MongoDB connection

```
import paho.mqtt.client as mqtt
import json
import csv
import os
from datetime import datetime
import pandas as pd
from pymongo import MongoClient
from urllib.parse import quote_plus
import pymongo
import matplotlib.pyplot as plt
import redis
```

The code works as follows. First, make a connection to MongoDB, the define the details for MQTT brokers, define some functions for callbacks. When the message is received, I parse the message, save the data in a csv for later analysis, and upload it the MongoDB

```
# MongoDB connection
mongo_username = quote_plus("")
mongo_password = quote_plus("")
mongo_uri = f"mongodb+srv://{mongo_username}:{mongo_password}@cluster0.950if.mongodb.net/" #
mongo_client = MongoClient(mongo_uri)
db = mongo_client["sensor_data"]
collection = db["accelerometer"]

# MQTT brokers details
broker = "943103f9e51648f9b2e42b0741f78511.s1.eu.hivemq.cloud"
port = 8883
username = ""
password = ""
topic = "data" # topic to subscribe to

# Function to get the current time
```

```
def timestamp():
    return datetime.now().strftime('%Y%m%d%H%M%S')

# Callback when the client connects to the broker
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to broker")
        client.subscribe(topic)
    else:
        print(f"Failed to connect, return code {rc}")

# Callback when a message is received
def on_message(client, userdata, msg):
    payload = msg.payload.decode()
    print(f"Message received: {msg.topic} {payload}")

    # Parse the data
    data = json.loads(payload)
    formatted_data = {
        "Timestamp": timestamp(),
        "x": data["x"],
        "y": data["y"],
        "z": data["z"]
    }

    # Save to CSV file
    with open("data.csv", mode='a', newline='') as csv_file:
        csv_writer = csv.writer(csv_file)
        csv_writer.writerow(formatted_data.values())

    # Save to JSON file
    df = pd.read_csv("data.csv", header=None, names=['Timestamp', 'x', 'y', 'z']) # read csv
    # json_data = df.to_json(orient='records') # convert dataframe to json dictionaries
    # with open('data.json', 'w') as file2:
    #     json.dump(json_data, file2, indent=4)

    # Upload data to MongoDB
    collection.insert_one(formatted_data)
    print("Data inserted into MongoDB")

# Initialize client
client = mqtt.Client()
```

```
# Set callbacks
client.on_connect = on_connect
client.on_message = on_message

# Set username and passwords
client.username_pw_set(username, password)

# Enable SSL/TLS
client.tls_set()

# Connect to the broker
client.connect(broker, port)

# Start the loop to process received messages
client.loop_forever()
```

The duration of the loop to process received messages is around 47 minutes.

Query data

```
# Query the first 5 document
all_documents = collection.find().limit(5)

for document in all_documents:
    print(document)
```

```
{'_id': ObjectId('66b962b99730dfa36b172e53'), 'Timestamp': '20240812111745', 'x': 0.00500488
{'_id': ObjectId('66b962bd9730dfa36b172e54'), 'Timestamp': '20240812111749', 'x': 0.0078125,
{'_id': ObjectId('66b962c19730dfa36b172e55'), 'Timestamp': '20240812111753', 'x': 0.00109863
{'_id': ObjectId('66b962c69730dfa36b172e56'), 'Timestamp': '20240812111758', 'x': 0.00024414
{'_id': ObjectId('66b962cb9730dfa36b172e57'), 'Timestamp': '20240812111803', 'x': 0.00097656
```

This simple query shows that the data is in the database. It will be a hassle to transform the document format into csv file, so I have deliberately done that inside the loop.

```
df1 = pd.read_csv("data.csv", header=None, names=['Timestamp', 'x', 'y', 'z'])
display(df1.head())
display(df1.info())
```

