AerialAV: Aerial Analysis and Visualisation

David Adeshina Arungbemi

Nile University of Nigeria

Department of Computer Engineering

Abstract

AerialAV integrates IoT principles into aerospace technology, enabling real-time data collection from UAVs and visualisation of motion and environmental conditions. Utilising MQTT messaging and a variety of sensors, including GPS and IMU, AerialAV empowers users with insights into UAV dynamics and surroundings. Despite challenges, the project delivers a responsive flight application. Future improvements focus on sensor reliability and expanded capabilities. AerialAV represents a notable advancement in aerospace technology with applications in drone flying, surveillance, and environmental monitoring.

Index Terms

Internet of Things (IoT), Unmanned Aerial Vehicles (UAVs), Real-time data collection Environmental monitoring, MQTT messaging protocol, Sensor integration, GPS, IMU (Inertial Measurement Unit), Sensor fusion.

I. INTRODUCTION

The AerialAV project represents an effort to integrate IoT principles into the aerospace field focusing on collecting real-time data remotely from UAV(Unmanned Aerial Vehicles) and providing live visualisation of motion and environmental condition. AerialAV aims to empower users with insights into their drones dynamics and environmental conditions. The project combines both hardware system with multiple sensors and a software application. In the following subsections we discuss the background; methodology including circuit design, implementation, messaging protocol and software application design; the performance of the system in terms of responsiveness and reliability; challenges encountered and solutions and further developments to be implemented.

II. BACKGROUND THEORY

A. Unmanned Aerial Vehicles and Internet of Things

UAVs, most commonly referred to as drones, can be simplify defined as aircrafts without a pilot, crew or any passenger on board. They operate similarly to our commercial or military aircraft but without the human element, space for crew, comfort and human elements are no longer required. UAVs are applied in general use including recreation, film making, surveillance of cities, ecosystems and most commonly, the military industry for reconnaissance and to destroy enemy infrastructure or soldiers. Their use greatly affects their physical design and operation as shown in Figure 1. Figure 1a operates similarly to a helicopter and is best suitable for recreation and photography. Figure 1b however, operates similarly to airplanes, with a focus on speed to reach its

target and destroy. They have different levels of autonomy from self stabilisation to preprogrammed flight modes. They achieve this using a variety of sensors and a computer to guide the aircraft and make necessary flight adjustments [1].

Internet of Things involves devices utilising sensors, computing hardware, networks and other technologies to connect with and share data with other devices over the internet or some network [2]. For example, IoT is applied in home automation. Using an app, voice or some other input form, one can control lighting, air conditioners and other smart appliances. Commands can be sent over a dedicated LAN network to each smart device. We also see IoT applied in medical devices such as pacemakers. Medical professionals can remotely monitor patient heart rate and rhythm and quickly respond if there is an emergency. In both examples, sensors, networks and some form of visualisation or interface is required. Additionally, some form of processing is required.

The fusion of IoT with UAVs enables them to collect, process and transmit data over the internet. This data can be sent to a central hub such as cloud for processing and analysis.

Fig. 1 - (a) A DJI Phantom quadcopter UAV for commercial and recreational aerial photography (b) A General Atomics MQ-9 Reaper, a hunter-killer surveillance UAV
Source: Adapted from [1]

B. Sensors, Data Collection and Analysis

Sensors are devices that produce an output signal in response to some external or changes in environmental factor. Sensor are utilised in everyday life, in our phones when we interact with the screen, keyboard actions on laptops/desktops, GPS (Global Positioning System) sensors to track locations on our phones and more.

In the fusion of aerospace and IOT, the following sensors can be used:

i. GPS Sensors: provides drone location based on longitude, latitude and altitude.

ii. IMU (Inertial Measurement Unit): consisting of gyroscope and accelerometer that measure angular velocity and linear acceleration respectively.

iii. Barometer: measures atmospheric pressure.

iv. Magnetometer: which detects earths magnetic field.

v. Cameras: to take images or videos of environment.

vi. Humidity and temperature sensors: measures relative humidity and temperature of location.

Using data collected from these sensors, one can generate new information from a single sensor or combine data from different sensors to derive more information (sensor fusion). For example, the barometer measures atmospheric pressure which changes with altitude. Hence, from the pressure readings we may also estimate altitude. Additionally, we may utilise sensor fusion of gyroscope and accelerometer(with magnetometer) to aircraft orientation, estimating the roll, yaw and pitch of a UAV.

C. Messaging Protocols in IOT and MQTT

Communication in IoT is wireless over some network, usually the internet. To facilitate data exchange and coordination between devices, messaging protocols are required. Commonly used protocols such MQTT (Message Queuing Telemetry Transport), AMQP (Advanced Message Queuing Protocol, CoAP (Constrained Application Protocol) [3].

