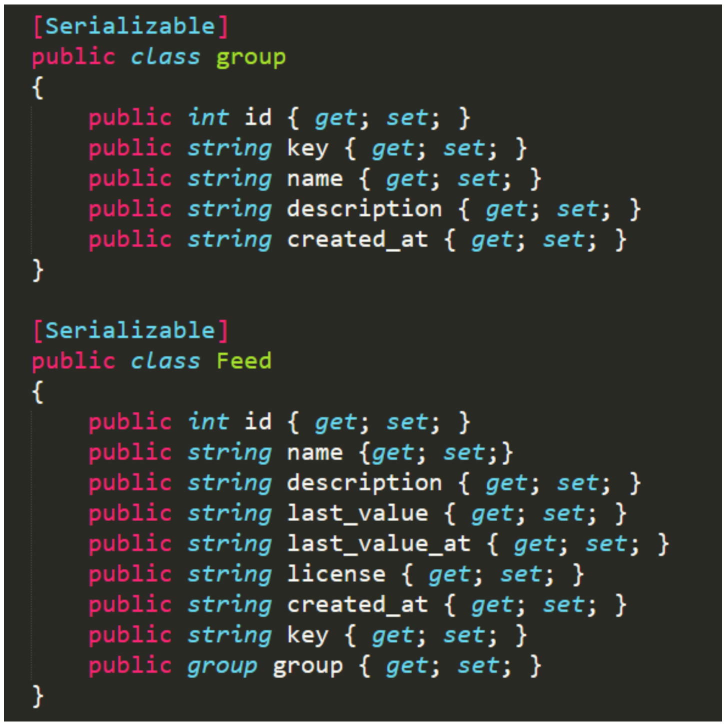
The 'ID', 'Name' and 'Key' headers are used to identify a given feed and the 'Last_Value' contains the most recent updated value under that field which is what is required for us to display.

Since unity does not support JSON natively, the JSON response is required to be populated into a C# class before it can be used.

Thus, a template or a wrapper class is first declared and an object of the that class is instantiated and populated with the data in the JSON leveraging Unity's library called JSONUtility.

This object can then be used to access any data pertaining to any of the feeds.

Fig 4.1.3.2 shows the definition of the C# wrapper class.



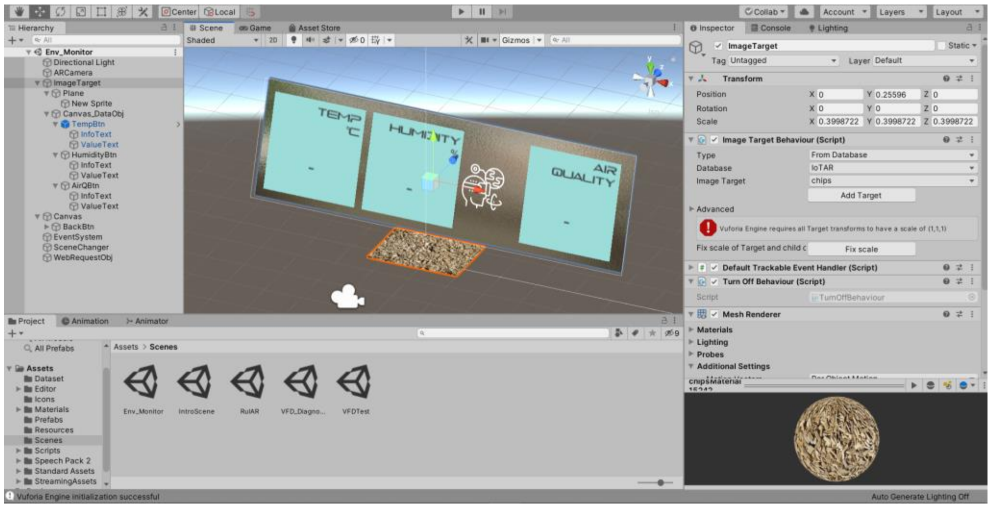
```
[Serializable]
public class group
{
    public int id { get; set; }
    public string key { get; set; }
    public string name { get; set; }
    public string description { get; set; }
    public string created_at { get; set; }
}

[Serializable]
public class Feed
{
    public int id { get; set; }
    public string name {get; set;}
    public string description { get; set; }
    public string last_value { get; set; }
    public string last_value_at { get; set; }
    public string license { get; set; }
    public string created_at { get; set; }
    public string key { get; set; }
    public group group { get; set; }
}
```

Fig 5.1.2 - Wrapper class to store JSON data.

The C# attribute [Serializable] is used here to declare these classes as serializable classes so that objects of this class can be written to a disk if the need arises in any future scope. These objects can again be deserialized and used in another program.

5.1.3 Code Methodology

1. Define the image target in the unity workspace.
2. Add digital assets on which sensor data is to be displayed, this includes the use of canvas', panels, buttons etc.
3. Ensuring spatial arrangement of said assets in order and sizes are taken relative to the size of the image target.
4. Writing C# scripts to define the initial state of assets and declare variable to store sensor data in.
5. Function definitions for querying cloud data through a GET web request, an update() function to update the scene after each frame and finally a function to change the color of the assets depending on the sensor values to clearly define a lower and upper threshold safety values for temperature and air quality.
6. C# class definition the response template to be written. Subsequently parsing the JSON web response and populating the C# wrapper classes.
7. Reading data members from the wrapper class and updating the assets through the update() function.



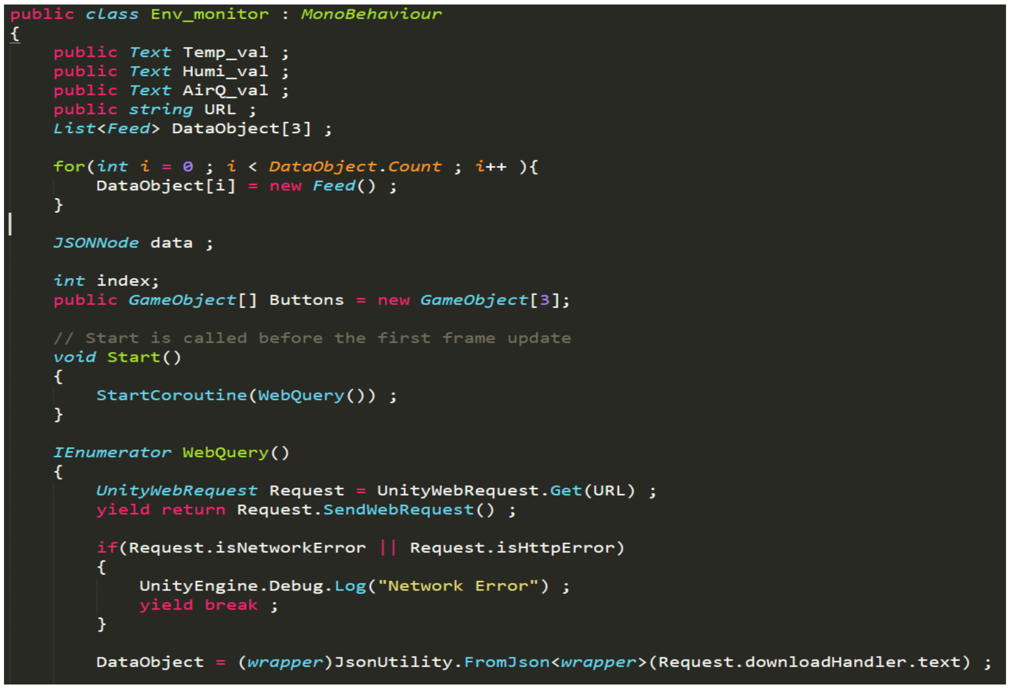
```
Hierarchy
Env_Monitor
  Directional Light
  ARCamera
  ImageTarget
    Plane
      New Sprite
    Canvas_DataObj
      TempBtn
        InfoText
        ValueText
      HumidityBtn
        InfoText
        ValueText
      AirQBtn
        InfoText
        ValueText
  Canvas
    BackBtn
  EventSystem
  SceneChanger
  WebRequestObj
```

```
Inspector
ImageTarget                         Static
Tag Untagged          Layer Default
Transform
Position     X 0           Y 0.25596     Z 0
Rotation     X 0           Y 0           Z 0
Scale        X 0.3998722   Y 0.3998722   Z 0.3998722
Image Target Behaviour (Script)
Type           From Database
Database       IoTAR
Image Target   chips
               Add Target
Advanced
Vuforia Engine requires all Target transforms to have a scale of (1,1,1)
Fix scale of Target and child c   Fix scale
Default Trackable Event Handler (Script)
Turn Off Behaviour (Script)
Script         TurnOffBehaviour
Mesh Renderer
Materials
Lighting
Probes
Additional Settings
chipsMaterial
```

```
Project  Assets > Scenes
Assets
  Dataset
  Editor
  Icons
  Materials
  Prefabs
  Resources
  Scenes
  Scripts
  Speech Pack 2
  Standard Assets
  StreamingAssets
Env_Monitor   IntroScene   RulAR   VFD_Diagno...   VFDTest
Vuforia Engine initialization successful
Auto Generate Lighting Off
```

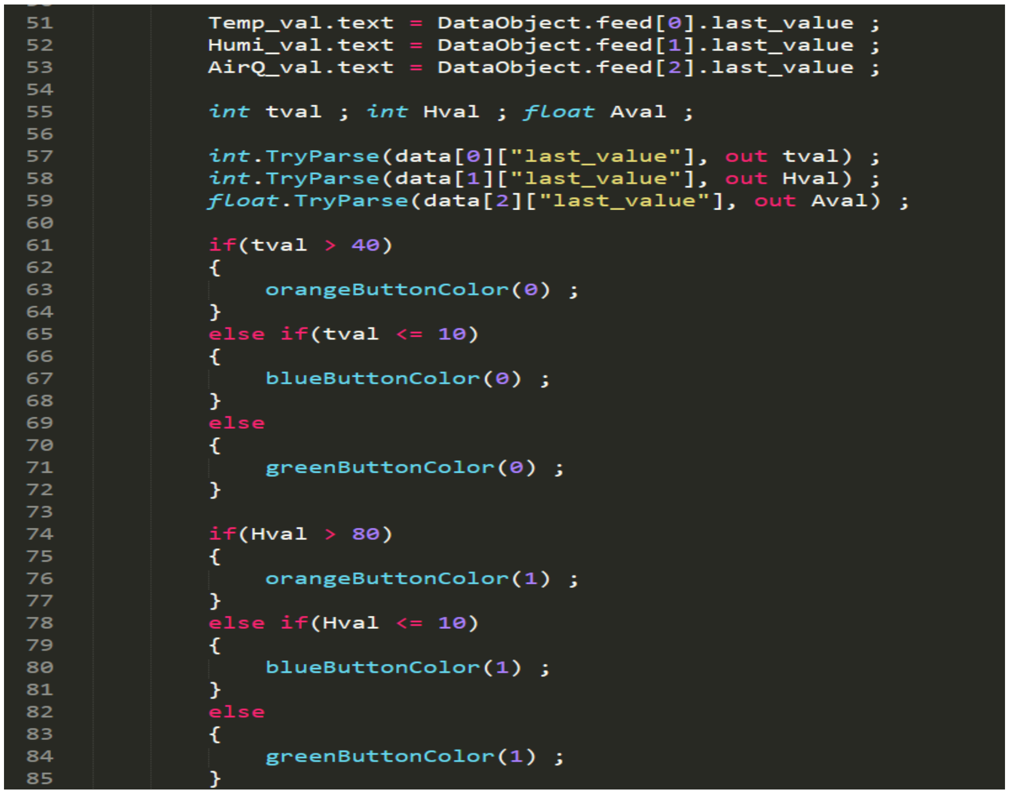
Fig 5.1.3 Unity workspace for Env Monitoring