	Timestamp	x	y	z
0	20240813092615	0.000000	-0.016235	1.023071
1	20240813092750	-0.000732	-0.014648	1.024536
2	20240813092842	-0.001831	-0.016846	1.026733
3	20240813092921	0.001099	-0.015869	1.025513
4	20240813093050	0.000610	-0.015991	1.026611

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17 entries, 0 to 16
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Timestamp  17 non-null     int64
 1   x          17 non-null     float64
 2   y          17 non-null     float64
 3   z          17 non-null     float64
dtypes: float64(3), int64(1)
memory usage: 676.0 bytes
```

```
None
```

```
# Convert `Timestamp` column to time dtype
df1['Timestamp'] = pd.to_datetime(df1['Timestamp'], format='%Y%m%d%H%M%S')
df1.head()
```

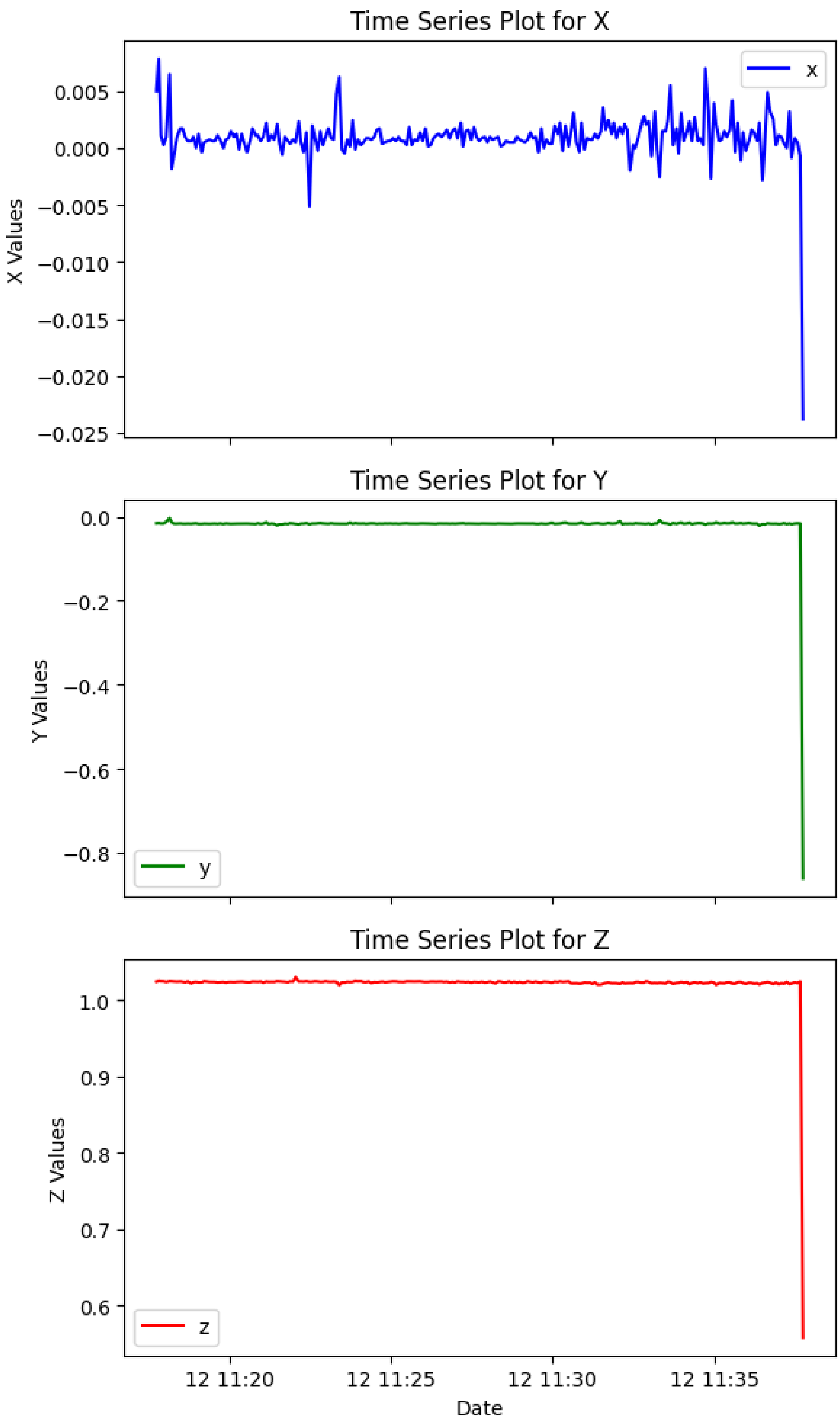
	Timestamp	x	y	z
0	2024-08-12 11:17:45	0.005005	-0.015991	1.024292
1	2024-08-12 11:17:49	0.007812	-0.015381	1.025391
2	2024-08-12 11:17:53	0.001099	-0.016479	1.024902
3	2024-08-12 11:17:58	0.000244	-0.016357	1.024780
4	2024-08-12 11:18:03	0.000977	-0.012573	1.023682