The AerialAV project utilises the MQTT protocol, a lightweight, open-source messaging protocol widely used in IoT (Internet of Things) and M2M (Machine-to-Machine) communication. The features of MQTT are:

i. Publish-Subscribe Architecture: MQTT follows a publish-subscribe messaging model. There are three main entities: publishers, subscribers, and a broker. Publishers are devices or applications that generate data or events. Subscribers are devices or applications that receive or act upon published data. The broker is a server responsible for receiving all messages from publishers and routing them to the appropriate subscribers [4]. The relationship between these 3 entities is illustrated in Figure 2.

ii. Topic-Based Messaging: Messages in MQTT are published to a specific topic which are hierarchical in nature and use a forward-slash (/) as a delimiter [4]. Subscribers can subscribe to specific topics or topic patterns using wildcards [5]. This allows for flexible control over which messages a subscriber receives.

iii. Quality of Service (QoS) Levels: MQTT supports three levels of QoS, which dictate the delivery guarantees for messages. In QoS 0 (At most once), the message is delivered once or not at all. This level offers the lowest assurance but is the most efficient. QoS 1 (At least once), guarantees the message to be delivered at least once, but duplicates may occur. QoS 2 (Exactly once), the message is guaranteed to be delivered exactly once using a two-level handshake. This ensures message integrity but incurs higher overhead [4].

iv. Lightweight and Efficient: MQTT is designed to be lightweight and efficient, making it suitable for use in resource-constrained devices and low-bandwidth networks. It uses TCP/IP or other transport protocols for communication, allowing it to operate over various network types. Additionally, its protocol header is minimal, reducing overhead [4].

v. Persistent Sessions and Retained Messages: MQTT supports persistent sessions, allowing clients to resume communications after disconnection without losing message ordering. It also

supports retained messages, where the broker stores the last message published to a topic and delivers it to new subscribers upon subscription [4].

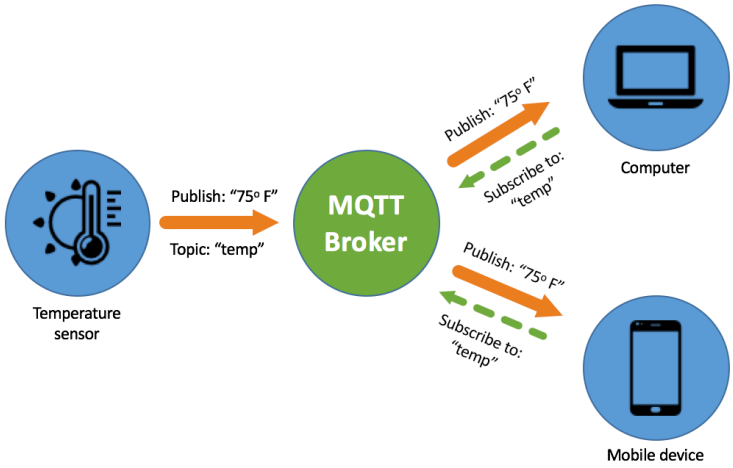

Fig. 2 - MQTT in Temperature Sensing

Source: Adapted from [6]

D. Application Software and Data Visualisation

Data visualisation are necessary to visualise relationships in data. Dashboards, maps and charts such as line graph and bar chart are used to present IoT data in a human-readable format. They help users to understand trends, patterns and anomalies in data.

Applications provide a platform to view these data visualisation and understand the trends and to potentially control actuators if needed. Developing such software requires consideration of factors like resource constraints (memory, processing limits), real-time requirements and security considerations.

III. METHODOLOGY

A. Circuit Components

The AerialAV device consists of the following:

i. ESP-WROOM-32 Microcontroller: Supplies each sensor with 3.3 V and communicates with them either by I2C (Inter-Integrated Circuit) or a single-wire digital protocol. Additionally, it performs sensor fusion using data from IMU sensor to derive orientation.

ii. MPU6050: A microelectromechanical systems (MEMS) IMU device that combines a 3-axis gyroscope and a 3-axis accelerometer in a single package. Communicates with the ESP32 using I2C protocol.

iii. BMP180: A digital barometric pressure sensor, designed to measure atmospheric pressure and temperature with high accuracy. It has a resolution of 0.01 hPa (hectopascals) for pressure and 0.1°C for temperature, allowing for precise environmental monitoring. Communicates via the I2C protocol.

iv. DHT11: A low-cost digital temperature and humidity sensor.Communicates with ESP32 using the single-wire digital protocol. The temperature accuracy is typically within ±2°C. and humidity accuracy within ±5% relative humidity (RH).

v. LM393 Sound Sensor: Designed to detect and measure the intensity of sound waves and convert them to electrical signals. Utilises the microphone as its primary components for picking sound with sensitivity of (1kHz): 52 to 48 dB. It provides an analog output which corresponds to the magnitude of the sound.