5.1.4 Code



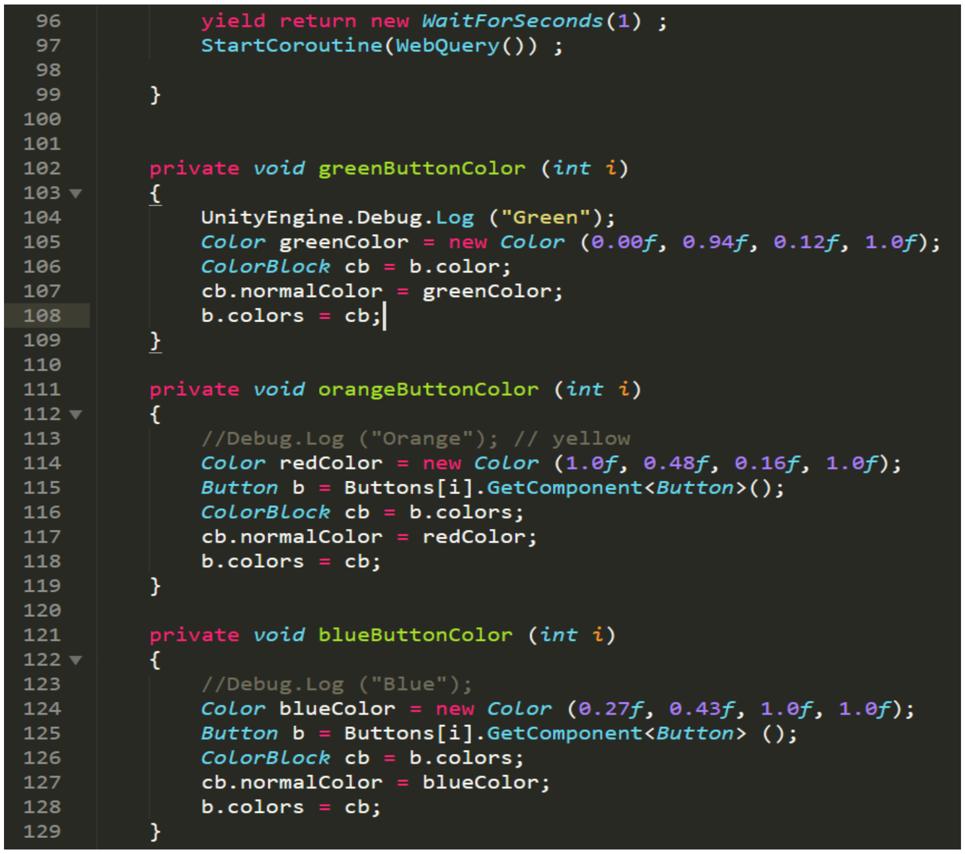
```
public class Env_monitor : MonoBehaviour
{
    public Text Temp_val ;
    public Text Humi_val ;
    public Text AirQ_val ;
    public string URL ;
    List<Feed> DataObject[3] ;

    for(int i = 0 ; i < DataObject.Count ; i++ ){
        DataObject[i] = new Feed() ;
    }

    JSONNode data ;

    int index;
    public GameObject[] Buttons = new GameObject[3];

    // Start is called before the first frame update
    void Start()
    {
        StartCoroutine(WebQuery()) ;
    }

    IEnumerator WebQuery()
    {
        UnityWebRequest Request = UnityWebRequest.Get(URL) ;
        yield return Request.SendWebRequest() ;

        if(Request.isNetworkError || Request.isHttpError)
        {
            UnityEngine.Debug.Log("Network Error") ;
            yield break ;
        }

        DataObject = (wrapper)JsonUtility.FromJson<wrapper>(Request.downloadHandler.text) ;
```

Fig 5.1.4 – GET request Function



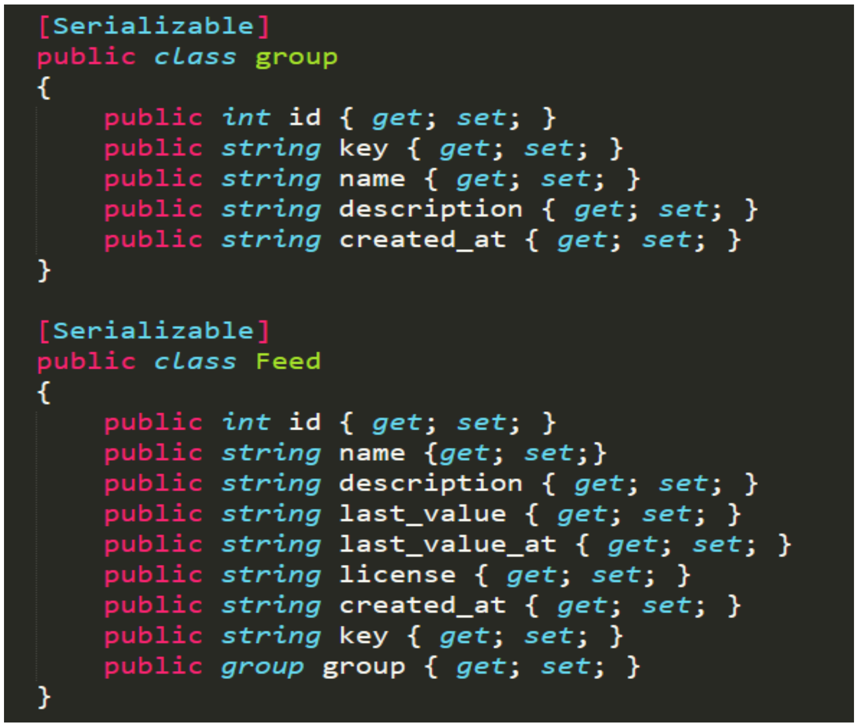
```
            Temp_val.text = DataObject.feed[0].last_value ;
            Humi_val.text = DataObject.feed[1].last_value ;
            AirQ_val.text = DataObject.feed[2].last_value ;

            int tval ; int Hval ; float Aval ;

            int.TryParse(data[0]["last_value"], out tval) ;
            int.TryParse(data[1]["last_value"], out Hval) ;
            float.TryParse(data[2]["last_value"], out Aval) ;

            if(tval > 40)
            {
                orangeButtonColor(0) ;
            }
            else if(tval <= 10)
            {
                blueButtonColor(0) ;
            }
            else
            {
                greenButtonColor(0) ;
            }

            if(Hval > 80)
            {
                orangeButtonColor(1) ;
            }
            else if(Hval <= 10)
            {
                blueButtonColor(1) ;
            }
            else
            {
                greenButtonColor(1) ;
            }
```

Fig 5.1.5 – Code to check whether read values are within normal range



```
        yield return new WaitForSeconds(1) ;
        StartCoroutine(WebQuery()) ;

    }


    private void greenButtonColor (int i)
    {
        UnityEngine.Debug.Log ("Green");
        Color greenColor = new Color (0.00f, 0.94f, 0.12f, 1.0f);
        ColorBlock cb = b.color;
        cb.normalColor = greenColor;
        b.colors = cb;
    }

    private void orangeButtonColor (int i)
    {
        //Debug.Log ("Orange"); // yellow
        Color redColor = new Color (1.0f, 0.48f, 0.16f, 1.0f);
        Button b = Buttons[i].GetComponent<Button>();
        ColorBlock cb = b.colors;
        cb.normalColor = redColor;
        b.colors = cb;
    }

    private void blueButtonColor (int i)
    {
        //Debug.Log ("Blue");
        Color blueColor = new Color (0.27f, 0.43f, 1.0f, 1.0f);
        Button b = Buttons[i].GetComponent<Button> ();
        ColorBlock cb = b.colors;
        cb.normalColor = blueColor;
        b.colors = cb;
    }
```

Fig 5.1.6 – Functions to change panel colors



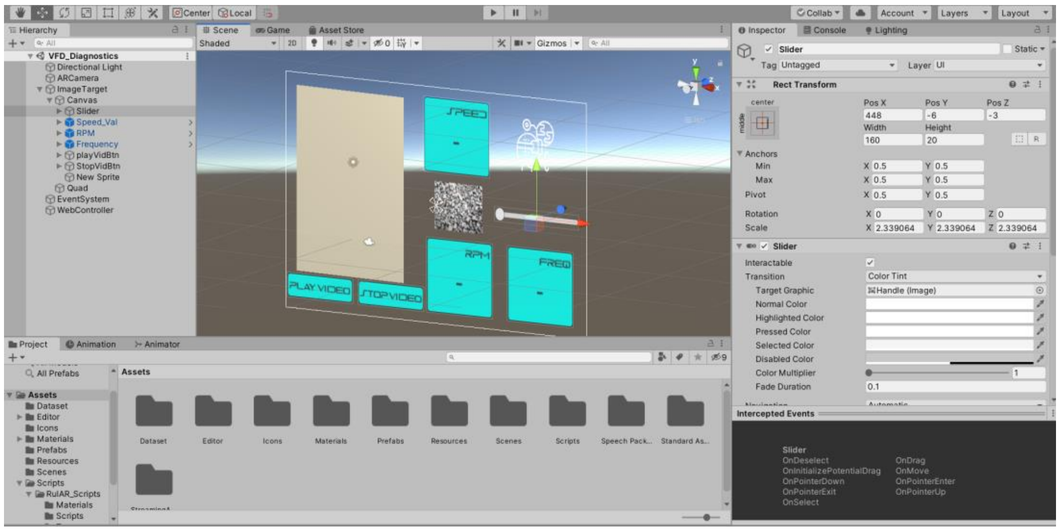
```
[Serializable]
public class group
{
    public int id { get; set; }
    public string key { get; set; }
    public string name { get; set; }
    public string description { get; set; }
    public string created_at { get; set; }
}

[Serializable]
public class Feed
{
    public int id { get; set; }
    public string name {get; set;}
    public string description { get; set; }
    public string last_value { get; set; }
    public string last_value_at { get; set; }
    public string license { get; set; }
    public string created_at { get; set; }
    public string key { get; set; }
    public group group { get; set; }
}
```

Fig 5.1.7 – Wrapper class to store JSON response

5.2 VFD Diagnostics - Implementation

5.2.1 VFD – AR Interface

In this section we outline the unity setup for the VFD diagnostics use case and the code for the same. Fig 5.2.1 is a snapshot of the unity environment. The idea behind this is to use AR to make certain useful information to the user i.e., through a visual stimulus to help make the information more comprehensible. By augmenting information about RPM, frequency, power rating etc. directly within the field of view of the user, the effort required in looking up this information separately is reduced. Also a video panel is enabled which can outline the common issues and solutions for the same with animated sequences with a virtual copy of the VFD will make the process of diagnosing and solving problems faster and simpler.