```
df1.describe()
```

	x	y	z
count	241.000000	241.000000	241.000000
mean	0.000947	-0.020217	1.021411

	x	y	z
std	0.002130	0.054396	0.030032
min	-0.023804	-0.860840	0.557495
25%	0.000488	-0.017212	1.022705
50%	0.000854	-0.016846	1.023560
75%	0.001465	-0.016479	1.024170
max	0.007812	-0.003662	1.029785

```
# Subplots
df1.set_index('Timestamp', inplace=True)
fig, axs = plt.subplots(3, 1, figsize=(6, 10), sharex=True)

# Plot x
axs[0].plot(df1.index, df1['x'], label='x', color='b')
axs[0].set_title('Time Series Plot for X')
axs[0].set_ylabel('X Values')
axs[0].legend()

# Plot y
axs[1].plot(df1.index, df1['y'], label='y', color='g')
axs[1].set_title('Time Series Plot for Y')
axs[1].set_ylabel('Y Values')
axs[1].legend()

# Plot z
axs[2].plot(df1.index, df1['z'], label='z', color='r')
axs[2].set_title('Time Series Plot for Z')
axs[2].set_xlabel('Date')
axs[2].set_ylabel('Z Values')
axs[2].legend()

# Display the plot
plt.tight_layout()
plt.show()
```


Time Series Plot for X
x
0.005
0.000
−0.005
−0.010
−0.015
−0.020
−0.025
X Values
Time Series Plot for Y
0.0
−0.2
−0.4
−0.6
−0.8
Y Values
y
Time Series Plot for Z
1.0
0.9
0.8
0.7
0.6
Z Values
z
12 11:20
12 11:25
12 11:30
12 11:35
Date

These 3 plots show the 3 rotations axis of the Gyroscope. So I left the sensor stationary on my desk while I was working. There seems to be minor fluctuations on the x-axis, my guess would be because I shake my legs while sitting, which causes some movements. The y and z rotations seem to be almost stationary for the whole recording period. At the end, there seems to be a sharp drop, this is likely due to my interaction with the sensor while interrupting the program.

Redis connection

```
# Redis connection
r = redis.Redis(
    host='localhost',
    port=6379,
    db=0
)

# MQTT brokers details
broker = "943103f9e51648f9b2e42b0741f78511.s1.eu.hivemq.cloud"
port = 8883
username = ""
password = ""
topic = "data" # topic to subscribe to

# Function to get the current time
def timestamp():
    return datetime.now().strftime('%Y%m%d%H%M%S')

# Callback when the client connects to the broker
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to broker")
        client.subscribe(topic)
    else:
        print(f"Failed to connect, return code {rc}")

# Callback when a message is received
def on_message(client, userdata, msg):
    payload = msg.payload.decode()
    print(f"Message received: {msg.topic} {payload}")