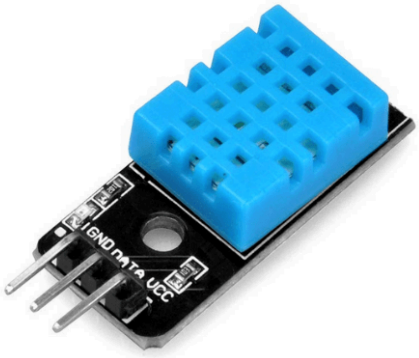

Fig. 3 - DHT11 Sensor
Source: Adapted from [7]

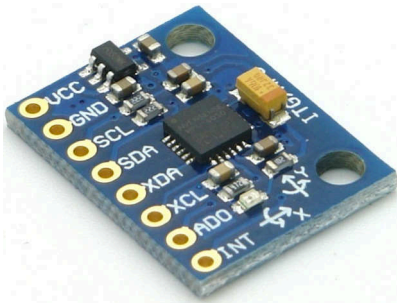

Fig. 4 - MPU6050
Source: Adapted from [8]

Fig. 5 - BMP180
Source: Adapted from [9]

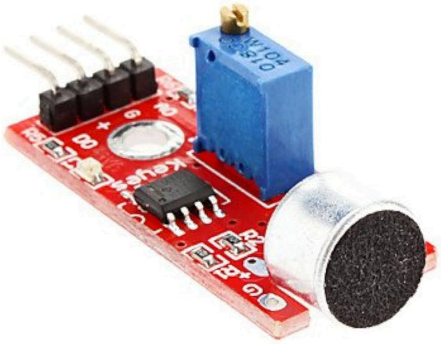

Fig. 6 - LM393 Sound Sensor
Source: Adapted from [10]

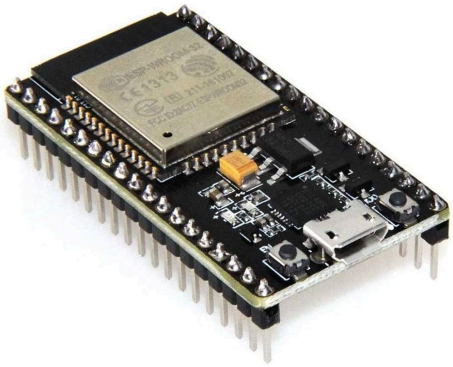

Fig. 7 - ESP-WROOM-32
Source: Adapted from [11]

B. Circuit Schematics

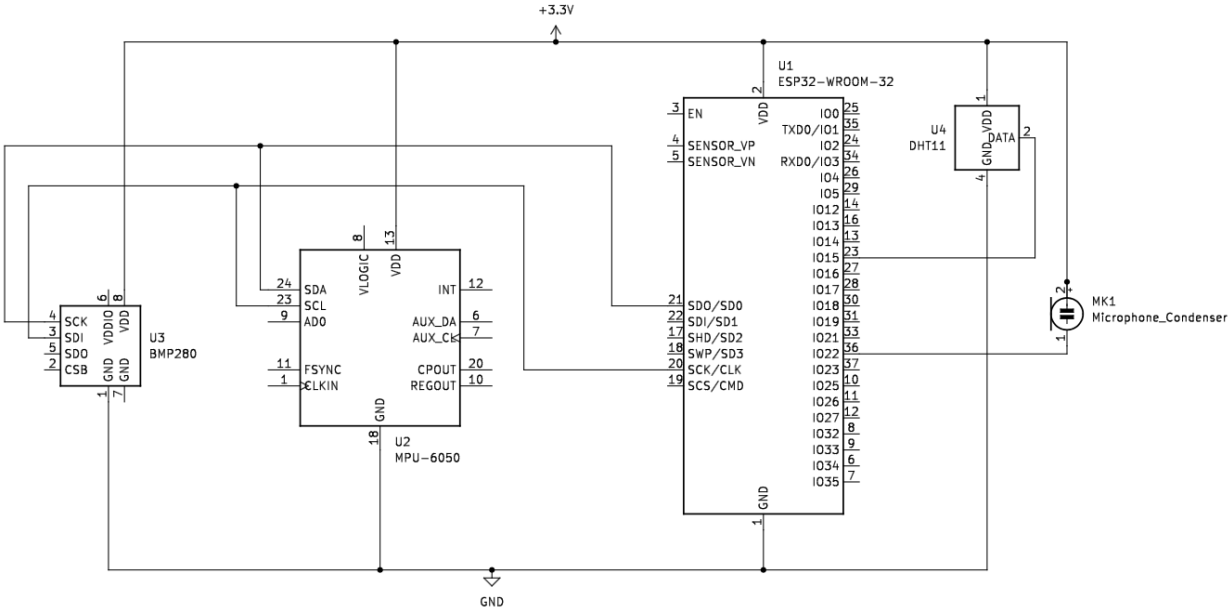

Fig. 8 - Schematic Diagram (AerialAV Hardware)

Note: While designed in KiCad, actual ESP-32 SCL and SDA pin assignments differ from schematic due to variant ESP-32 models, emphasising the need for meticulous hardware validation. KiCad did not have the LM393 sound sensor, hence a condenser microphone, a main component of the LM393 was used instead. See Appendix C for breadboard implementation.