Slider — Tag: Untagged — Layer: UI
Rect Transform: center / middle; Pos X 448, Pos Y -6, Pos Z -3; Width 160, Height 20
Anchors: Min X 0.5 Y 0.5; Max X 0.5 Y 0.5
Pivot: X 0.5 Y 0.5
Rotation: X 0 Y 0 Z 0
Scale: X 2.339064 Y 2.339064 Z 2.339064
Slider: Interactable ✓; Transition: Color Tint; Target Graphic: Handle (Image); Normal Color; Highlighted Color; Pressed Color; Selected Color; Disabled Color; Color Multiplier 1; Fade Duration 0.1
Intercepted Events: Slider — OnDeselect, OnInitializePotentialDrag, OnPointerDown, OnPointerExit, OnSelect, OnDrag, OnMove, OnPointerEnter, OnPointerUp

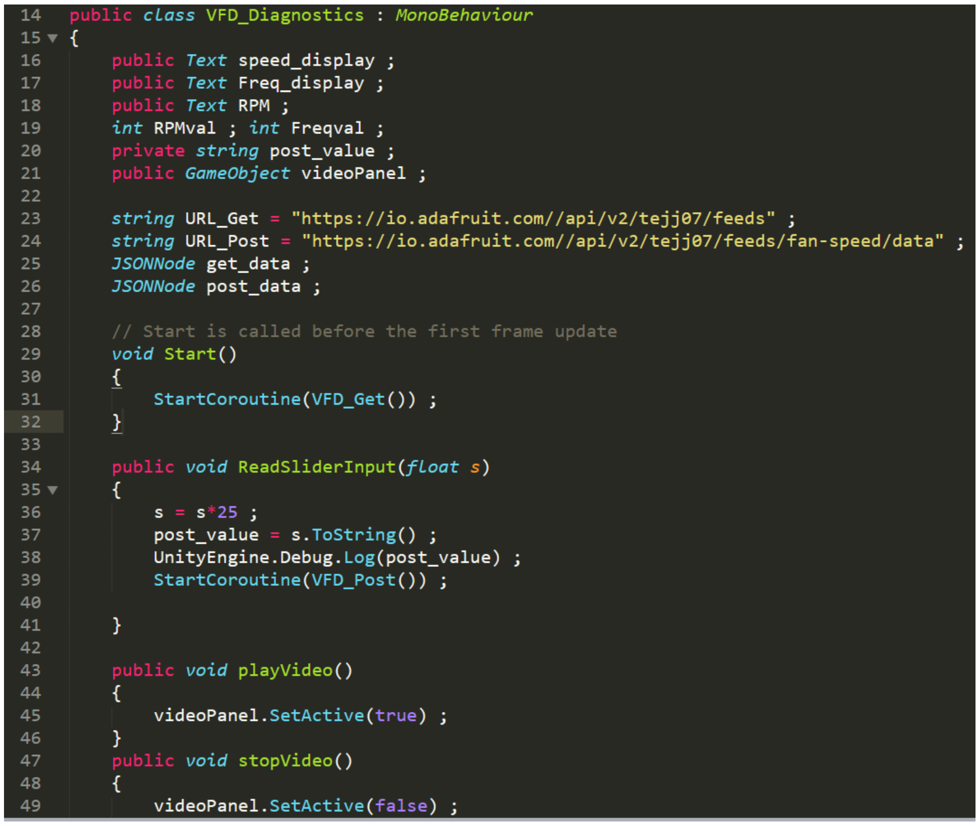
Fig 5.2.1
Unity environment for VFD diagnosis use case

5.2.2 Code Implementation

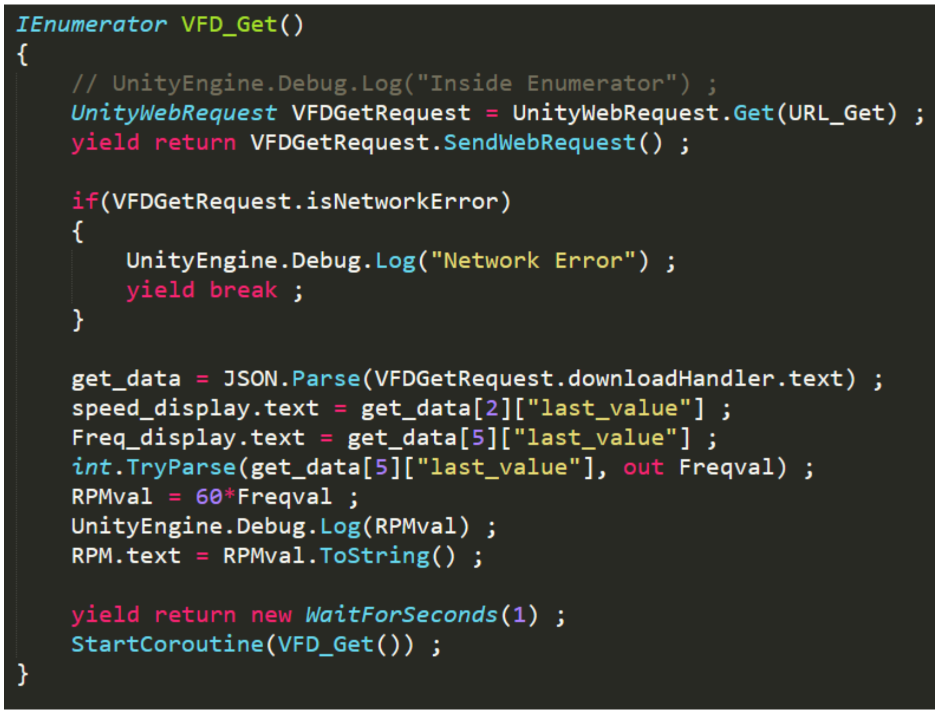
1. The setup requires data such as RPM, frequency and speed setting to be displayed to the user along with a video panel.
2. The speed setting shows the value of the slider input used to control the frequency. Three frequencies are used – 25Hz, 50Hz (Motor Rating) and 75Hz.

3. RPM displays the rpm of the motor using the formula mentioned in section 4.2.5.1 The use of an image target is similar to the one covered in section 5.1.3. This concludes the setup in the unity environment.
4. First a function is defined to read the slider value, this is the user input cue to change the speed of the VFD.
5. This value then needs to be updated in AdaFruit IO cloud. A HTTP POST request is to be send to Adafruit's API carrying the new information.
6. The connection to the API link is similar to what was discussed in section 5.1.3 and 4.1.3.1. For posting new data onto cloud, a POST request is created using the unity HTTP library function – UnityWebRequest.POST().
7. The account credentials which include the username and authentication key are passed as headers and the value to be posted is added in the body of the request.
8. The method to GET data and parse it into a C# class from the cloud for RPM, frequency etc. is similar to what was discussed in section 5.1.2.

5.2.3 Unity Code Snippets

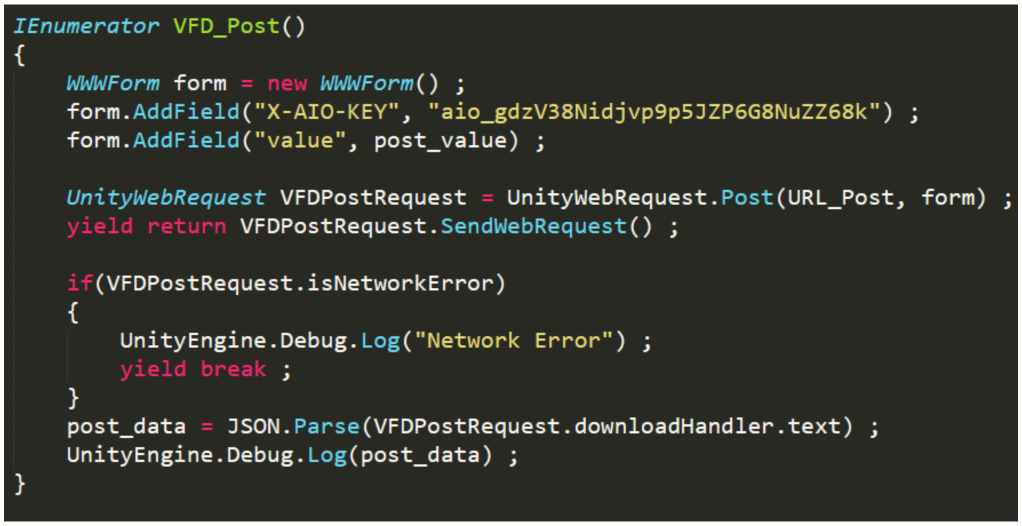


```
public class VFD_Diagnostics : MonoBehaviour
{
    public Text speed_display ;
    public Text Freq_display ;
    public Text RPM ;
    int RPMval ; int Freqval ;
    private string post_value ;
    public GameObject videoPanel ;

    string URL_Get = "https://io.adafruit.com//api/v2/tejj07/feeds" ;
    string URL_Post = "https://io.adafruit.com//api/v2/tejj07/feeds/fan-speed/data" ;
    JSONNode get_data ;
    JSONNode post_data ;

    // Start is called before the first frame update
    void Start()
    {
        StartCoroutine(VFD_Get()) ;
    }

    public void ReadSliderInput(float s)
    {
        s = s*25 ;
        post_value = s.ToString() ;
        UnityEngine.Debug.Log(post_value) ;
        StartCoroutine(VFD_Post()) ;

    }

    public void playVideo()
    {
        videoPanel.SetActive(true) ;
    }
    public void stopVideo()
    {
        videoPanel.SetActive(false) ;
```



```
IEnumerator VFD_Get()
{
    // UnityEngine.Debug.Log("Inside Enumerator") ;
    UnityWebRequest VFDGetRequest = UnityWebRequest.Get(URL_Get) ;
    yield return VFDGetRequest.SendWebRequest() ;

    if(VFDGetRequest.isNetworkError)
    {
        UnityEngine.Debug.Log("Network Error") ;
        yield break ;
    }

    get_data = JSON.Parse(VFDGetRequest.downloadHandler.text) ;
    speed_display.text = get_data[2]["last_value"] ;
    Freq_display.text = get_data[5]["last_value"] ;
    int.TryParse(get_data[5]["last_value"], out Freqval) ;
    RPMval = 60*Freqval ;
    UnityEngine.Debug.Log(RPMval) ;
    RPM.text = RPMval.ToString() ;

    yield return new WaitForSeconds(1) ;
    StartCoroutine(VFD_Get()) ;
}
```