    # Parse the data
```

```python
    data = json.loads(payload)
    formatted_data = {
        "Timestamp": timestamp(),
        "x": data["x"],
        "y": data["y"],
        "z": data["z"]
    }

    # Save to CSV file
    with open("data.csv", mode='a', newline='') as csv_file:
        csv_writer = csv.writer(csv_file)
        csv_writer.writerow(formatted_data.values())

    # Save to JSON file
    df = pd.read_csv("data.csv", header=None, names=['Timestamp', 'x', 'y', 'z']) # read csv
    # json_data = df.to_json(orient='records') # convert dataframe to json dictionaries
    # with open('data.json', 'w') as file2:
    #     json.dump(json_data, file2, indent=4)

    # Store data in Redis
    counter = r.incr("gyro_counter")
    key = f"gyro: {counter}"  # Use the timestamp as a unique key

    for field, value in formatted_data.items():
        r.hset(key, field, value)
    print(f"Stored data with key: {key}")

# Initialize client
client = mqtt.Client()

# Set callbacks
client.on_connect = on_connect
client.on_message = on_message

# Set username and passwords
client.username_pw_set(username, password)

# Enable SSL/TLS
client.tls_set()

# Connect to the broker
client.connect(broker, port)
```

```
# Start the loop to process received messages
client.loop_forever()
```

85 minutes recording

Query data

```
# Retrieve and print data and Redis
keys = r.keys('gyro:*')
for key in keys:
    print(f"{key.decode()}: {r.hgetall(key)}")
```

```
gyro: 3: {b'Timestamp': b'20240813093418', b'x': b'-0.00012207', b'y': b'-0.015869141', b'z'
gyro: 8: {b'Timestamp': b'20240813093443', b'x': b'-0.000366211', b'y': b'-0.014282227', b'z
gyro: 1: {b'Timestamp': b'20240813093412', b'x': b'0.000488281', b'y': b'-0.018188477', b'z'
gyro: 9: {b'Timestamp': b'20240813093448', b'x': b'-0.001098633', b'y': b'-0.016845703', b'z
gyro: 2: {b'Timestamp': b'20240813093413', b'x': b'-0.002685547', b'y': b'-0.014160156', b'z
gyro: 6: {b'Timestamp': b'20240813093433', b'x': b'-0.000732422', b'y': b'-0.016235352', b'z
gyro: 4: {b'Timestamp': b'20240813093423', b'x': b'0.000488281', b'y': b'-0.016479492', b'z'
gyro: 5: {b'Timestamp': b'20240813093428', b'x': b'0.002807617', b'y': b'-0.014038086', b'z'
gyro: 7: {b'Timestamp': b'20240813093438', b'x': b'-0.000976563', b'y': b'-0.018798828', b'z
```

Since I have already saved the data inside of a csv file, I will analyze from it.

```
df2 = pd.read_csv("./redis data/data.csv", header=None, names=['Timestamp', 'x', 'y', 'z'])
display(df2.head())
display(df2.info())
```

	Timestamp	x	y	z
0	20240812172100	0.002197	-0.015137	1.023682
1	20240812172103	0.001709	-0.016968	1.022949
2	20240812172107	0.001953	-0.017090	1.023071
3	20240812172112	-0.003662	-0.018799	1.022949
4	20240812172117	0.001953	-0.016724	1.023560

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1020 entries, 0 to 1019
Data columns (total 4 columns):
```