C. Firmware Development

The ESP-WROOM-32 microcontroller performs the following:

i. Power the Sensors (which is done via the 3.3 V pin).

ii. Communicate with devices via I2C and single-wire digital protocol and read data.

iii. Perform sensor fusion of gyroscope and accelerometer readings to derive orientation.

iv. Establish connection with online MQTT broker and publish data to Topic.

The microcontroller is programmed using C++ in the Arduino IDE. The BMP180 and MPU6050 as mentioned communicate with the microcontroller using the I2C protocol. Both sensors have their SCL and SDA pins connected to pin 22 and 21 respectively. By default, ‘0x68’ and ‘0x77’ address is assigned to the MPU6050 and BMP180 sensors respectively.

The microcontroller communicates with the MPU6050 and read its raw data, using the MPU6050 and Wire library [12]. The raw data is processed by dividing raw data for accelerometer and gyroscope by their respective sensitivities (16384 LSB/g and 131 $LSB/°/s$), which gives us values in the range of ±2g for accelerometer and ±250°/s for gyroscope.

Orientation readings are derived using the xioTechnologies' Fusion library [13]. The library implements the Madgwick filter for sensor fusion. The results from the calculations are Euler roll, pitch and yaw angles to describe orientation.

Th BMP180 utilises the Adafruit BMP085 library to get pressure and altitude readings. The library performs all necessary calculations and provides the pressure in pascals and altitude in metres.

Both DHT11 and LM393 sound sensor communicate with the microcontroller using single-wire digital protocol. It makes use of the Adafruit Unified Sensor library. It communicates with the DHT11 sensor through pin 23 and reads temperature (in Celsius) and relative humidity (%).

The LM393 sound sensor requires no library, simply setting the pin 36 of the ESP-32 as input is enough to collect vibration readings.

The microcontroller utilises the its WiFi library for connecting to WiFI networks. In this project, it connects to a nearby WiFi for internet connection. The PubSubClient library is an MQTT client library for connecting to an MQTT broker, publishing and subscribing to topics. The library is utilised to publish data from the sensors. The project makes use of the HiveMQ public broker [14] and publishes to the topic "AerialAV".

The PubSubClient library utilises the 1883 port for communication with the broker, however it is not encrypted. The quality of service selected in QoS 0. While quality of service 0 does not ensure that subscriber receives the data. It avoids duplicates and overheads, which would lead to inefficient and slow response to changing events of the UAV.

The data format chosen was the JSON format due to its minimal overhead, ease of parsing and its compatibility. The ArduinoJSON library [15] was utilised to format data to be sent and serialise to string.

Publishing occurs in two stages. The first stage publishes the orientation, gyroscope and accelerometer readings. The second stage deals with publishing readings from the DHT11, LM393 and BMP180 sensor. This was done to due to the message size constraint from the PubSubClient library (maximum of 128 bytes).

See Appendix I for snippet of firmware code.

D. Software Application Development

The flight software for remote computers is built with Processing, an IDE and library that allows developers to creating interactive applications in JAVA [16]. The flight software application utilises the following dependencies: gicentreUtils library for charts [17], PeasyCam library for camera control with mouse [18], MQTT library [19], and a free 3D Model of a Drone [20].