Get request Function

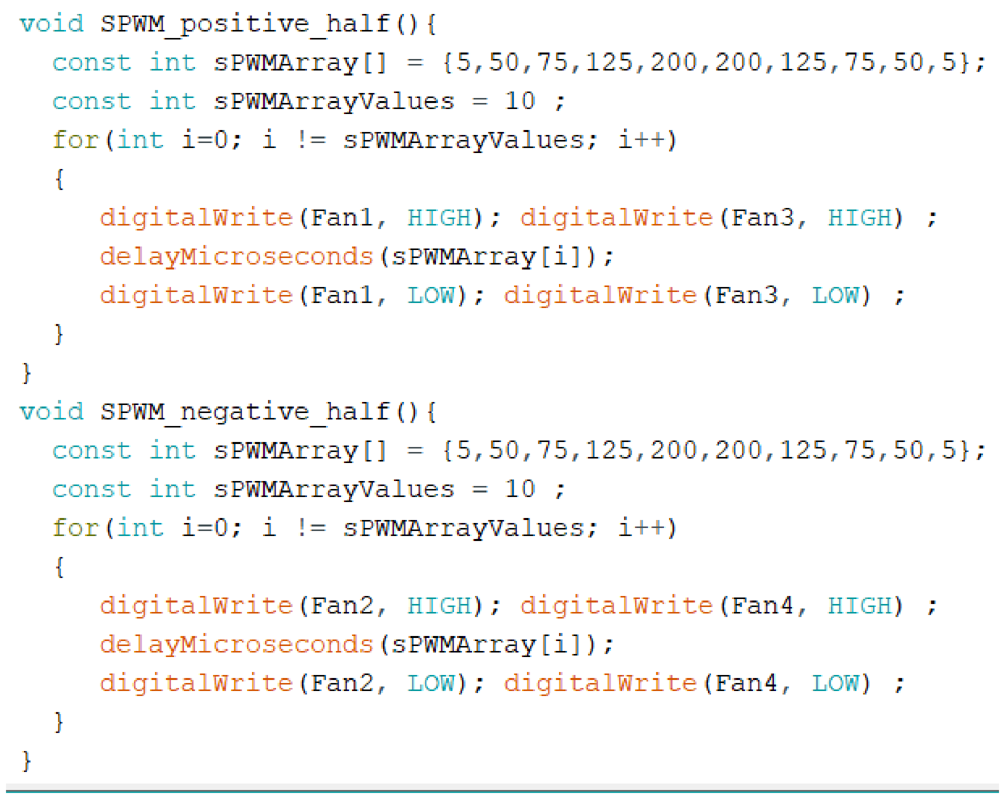


```
IEnumerator VFD_Post()
{
    WWWForm form = new WWWForm() ;
    form.AddField("X-AIO-KEY", "aio_gdzV38Nidjvp9p5JZP6G8NuZZ68k") ;
    form.AddField("value", post_value) ;

    UnityWebRequest VFDPostRequest = UnityWebRequest.Post(URL_Post, form) ;
    yield return VFDPostRequest.SendWebRequest() ;

    if(VFDPostRequest.isNetworkError)
    {
        UnityEngine.Debug.Log("Network Error") ;
        yield break ;
    }
    post_data = JSON.Parse(VFDPostRequest.downloadHandler.text) ;
    UnityEngine.Debug.Log(post_data) ;
}
```

Post request function

Here we are using another library called SimpleJSON to parse the JSON response. The native library – JSONUtility can be used to get same results.

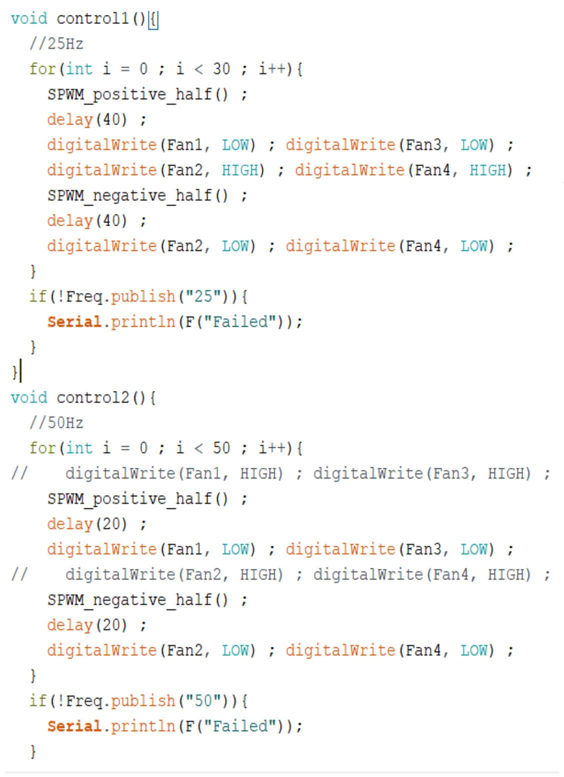
5.2.4 ESP Code Snippets



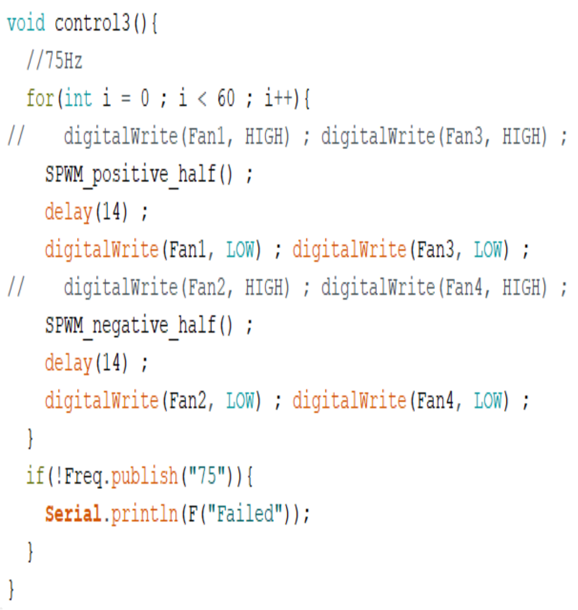
```
void SPWM_positive_half(){
  const int sPWMArray[] = {5,50,75,125,200,200,125,75,50,5};
  const int sPWMArrayValues = 10 ;
  for(int i=0; i != sPWMArrayValues; i++)
  {
     digitalWrite(Fan1, HIGH); digitalWrite(Fan3, HIGH) ;
     delayMicroseconds(sPWMArray[i]);
     digitalWrite(Fan1, LOW); digitalWrite(Fan3, LOW) ;
  }
}
void SPWM_negative_half(){
  const int sPWMArray[] = {5,50,75,125,200,200,125,75,50,5};
  const int sPWMArrayValues = 10 ;
  for(int i=0; i != sPWMArrayValues; i++)
  {
     digitalWrite(Fan2, HIGH); digitalWrite(Fan4, HIGH) ;
     delayMicroseconds(sPWMArray[i]);
     digitalWrite(Fan2, LOW); digitalWrite(Fan4, LOW) ;
  }
}
```

SPWM Generator Function

This is the SPWM generator function, an array of fixed values is first declared.
These will carry the values for the pulse width. It is noticeable that the pulse width starts off small and reaches a maxima midway and reduces again, this is to simulate a sinusoid.
Then a simple for loop is written to loop through the array and the required output pins are turned on for a delay from the SPWM array.
Similarly another for loop is done for the negative half cycle where the other two output pins are turned on and off.



```
void control1(){
  //25Hz
  for(int i = 0 ; i < 30 ; i++){
    SPWM_positive_half() ;
    delay(40) ;
    digitalWrite(Fan1, LOW) ; digitalWrite(Fan3, LOW) ;
    digitalWrite(Fan2, HIGH) ; digitalWrite(Fan4, HIGH) ;
    SPWM_negative_half() ;
    delay(40) ;
    digitalWrite(Fan2, LOW) ; digitalWrite(Fan4, LOW) ;
  }
  if(!Freq.publish("25")){
    Serial.println(F("Failed"));
  }
}
void control2(){
  //50Hz
  for(int i = 0 ; i < 50 ; i++){
//    digitalWrite(Fan1, HIGH) ; digitalWrite(Fan3, HIGH) ;
    SPWM_positive_half() ;
    delay(20) ;
    digitalWrite(Fan1, LOW) ; digitalWrite(Fan3, LOW) ;
//    digitalWrite(Fan2, HIGH) ; digitalWrite(Fan4, HIGH) ;
    SPWM_negative_half() ;
    delay(20) ;
    digitalWrite(Fan2, LOW) ; digitalWrite(Fan4, LOW) ;
  }
  if(!Freq.publish("50")){
    Serial.println(F("Failed"));
  }
```



```
void control3(){
  //75Hz
  for(int i = 0 ; i < 60 ; i++){
//    digitalWrite(Fan1, HIGH) ; digitalWrite(Fan3, HIGH) ;
    SPWM_positive_half() ;
    delay(14) ;
    digitalWrite(Fan1, LOW) ; digitalWrite(Fan3, LOW) ;
//    digitalWrite(Fan2, HIGH) ; digitalWrite(Fan4, HIGH) ;
    SPWM_negative_half() ;
    delay(14) ;
    digitalWrite(Fan2, LOW) ; digitalWrite(Fan4, LOW) ;
  }
  if(!Freq.publish("75")){
    Serial.println(F("Failed"));
  }
}
```