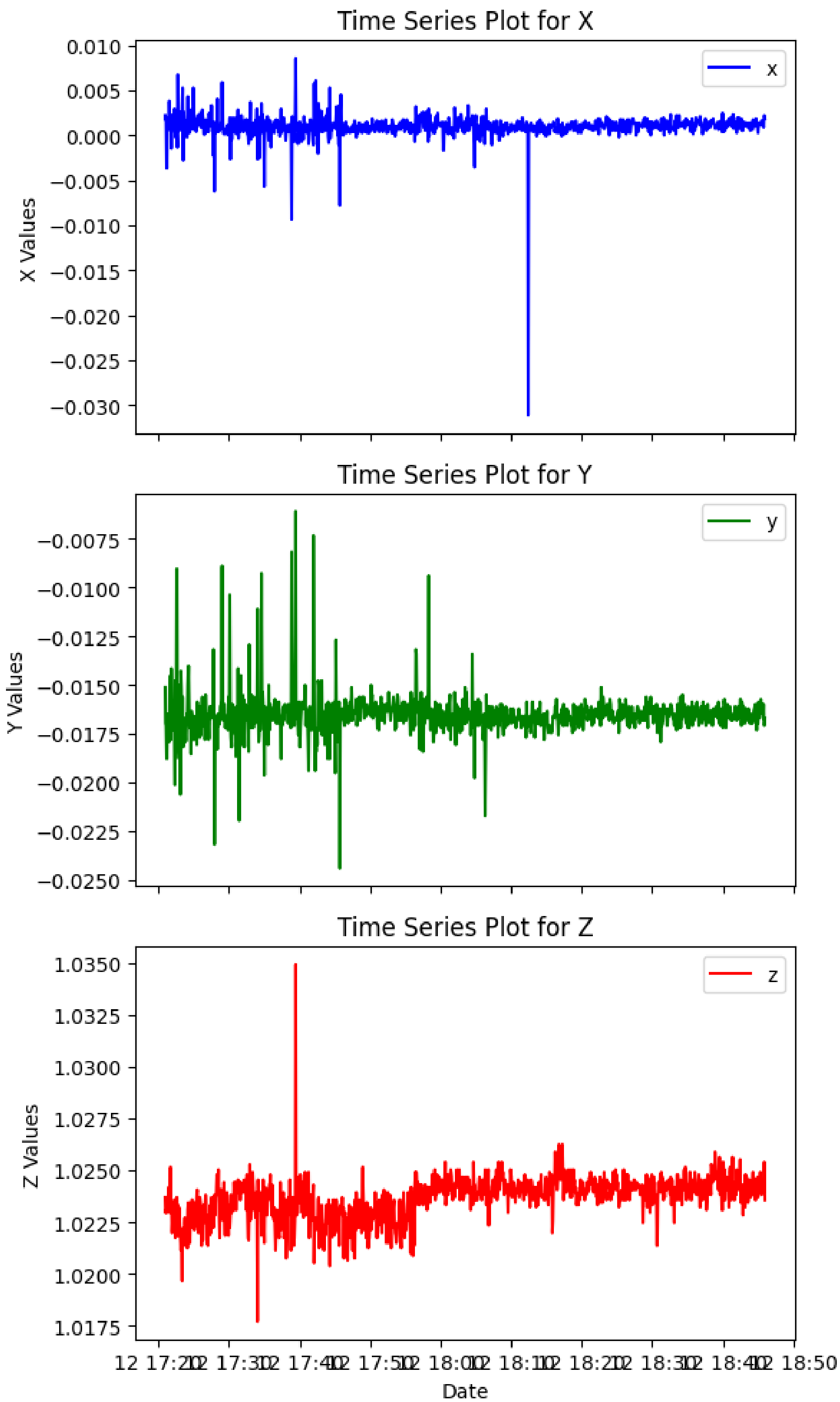
```
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Timestamp  1020 non-null   int64
 1   x          1020 non-null   float64
 2   y          1020 non-null   float64
 3   z          1020 non-null   float64
dtypes: float64(3), int64(1)
memory usage: 32.0 KB

None
```

```
# Convert `Timestamp` column to time dtype
df2['Timestamp'] = pd.to_datetime(df2['Timestamp'], format='%Y%m%d%H%M%S')
df2.head()
```