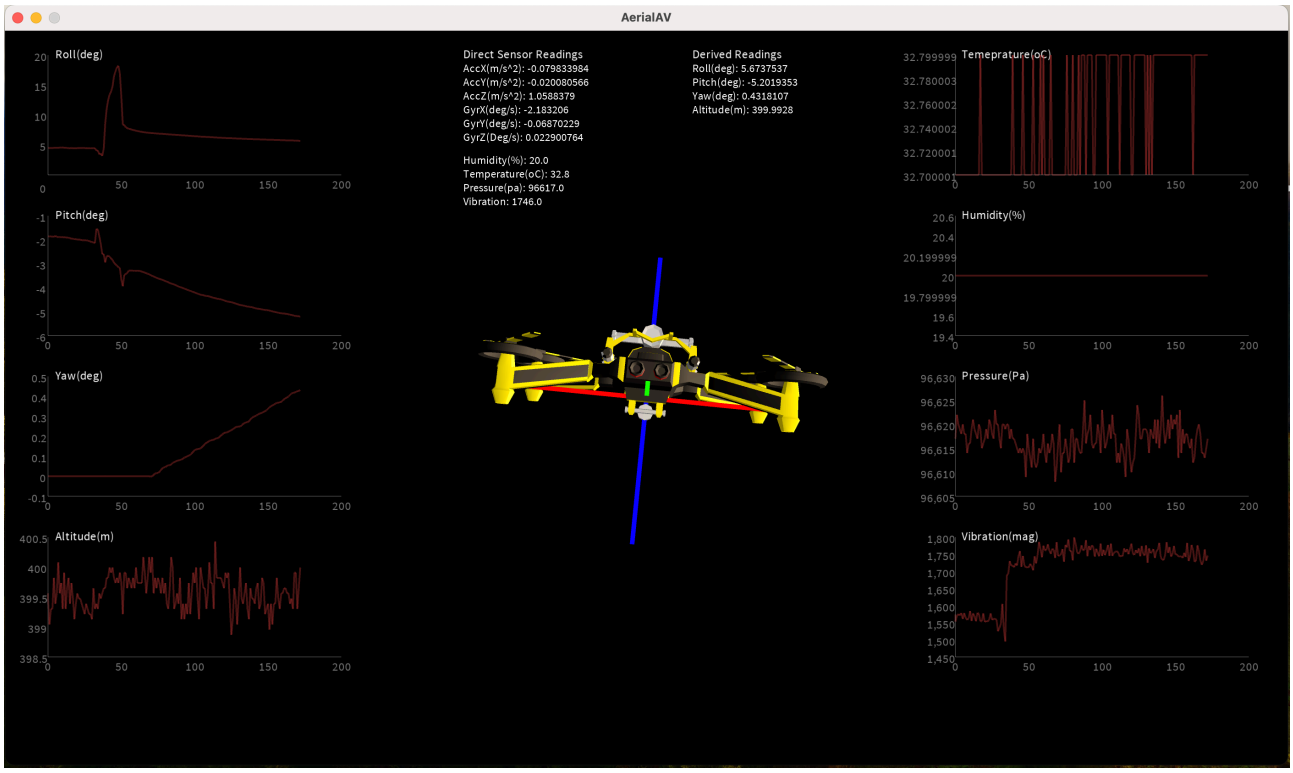

Fig. 9 - AerialAV Flight Software

Figure 9 above is the user interface of the AerialAV flight software. The displays motion data including orientation (roll, pitch and yaw) and altitude of the UAV; environmental data including temeperature, humidity, pressure and vibration from the surrounding. The software provides direct readings in text format such as acceleration, angular velocity, humidity, temperature, pressure and vibration; and derived orientation and altitude readings.

To better visualise the orientation, the software includes a 3D model of a drone whose orientation changes according to the UAV's orientation. The blue, red and green lines represent the z, x and y axis to give an idea of direction.

See Appendix II for snippet of flight software code.

IV. DISCUSSION

A. Challenges and Solutions

During development of this project, certain obstacles were encountered. However, at the end of the project. It can be confidently said that it was a success.

i. Error in DHT11 sensor readings: Sometimes the humidity and temperature readings of the DHT11 sensor display zeros or simply nulls as output. Different libraries were tested on the sensor, however the same issue persisted. With an Arduino ATMega present at the time, I tested the DHT11 sensor and the problems did not occur. With further research, it seems some DHT11 sensors do not work well at 3.3 V. The DHT11 sensor was unreliable as the data sheet provided a voltage range of 3.3-5.5 V DC.

ii. Sensor drift: Orientation readings drift even without any movement. This is caused by the inaccuracies of MPU6050 sensor which accumulates over time. Usually, the solution is a sensor fusion which helps to provide new readings that are less prone to error. However, the yaw readings do suffer drift the most. This problem is usually solved with a magnetometer, however the magnetometer purchased from vendor was faulty and was not get refund. The issue with yaw drifting was left unsolved.

iii. Unresponsive 3D model in flight software: Initially, the responsiveness of the 3D model was slow. The language used, internet connection and more were considered as possible causes. However, on observing the orientation readings direct in the serial terminal, the readings were also slow, which means the issue from code related. It was discovered that by reducing the sampling rate initially set at 1000 Hz to 100 Hz, the performance greatly increased but at the cost of less sensitivity, which was acceptable.

B. Responsiveness and Reliability

In general, the flight application is highly responsive to sensor changes as long as internet connection is stable. Poor internet connection may impede responsiveness. The readings from the sensors are reliable to an acceptable degree, however over time its reliability will go down as the orientation readings continue to drift. Hence, its only reliable for a short while.

C. Future Developments

In future developments, changes to sensors will include a much reliable temperature and humidity sensor that can work with 3.3 V, an IMU sensor that collects not just accelerations and angular velocity but also magnetic field for more accurate sensor fusion(9-DOF IMU). A GPS sensor could also be added to track position.

V. CONCLUSION

The AerialAV project merges IoT principles with aerospace technology, enabling real-time data collection from UAVs and live visualisation of motion and environmental conditions. By integrating hardware components and a sophisticated software application, AerialAV provides users with valuable insights into drone dynamics and surrounding environmental factors.

Through the fusion of IoT with UAVs, AerialAV facilitates data collection, processing, and transmission over the internet, offering enhanced monitoring and analysis capabilities. The project utilises MQTT messaging for efficient communication, while its hardware setup includes various sensors like IMU, barometer, DHT11 for comprehensive data collection.