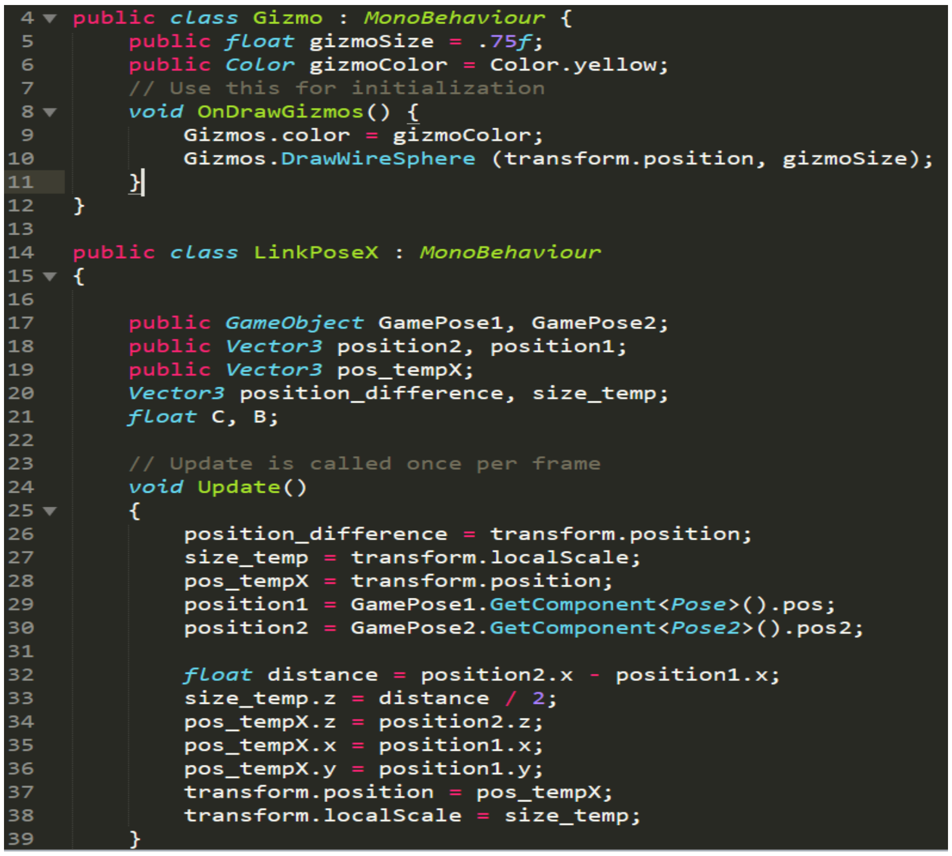
The above code snippets show the functions for the three frequency settings of 25Hz, 50Hz and 75Hz.

The frequency is decided by the delay in the switching between the two pairs of IGBTs. Therefore, a simple delay function is called between each switch.

And finally subsequently the frequency is published to the cloud to be displayed on the app.

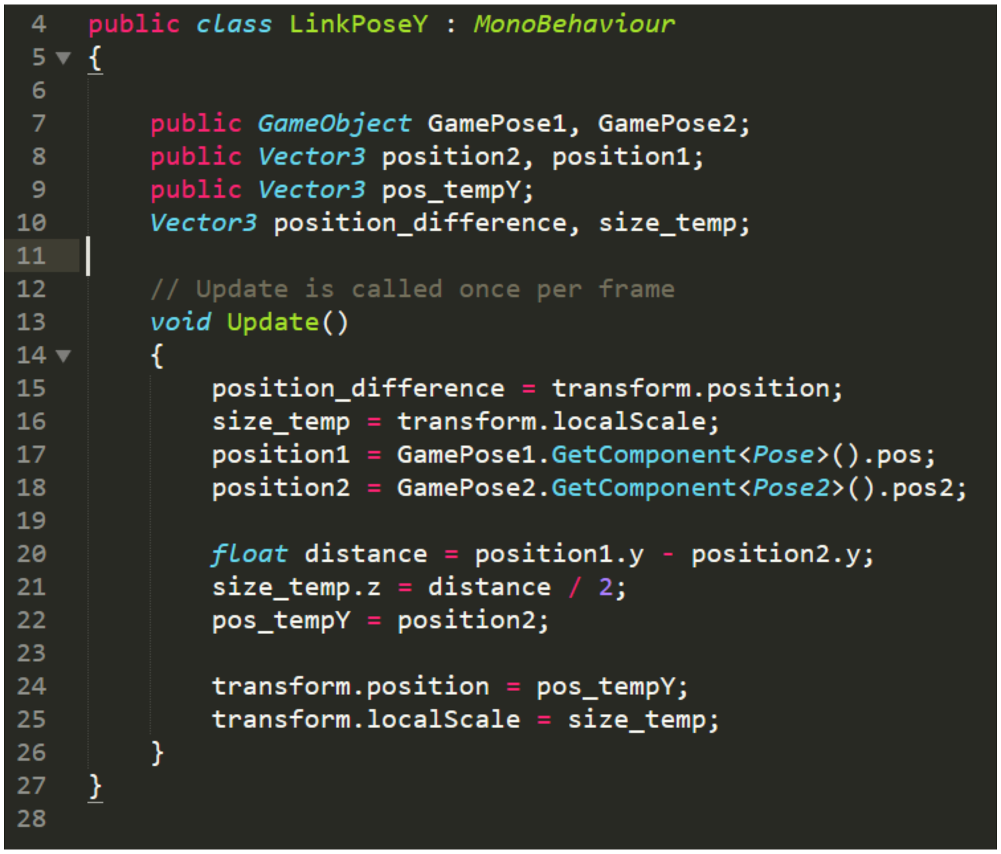
5.3 RulAR

5.3.1 Code Implementation

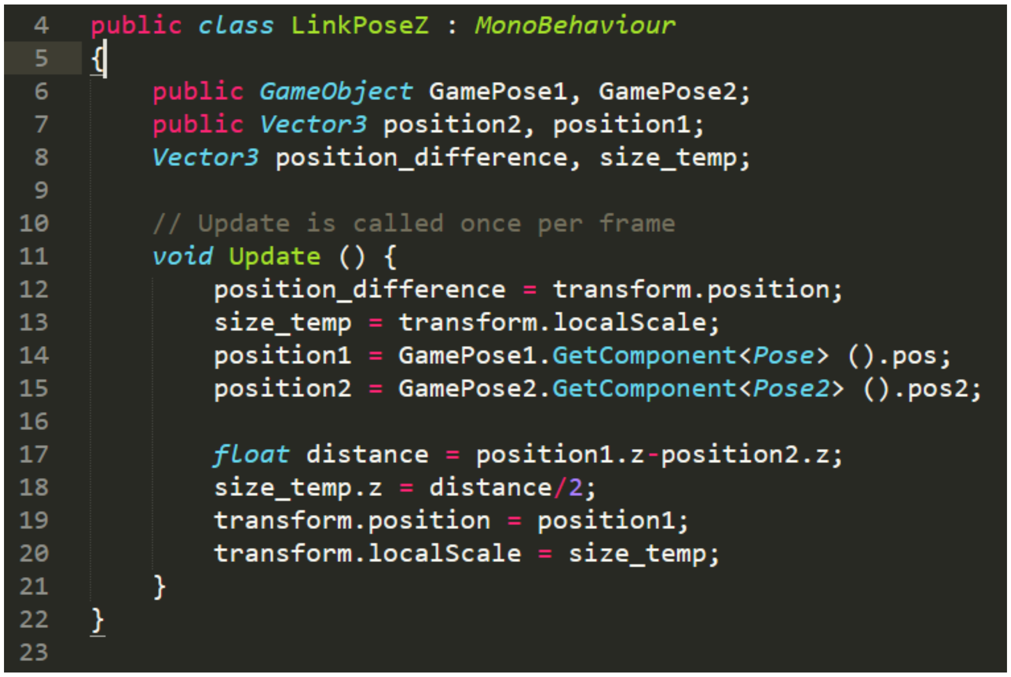


```
public class Gizmo : MonoBehaviour {
    public float gizmoSize = .75f;
    public Color gizmoColor = Color.yellow;
    // Use this for initialization
    void OnDrawGizmos() {
        Gizmos.color = gizmoColor;
        Gizmos.DrawWireSphere (transform.position, gizmoSize);
    }
}

public class LinkPoseX : MonoBehaviour
{

    public GameObject GamePose1, GamePose2;
    public Vector3 position2, position1;
    public Vector3 pos_tempX;
    Vector3 position_difference, size_temp;
    float C, B;

    // Update is called once per frame
    void Update()
    {
        position_difference = transform.position;
        size_temp = transform.localScale;
        pos_tempX = transform.position;
        position1 = GamePose1.GetComponent<Pose>().pos;
        position2 = GamePose2.GetComponent<Pose2>().pos2;

        float distance = position2.x - position1.x;
        size_temp.z = distance / 2;
        pos_tempX.z = position2.z;
        pos_tempX.x = position1.x;
        pos_tempX.y = position1.y;
        transform.position = pos_tempX;
        transform.localScale = size_temp;
    }
```

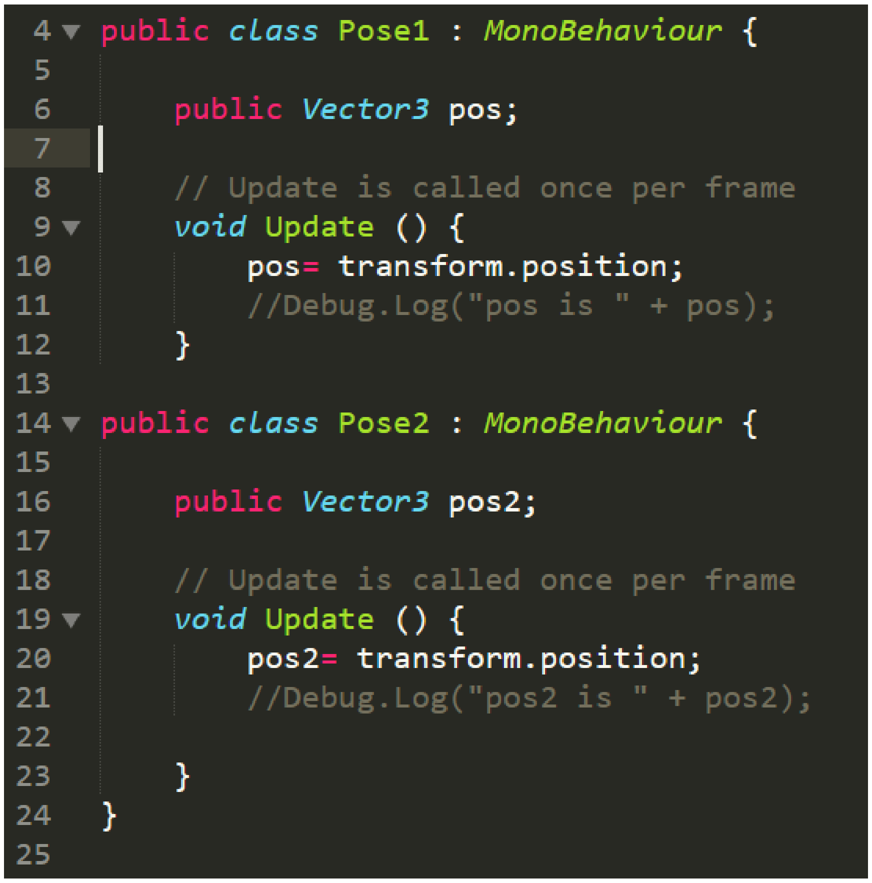
1. Firstly, a gizmo is set up for each connector object. Gizmos are used to give visual debugging or set up aids and here we are using it to ensure the connectors are oriented along the right axes.
2. GamePose1 and GamePose2 contain the cube anchors on each image target and subsequently vectors are initialized to store the position and distance vectors.
3. Vectors position1and position2 are populated with the position vectors of each cube anchor and the relative distance between them is calculated.
4. The variable pos_tempX is used to define the position vector of the connector along X-axis.
5. Similarly, a function like this is made for both the Y-axis and the Z-axis respectively as shown in the next page.



```
public class LinkPoseY : MonoBehaviour
{

    public GameObject GamePose1, GamePose2;
    public Vector3 position2, position1;
    public Vector3 pos_tempY;
    Vector3 position_difference, size_temp;

    // Update is called once per frame
    void Update()
    {
        position_difference = transform.position;
        size_temp = transform.localScale;
        position1 = GamePose1.GetComponent<Pose>().pos;
        position2 = GamePose2.GetComponent<Pose2>().pos2;

        float distance = position1.y - position2.y;
        size_temp.z = distance / 2;
        pos_tempY = position2;

        transform.position = pos_tempY;
        transform.localScale = size_temp;
    }
}