Despite challenges encountered during development, such as sensor reading errors and drift, the project successfully delivers a responsive and reliable flight application. Future enhancements aim to address sensor reliability issues and expand capabilities, including the integration of more reliable sensors and GPS for position tracking.

In conclusion, AerialAV represents a promising advancement in aerospace technology, offering practical applications in recreational drone flying, aerial surveillance, and environmental monitoring with its innovative IoT integration.

VI. REFERENCE

[1] Wikipedia Contributors, “Unmanned aerial vehicle,” Wikipedia, Feb. 09, 2024. Available: https://en.wikipedia.org/wiki/Unmanned_aerial_vehicle#Aircraft_configuration. [Accessed: Feb. 19, 2024]

[2] Wikipedia Contributors, “Internet of things,” Wikipedia, Mar. 16, 2019. Available: https://en.wikipedia.org/wiki/Internet_of_Things. [Accessed: Feb. 19, 2024]

[3] A. Bhatt, “A Quick Guide to Understanding IoT Application Messaging Protocols,” eInfochips, Jun. 27, 2018. Available: https://www.einfochips.com/blog/a-quick-guide-to-understanding-iot-application-messaging-protocols/. [Accessed: Feb. 19, 2024]

[4] Wikipedia Contributors, “MQTT,” Wikipedia, Dec. 28, 2019. Available: https://en.wikipedia.org/wiki/MQTT

[5] G. Hillar, MQTT Essentials - A Lightweight IoT Protocol. Packt, 2017.

[6] N. Sharma, “Getting Started with MQTT — Part 1,” Medium, Apr. 29, 2020. Available: https://nitin-sharma.medium.com/getting-started-with-mqtt-part-1-a3c365e3a488

[7] “DHT11 Temperature and Humidity Sensor Module,” Cretechs®. Available: https://shop.cretechs.in/product/dht11-temperature-and-humidity-sensor-module/. [Accessed: Feb. 19, 2024]

[8] “MPU6050 (Gyroscope + Accelerometer + Temperature) Sensor Module |..,” Electronicwings.com, 2020. Available: https://www.electronicwings.com/sensors-modules/mpu6050-gyroscope-accelerometer-temperature-sensor-module

[9] “BMP180 Digital Barometric Sensor Module compatible with Arduino,” Robu.in. Available: https://robu.in/product/bmp180-digital-barometric-pressure-sensor-board-module-arduino-compatible-2/

[10] “Adraxx Sound Sensor Module Micro For Arduino, ARM And Other MCU : Amazon.in: Industrial & Scientific,” www.amazon.in. Available: https://www.amazon.in/Adraxx-Sound-Sensor-Module-Arduino/dp/B071CGJNK2. [Accessed: Feb. 19, 2024]

[11] “IoT-based smart Irrigation Control System using LoRa communication,” ResearchGate. Available: https://www.researchgate.net/figure/HiLetgo-ESP-WROOM-32-ESP32_fig3_343079732

[12] J. Rowberg, “i2cdevlib/ESP32_ESP-IDF/components at master · jrowberg/i2cdevlib,” GitHub. Available: https://github.com/jrowberg/i2cdevlib/tree/master/ESP32_ESP-IDF/components. [Accessed: Feb. 19, 2024]

[13] x-io Technologies, “xioTechnologies/Fusion,” GitHub, Feb. 19, 2024. Available: https://github.com/xioTechnologies/Fusion. [Accessed: Feb. 19, 2024]

[14] “The Free Public MQTT Broker & MQTT Client by HiveMQ - Check out our MQTT Demo,” HiveMQ. Available: https://www.hivemq.com/mqtt/public-mqtt-broker/

[15] Benoit Blanchon, “ArduinoJson: Efficient JSON serialization for embedded C++,” ArduinoJson. Available: https://arduinojson.org/

[16] Processing, “Processing.org,” Processing.org, 2019. Available: https://processing.org/

[17] giCentre, “giCentre utilities,” giCentre. Available: https://www.gicentre.net/utils. [Accessed: Feb. 19, 2024]

[18] J. Feinberg, “jdf/peasycam,” GitHub, Jan. 16, 2024. Available: https://github.com/jdf/peasycam. [Accessed: Feb. 19, 2024]

[19] J. Gähwiler, “256dpi/processing-mqtt,” GitHub, Jan. 23, 2024. Available: https://github.com/256dpi/processing-mqtt. [Accessed: Feb. 19, 2024]

[20] Free3D, “Drone Costume Free 3D Model - .blend .fbx .obj - Free3D,” free3d.com. Available: https://free3d.com/3d-model/drone-costume-411845.html. [Accessed: Feb. 19, 2024]