```



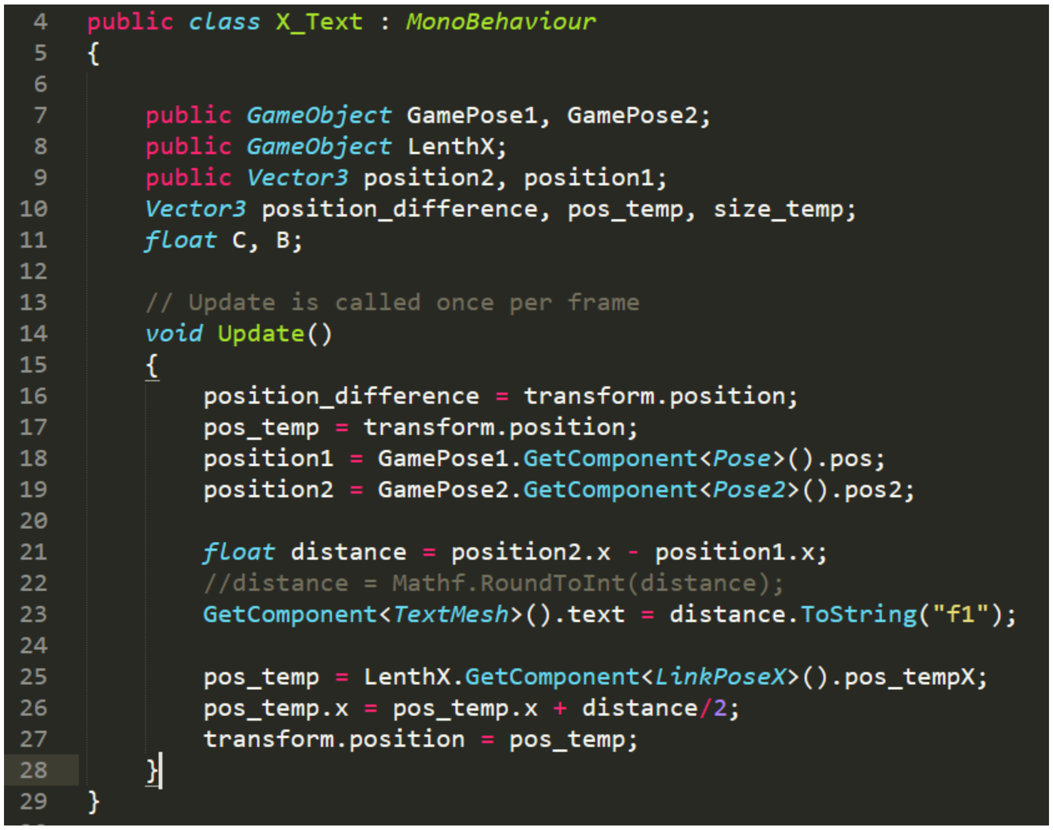
```
public class LinkPoseZ : MonoBehaviour
{
    public GameObject GamePose1, GamePose2;
    public Vector3 position2, position1;
    Vector3 position_difference, size_temp;

    // Update is called once per frame
    void Update () {
        position_difference = transform.position;
        size_temp = transform.localScale;
        position1 = GamePose1.GetComponent<Pose> ().pos;
        position2 = GamePose2.GetComponent<Pose2> ().pos2;

        float distance = position1.z-position2.z;
        size_temp.z = distance/2;
        transform.position = position1;
        transform.localScale = size_temp;
    }
}

```



```
public class Pose1 : MonoBehaviour {

    public Vector3 pos;

    // Update is called once per frame
    void Update () {
        pos= transform.position;
        //Debug.Log("pos is " + pos);
    }

public class Pose2 : MonoBehaviour {

    public Vector3 pos2;

    // Update is called once per frame
    void Update () {
        pos2= transform.position;
        //Debug.Log("pos2 is " + pos2);

    }
}

```

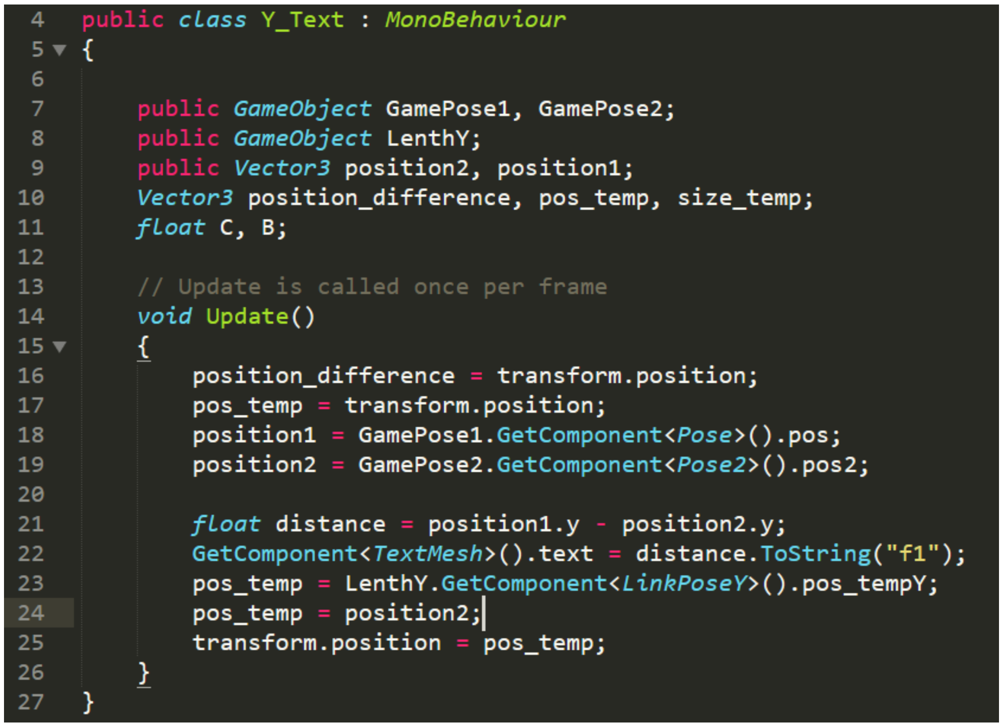
This is a simple script to be attached to the cube anchors to get their spatial position vector in every frame.

The Pose1 is attached to the first cube and Pose2 is attached to the second cube.

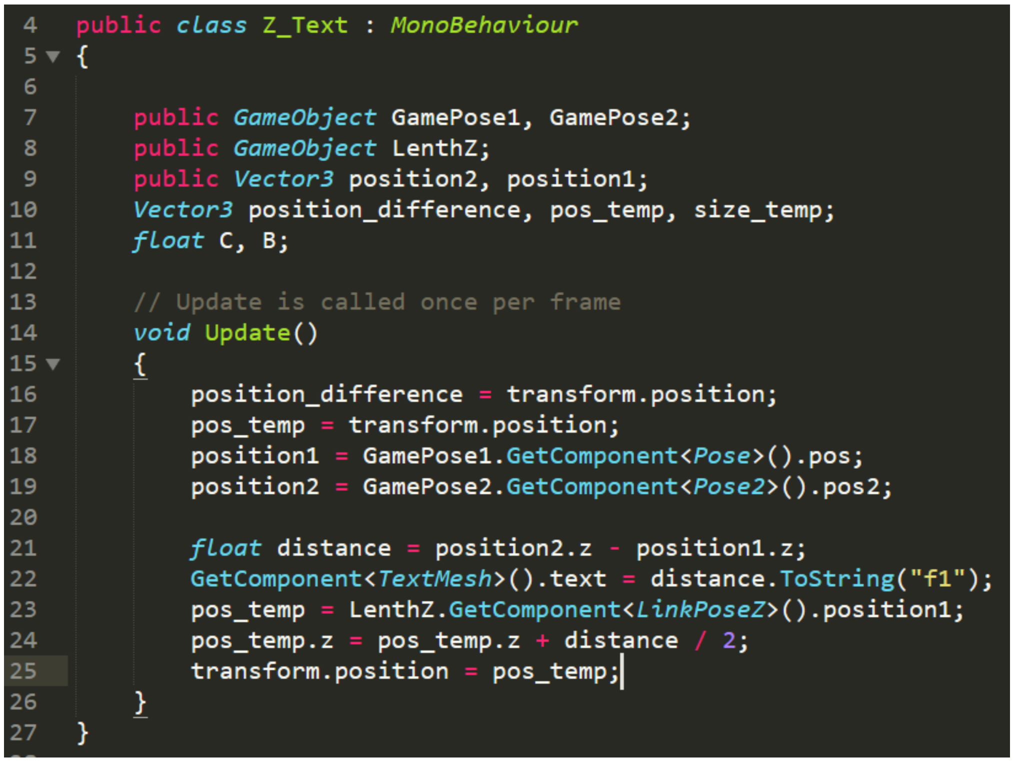


```
public class X_Text : MonoBehaviour
{

    public GameObject GamePose1, GamePose2;
    public GameObject LenthX;
    public Vector3 position2, position1;
    Vector3 position_difference, pos_temp, size_temp;
    float C, B;

    // Update is called once per frame
    void Update()
    {
        position_difference = transform.position;
        pos_temp = transform.position;
        position1 = GamePose1.GetComponent<Pose>().pos;
        position2 = GamePose2.GetComponent<Pose2>().pos2;

        float distance = position2.x - position1.x;
        //distance = Mathf.RoundToInt(distance);
        GetComponent<TextMesh>().text = distance.ToString("f1");

        pos_temp = LenthX.GetComponent<LinkPoseX>().pos_tempX;
        pos_temp.x = pos_temp.x + distance/2;
        transform.position = pos_temp;
    }
}
```

1. The above script is to dynamically change the text that displays the measured distance value along the X-axis
2. The text component must change its value according to the distance and also change its position vector to always stay at the centre of the connector objects.
3. The script uses the same logic as the LinkPose function to find the positional vectors of the cubes and we use half the measured distance for the vector of the dynamic text.
4. This function is called in every frame, hence the text will update itself to the measured distance.
5. Similar functions for the text components to display the measured distance along Y-axis and Z-axis are defined as shown in the next page.



```csharp
public class Y_Text : MonoBehaviour
{

    public GameObject GamePose1, GamePose2;
    public GameObject LenthY;
    public Vector3 position2, position1;
    Vector3 position_difference, pos_temp, size_temp;
    float C, B;

    // Update is called once per frame
    void Update()
    {
        position_difference = transform.position;
        pos_temp = transform.position;
        position1 = GamePose1.GetComponent<Pose>().pos;
        position2 = GamePose2.GetComponent<Pose2>().pos2;

        float distance = position1.y - position2.y;
        GetComponent<TextMesh>().text = distance.ToString("f1");
        pos_temp = LenthY.GetComponent<LinkPoseY>().pos_tempY;
        pos_temp = position2;
        transform.position = pos_temp;
    }
}
```



```csharp
public class Z_Text : MonoBehaviour
{

    public GameObject GamePose1, GamePose2;
    public GameObject LenthZ;
    public Vector3 position2, position1;
    Vector3 position_difference, pos_temp, size_temp;
    float C, B;

    // Update is called once per frame
    void Update()
    {
        position_difference = transform.position;
        pos_temp = transform.position;
        position1 = GamePose1.GetComponent<Pose>().pos;
        position2 = GamePose2.GetComponent<Pose2>().pos2;

        float distance = position2.z - position1.z;
        GetComponent<TextMesh>().text = distance.ToString("f1");
        pos_temp = LenthZ.GetComponent<LinkPoseZ>().position1;
        pos_temp.z = pos_temp.z + distance / 2;
        transform.position = pos_temp;
    }
}
```

CHAPTER 6 – Result and Discussion

6.1 Environment Monitoring – Results

6.1.1 Adafruit Output

The Fig 6.1.1 shows the cloud dashboard before a piece of paper was burnt near it and Fig 6.1.2 shows the same dashboard after the paper was burnt. A significant change can be noticed in the value of the MQ135 air quality sensor.

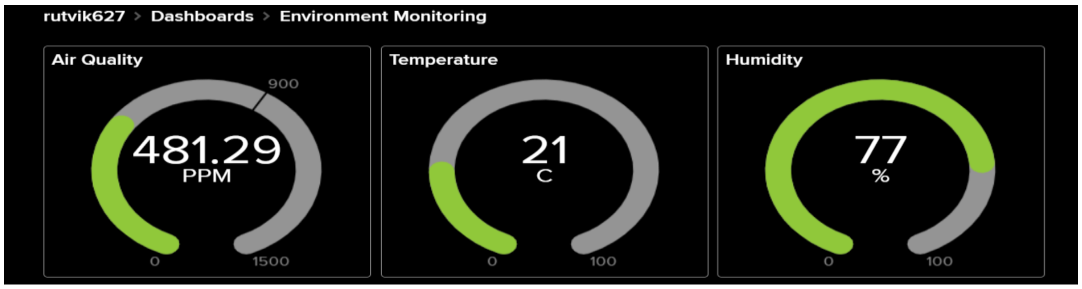


Fig 6.1.1. – Before burning paper

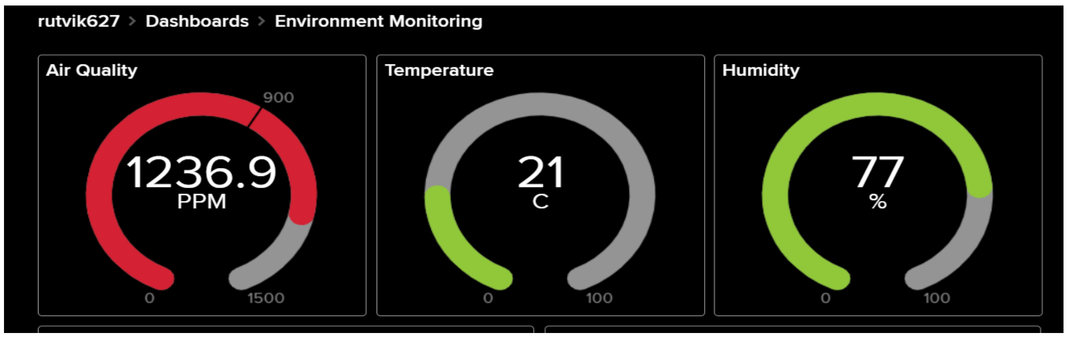


Fig 6.1.2 – After burning paper

Graphical Representation of the sensors over a period of time:

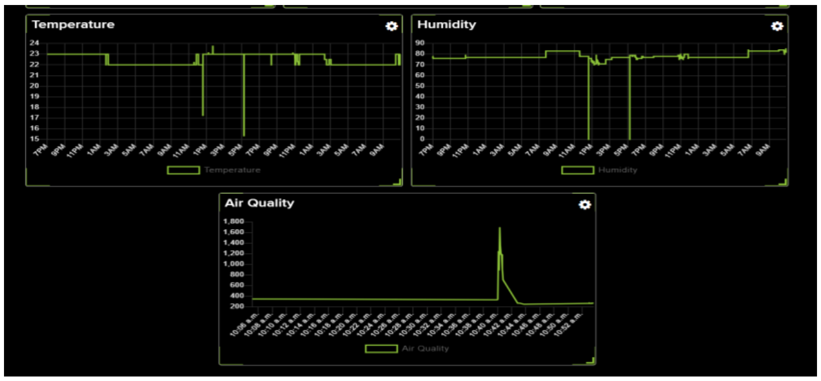


Fig 6.1.3 – Graphical data of sensor values

6.1.2 AR Application Results

Fig 6.1.4 shows the application results, upon recognition of the image target the assets are overlayed showing the sensors readings in real time. Green shows that the environment variables are in safe limits.

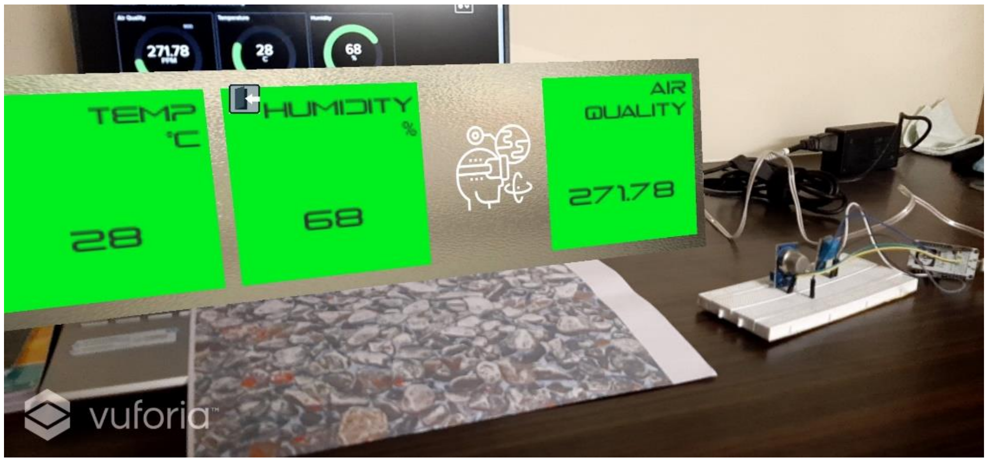


Fig 6.1.4 – AR output (Before)

Fig 6.1.5 shows the application results after a piece of paper was burnt near the sensor circuit to try and change the sensors readings substantially. The assets change color to orange indicating that air quality value is beyond safe limits.

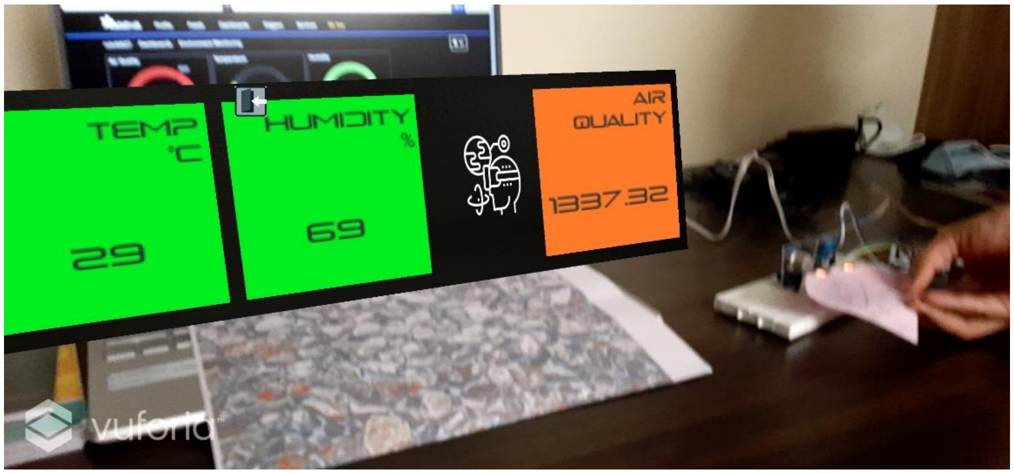


Fig 6.1.5 – AR output (After)