APPENDIX A

```
#include <DHT.h>
#include <DHT_U.h>
#include <Adafruit_Sensor.h>
#include <ArduinoJson.h>
#include <PubSubClient.h>
#include <Adafruit_BMP085.h>
#include "Orientation.h"
#include <WiFi.h>
#define SOUND_SEN_PIN 36
#define DHTPIN 23
#define DHTTYPE DHT11

MPU mpu;
DHT_Unified  dht(DHTPIN, DHTTYPE); //temperature and humidity sensor
Adafruit_BMP085 bmp180;
const char* ssid = “WIFI NAME”;
const char* password = "PASSWORD";
const char* mqtt_server = "broker.hivemq.com";
const int mqtt_port = 1883; // non-secure port. Don’t send sensitive data
const char* topic = "AerialAV";
WiFiClient espClient;
PubSubClient client(espClient);
int second_payload_counter = 0;

void WiFiConnect(){
  Serial.println("Connecting to WiFi...");
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    Serial.println("Couldn't get a wifi connection. Trying again...");
    delay(1000);
    Serial.print(".");
  }
  Serial.println("Wifi connected!");
}

void ConnectToBroker(){
  while(!client.connected()){
    Serial.println("Connecting to MQTT Broker via port " + String(mqtt_port));

    if (client.connect("AerialAV")) {
```

```
      Serial.println("Connected!");
    }
    else{
      Serial.println("Failed to connect!");
      Serial.println(client.state());
      Serial.println("Reattempting....");
      delay(1000);
    }
  }
}

void setup() {
  pinMode(19, OUTPUT);
  digitalWrite(19, HIGH);
  Serial.begin(115200);
  while(!Serial)
    delay(1000);

  //begin bmp180, mpu and dht
  mpu.Begin();
  delay(2000);
  dht.begin();
  while (!bmp180.begin()){
    Serial.println(("Failed to connect"));
    delay(1000);
  }
  delay(2000);
  WiFiConnect();
  client.setServer(mqtt_server, mqtt_port); //connect to server
}

void loop() {
  if (!client.connected()){
    ConnectToBroker();
  }
  float* orientation = mpu.DeriveOrientation();

  // Create a JSON document
  StaticJsonDocument<200> json;
  // Add data to the JSON document
  json["orientation"]["roll"] = orientation[0];
  json["orientation"]["pitch"] = orientation[1];
```

```
json["orientation"]["yaw"] = orientation[2];
float* raw_imu = mpu.ProcessedDataPointerAccess();
json["accX"] = raw_imu[0];
json["accY"] = raw_imu[1];
json["accZ"] = raw_imu[2];
json["gyrX"] = raw_imu[3];
json["gyrY"] = raw_imu[4];
json["gyrZ"] = raw_imu[5];

// Serialize the JSON document to a string
String jsonString;
serializeJson(json, jsonString);
const char* message_payload = jsonString.c_str();
if(!client.publish(topic, message_payload, 0)){
  Serial.println("Failed to publish!");
}
else{
  Serial.println("Publish successful!");
}
  //SECOND PAYLOAD
  sensors_event_t event;
  StaticJsonDocument<200> json_2;
  dht.temperature().getEvent(&event);
  if (isnan(event.temperature)) {
    Serial.println("Problem comrade!");
  }
  json_2["temperature"] = bmp180.readTemperature();
  dht.humidity().getEvent(&event);
  // json_2["humidity"] = bmp180.readTemperature();;
  json_2["pressure"] = bmp180.readPressure();
  json_2["vibration"] = analogRead(SOUND_SEN_PIN);
  json_2["altitude"] = bmp180.readAltitude();

  // Serialize the JSON document to a string
  String jsonString_2;
  serializeJson(json_2, jsonString_2);

  Serial.println(jsonString_2);
  const char* message_payload_2 = jsonString_2.c_str();
  if(!client.publish(topic, message_payload_2, 2)){
    Serial.println("Failed to publish!");
  }
```

```
    else{
      Serial.println("Publish successful!");
    }
          }
}
```

APPENDIX B

```
void draw(){ //for redrawing every frame
  background(0);

  cam.beginHUD(); //prevents interference from 3d camera peasy
  //update graph data
  roll_ch.setData(convertToArray(time_collection),convertToArray(roll_collection));
  pitch_ch.setData(convertToArray(time_collection),convertToArray(pitch_collection));
  yaw_ch.setData(convertToArray(time_collection),convertToArray(yaw_collection));
  altitude_ch.setData(convertToArray(time_collection_2),convertToArray(altitude_collection));


temperature_ch.setData(convertToArray(time_collection_2),convertToArray(temperature_collectio
n));
  humidity_ch.setData(convertToArray(time_collection_2),convertToArray(humidity_collection));
  pressure_ch.setData(convertToArray(time_collection_2),convertToArray(pressure_collection));
  vibration_ch.setData(convertToArray(time_collection_2),convertToArray(vibration_collection));


  //draw graph
  roll_ch.draw(spacing_w,spacing_h,width/4,height/5);
  pitch_ch.draw(spacing_w,extra_spacing+(height/5),width/4,height/5);
  yaw_ch.draw(spacing_w,extra_spacing+(height/5)*2 + spacing_h,width/4,height/5);
  altitude_ch.draw(spacing_w,extra_spacing+(height/5)*3+ spacing_h*2,width/4,height/5);

  temperature_ch.draw(width-spacing_w-width/4,spacing_h,width/4,height/5);
  humidity_ch.draw(width-spacing_w-width/4,extra_spacing+(height/5),width/4,height/5);
       pressure_ch.draw(width-spacing_w-width/4,extra_spacing+(height/5)*2  +  spacing_h,width/
4,height/5);
     vibration_ch.draw(width-spacing_w-width/4,extra_spacing+(height/5)*3  +  spacing_h*2,width/
4,height/5);

  //ensure data is of limited size
  popCollections(roll_collection);
  popCollections(pitch_collection);
  popCollections(yaw_collection);
  popCollections(time_collection);
```

```
  popCollections(time_collection_2);
  popCollections(temperature_collection);
  popCollections(humidity_collection);
  popCollections(pressure_collection);
  popCollections(vibration_collection);
  popCollections(altitude_collection);

   //display direct sensor readings
  if(vibration_collection.size() > 0){
    textSize(13);
    text("Direct Sensor Readings",width/2-200,spacing_h*2);
    textSize(12);
    text("AccX(m/s^2): " +accX,width/2-200,spacing_h*2+15);
    text("AccY(m/s^2): " +accY,width/2-200,spacing_h*2+30);
    text("AccZ(m/s^2): " +accZ,width/2-200,spacing_h*2+45);
    text("GyrX(deg/s): " +gyrX,width/2-200,spacing_h*2+60);
    text("GyrY(deg/s): " +gyrY,width/2-200,spacing_h*2+75);
    text("GyrZ(Deg/s): " +gyrZ,width/2-200,spacing_h*2+90);

          text("Humidity(%): " +humidity_collection.get(humidity_collection.size()-1), width/2-200,
spacing_h*2+115);
        text("Temperature(oC): "+temperature_collection.get(temperature_collection.size()-1), width/
2-200, spacing_h*2+130);
            text("Pressure(pa): " +pressure_collection.get(pressure_collection.size()-1), width/2-200,
spacing_h*2+145);
             text("Vibration: " +vibration_collection.get(vibration_collection.size()-1), width/2-200,
spacing_h*2+160);
  }
  //display derived readings
  textSize(13);
  text("Derived Readings",width/2+50,spacing_h*2);
  textSize(12);
  if (roll_collection.size() > 0){
    text("Roll(deg): " +roll_collection.get(roll_collection.size()-1), width/2+50, spacing_h*2+15);
              text("Pitch(deg): " +pitch_collection.get(pitch_collection.size()-1), width/2+50,
spacing_h*2+30);
    text("Yaw(deg): "+yaw_collection.get(yaw_collection.size()-1), width/2+50, spacing_h*2+45);
              text("Altitude(m): "+altitude_collection.get(altitude_collection.size()-1), width/2+50,
spacing_h*3+45);
  }

  //graph labels
  textSize(13);
```

```
  text("Roll(deg)", spacing_w + 25,spacing_h*2);
  text("Pitch(deg)", spacing_w + 25,height/5+spacing_h*3);
  text("Yaw(deg)", spacing_w + 25,(height/5)*2+spacing_h*4);
  text("Altitude(m)", spacing_w + 25,(height/5)*3+spacing_h*5);
  text("Temeprature(oC)", width/1.335 - spacing_w + 25,spacing_h*2);
  text("Humidity(%)",  width/1.335 - spacing_w + 25,height/5+spacing_h*3);
  text("Pressure(Pa)",  width/1.335 - spacing_w + 25,(height/5)*2+spacing_h*4);
  text("Vibration(mag)",  width/1.335 - spacing_w + 25,(height/5)*3+spacing_h*5);
  cam.endHUD();

  push();
  translate(width/2, height/2);
  scale(5);
  rotateX(PI);
  rotateY(PI);

  //light
  lights();
  lightSpecular(255, 220, 180);
  directionalLight(255, 220, 180, 1, 0, 1);
  ambient(255, 220, 180);
  //rotate 3d model according to orientation readings
  if (roll_collection.size() > 0){ //ensure collection is greater than 0;
    int last_pos = roll_collection.size()-1;
    rotateY(-1.0*radians(yaw_collection.get(last_pos)));
    rotateX(radians(pitch_collection.get(last_pos)));
    rotateZ(radians(roll_collection.get(last_pos)));
  }
  //axis lines
  stroke(0,0,255); //make blue
  line(0,-5,0,5); //z axis
  stroke(255,0,0); //make red
  line(-5,0,5,0); //x axis
  push();
  stroke(0,255,0); //make green
  rotateX(PI/2);
  line(0,-5,0,5); //y axis

  pop();
  shape(model);
  pop();
}
```

APPENDIX C