6.2 VFD Diagnostics Results

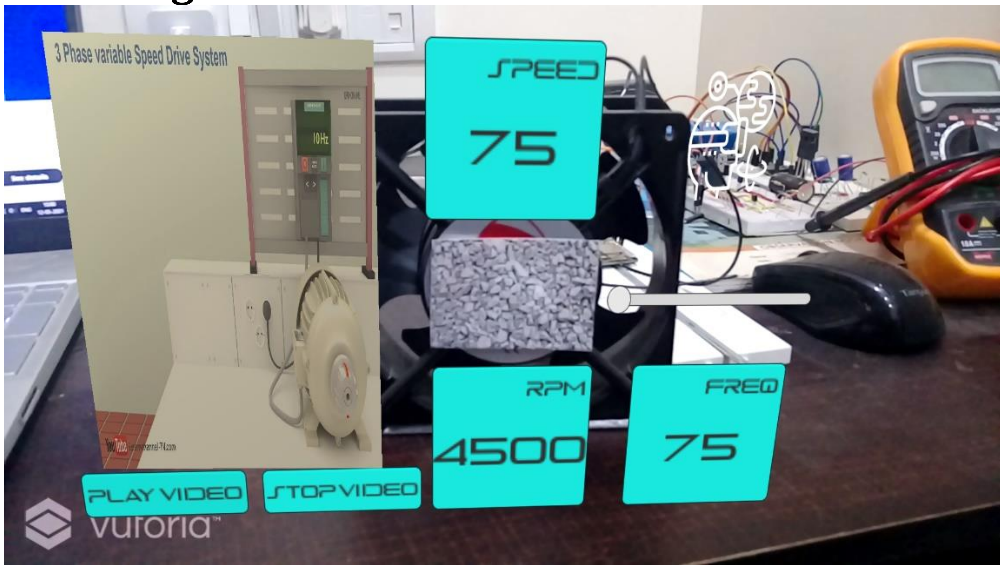


Fig 6.2.1 – VFD diagnostics use case results

Here the RPM, frequency and relative speed values are displayed through augmented digital assets.

Additionally, a video is also available which can contain useful tutorials or outline common problems and solutions pertaining to the VFD.

A slider is present for user input and this value is updated to the cloud through a POST request over HTTP and subsequently to the ESP8266 on the circuit which will change the frequency accordingly.

We are using three values of frequency for the purpose of testing, 25Hz, 50Hz and 75Hz. The fan in question is rated for 220v/ 50Hz.

It was observed that the fan rotates much slower at 25Hz and faster at 75Hz in terms of RPM.

6.3 RulAR Results

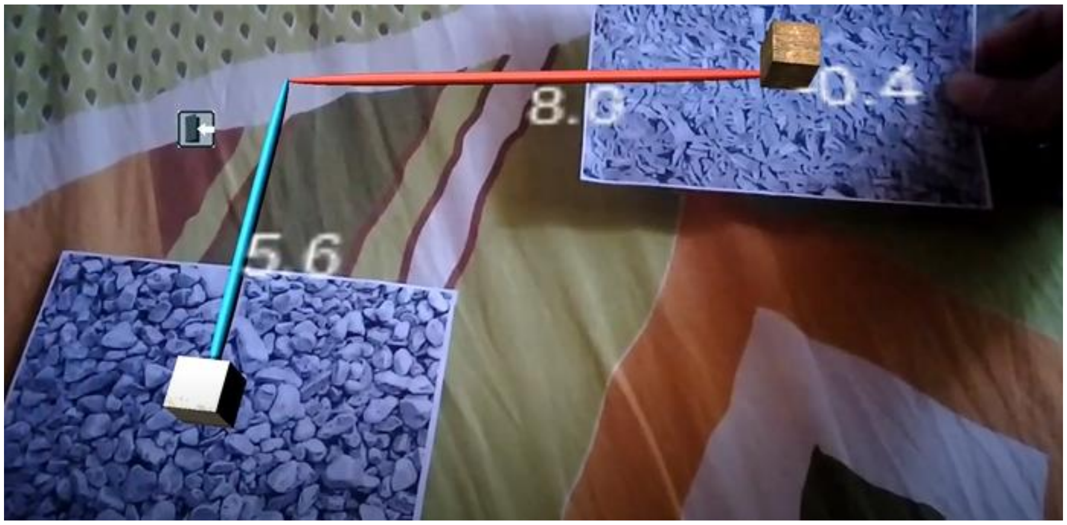


Fig 6.3.1

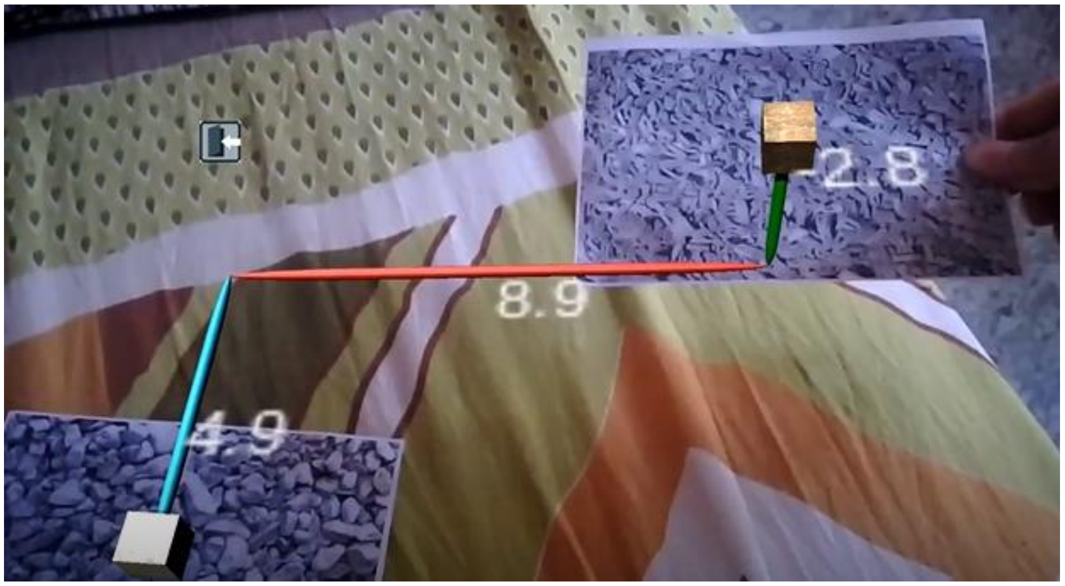


Fig 6.3.1

As Fig 6.3.1 and Fig 6.3.2 show, the blue, red and green lines make up the connector objects and show the measured distances along all three coordinate axes.

CHAPTER 7 – Conclusion And References

7.1 Conclusion

During the course of this project, we have tried to implement a warehousing environment equipped with technologies such as IoT and augmented reality.
Warehousing duties has increased manifold in every field with the coming of the consumer age and equipping them with the technology to make their processes faster and simpler is vital.
The true utility of the concepts and technologies discussed of this project is in the use of head-mounted displays in warehousing environments. A head-mounted display provides a rather lightweight and seamless yet highly intuitive experience to the user. The current application developed as part of this project can be deployed as is or with little changes to most head-mounted displays in the market. In the interest of cost we have deployed the application on a mobile device but its true utility is in head-mounted displays like the Microsoft Holo lens.

The user can then constantly monitor environment variables remotely simply to the head gear with the information being augmented right in front of the eyes. This also allows for a quicker response in the event of an emergency.

Similarly, the VFD diagnostics use case is discussed to allow for an easier process of diagnosing problems in warehouse equipment. By leveraging AR in a head-mounted display, a technician can get any required information directly in his field of view. Besides, AR can be used to map out a maintenance process using animated sequences of a virtual copy of the said equipment thus allowing for managers to be able to fix minor to moderate problems by themselves without much technical assistance.

Finally the RulAR use case was adopted again to show the extent of impact AR can have in the warehouse space by completely eliminating the need for much physical effort in measuring short distances.

7.2 References

[1] Jo D, Kim GJ. AR Enabled IoT for a Smart and Interactive Environment: A Survey and Future Directions. *Sensors*. 2019; 19(19):4330.
Link

[2] White, G., Christian Cabrera, Andrei Palade and S. Clarke. "Augmented Reality in IoT." *ICSOC Workshops* (2018).
Link

[3] Niazi, Muhammad & Hayat, Qaisar & Khan, Basit & Afaq, Muhammad. (2020). Speed Control of Three Phase Induction Motor using Variable Frequency Drive Control System. 10.14741/ijcet/v.10.1.2.
Link

[4] C. Natesan, A. Devendiran, S. Chozhavendhan, D. Thaniga and R. Revathi, "IGBT and MOSFET: A comparative study of power electronics inverter topology in distributed generation," 2015 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2015], Nagercoil, 2015, pp. 1-5, doi: 10.1109/ICCPCT.2015.7159453.
Link

[5] Thangavel, Dinesh & Ma, Xiaoping & Valera, Alvin & Tan, Hwee-Xian & Tan, Colin. (2014). Performance evaluation of MQTT and CoAP via a common middleware. 10.1109/ISSNIP.2014.6827678.
Link

[6] Meenakshi M, Monish M, Dikshit K J, Bharath S, Arduino Based Smart Fingerprint Authentication System, Hindustan Institute of Technology and Science Chennai,India : 10.1109/ICIICT1.2019.87414
Link

[7] Bikash Kumar Moharana, Prateek Anand, Sarvesh Kumar and Prakash Kodali, "Development of an IoT-based Real-Time Air Quality Monitoring Device" International Conference on Communication and Signal Processing, July 28 - 30, 2020, India, doi:10.1109/ICCSP48568.2020.9182330
Link

[8] Mayuresh Kharade , Shubham Katangle , Ganesh M. Kale , S. B. Deosarkar , S. L. Nalbalwar "A NodeMCU based Fire Safety and Air Quality Monitoring Device" 2020 International Conference for Emerging Technology (INCET) Belgaum, India. Jun 5-7, 2020
Link

[9] Marie-Hélène Stoltz, Vaggelis Giannikas, Duncan McFarlane, James Strachan, Jumyung Um, Rengarajan Srinivasan,
Augmented Reality in Warehouse Operations: Opportunities and Barriers, IFAC-PapersOnLine,Volume 50, Issue 1,2017,ISSN 2405-8963, https://doi.org/10.1016/j.ifacol.2017.08.1807.
Link

[10] F. E. Jamiy and R. Marsh, "Distance Estimation In Virtual Reality And Augmented Reality: A Survey," 2019 IEEE International Conference on Electro Information Technology (EIT), 2019, pp. 063-068, doi: 10.1109/EIT.2019.8834182.
Link

[11] Mourtzis, Dimitris & Samothrakis, Vasilios & Zogopoulos, Vasilios &Vlachou, Katerina. (2019). Warehouse Design and Operation using Augmented Reality technology: A Papermaking Industry Case Study. Procedia CIRP. 79. 574-579. 10.1016/j.procir.2019.02.097.
link