	Timestamp	x	y	z
0	2024-08-12 17:21:00	0.002197	-0.015137	1.023682
1	2024-08-12 17:21:03	0.001709	-0.016968	1.022949
2	2024-08-12 17:21:07	0.001953	-0.017090	1.023071
3	2024-08-12 17:21:12	-0.003662	-0.018799	1.022949
4	2024-08-12 17:21:17	0.001953	-0.016724	1.023560

```
# Subplots
df2.set_index('Timestamp', inplace=True)
fig, axs = plt.subplots(3, 1, figsize=(6, 10), sharex=True)

# Plot x
axs[0].plot(df2.index, df2['x'], label='x', color='b')
axs[0].set_title('Time Series Plot for X')
axs[0].set_ylabel('X Values')
axs[0].legend()

# Plot y
axs[1].plot(df2.index, df2['y'], label='y', color='g')
axs[1].set_title('Time Series Plot for Y')
axs[1].set_ylabel('Y Values')
axs[1].legend()

# Plot z
axs[2].plot(df2.index, df2['z'], label='z', color='r')
```

```python
axs[2].set_title('Time Series Plot for Z')
axs[2].set_xlabel('Date')
axs[2].set_ylabel('Z Values')
axs[2].legend()

# Display the plot
plt.tight_layout()
plt.show()
```


Time Series Plot for X
X Values
x
0.010
0.005
0.000
−0.005
−0.010
−0.015
−0.020
−0.025
−0.030
Time Series Plot for Y
Y Values
y
−0.0075
−0.0100
−0.0125
−0.0150
−0.0175
−0.0200
−0.0225
−0.0250
Time Series Plot for Z
Z Values
z
1.0350
1.0325
1.0300
1.0275
1.0250
1.0225
1.0200
1.0175
Date

For this data recording, I have left the Gyroscope sensor stationary on my desk while I was cooking. There seems to be an small fluctuations in all of the rotations. My best guess would be that the Arduino Board moves on itself due to the way I station the wires, which causes the board to rotate slightly.

Compare the databases

Criteria	Firebase	MongoDB	Redis
Configuration	Minimal set up	Can either be self-hosted or via cloud	Can either be self-hosted or via cloud
Data Storage Model	NoSQL document database	NoSQL document database	In memory key value database with support for other data structure
Data Storing APIs	JSON	BSON	Key-value pairs with support for other data structure
Scalability	Automatic scaling	Horizontal scaling	Horizontal scaling in cloud version
Performance	High performance	High performance	Very fast due to in-memory storing
Offline Support	Offline support and sync when online	Offline support and sync when online	In-memory database
Pricing	has both free and paid tier	has both free and paid tier	in-memory is free, cloud is paid
Complexity	can be both simple and complex	moderate complexity	simple, but can be complex if need be
Integration	Integrate with other Firebases and Google Cloud services	Works well with data tools	works well with in-memory database and microservices

For more information, please check its respective website.

Serial communications vs MQTT in terms of data sending and receiving/parsing effort

Serial communication is a method that uses one or two transmission lines that send (TX) and receive (RX) data (CONTEC, 2022). The data is sent bit by bit. Common protocols under serial communication are UART, I2C, and SPI. Since data is transmitted 1 bit at a time, then the protocol complexity is not high. The parsing effort is quite straightforward, which involves continuous stream of data, correct baud rate and framing. But because its simple to set up, it is not scalable to a network of devices and can only be used for device-to-device connection (CONTEC, 2022).

MQTT protocol is a lightweight, publish/subscribe messaging transport that can work for resource-constrained devices and unreliable networks (P, P.K., 2021). Publish subscribe protocol means devices (clients) can publish data to a broken (MQTT service), and other devices can subscribe the topic to get those data. The concept and the working may seem daunting, but MQTT services like Hivemq does a great job in abstracting the complexity, and allows us to easily publish and subscribe to topic. The parsing of MQTT protocol is simplified by topics and payloads, where each message is associated to a topic, and a topic has payload or data. The format is usually in JSON and to receive those data, a device need to subscribe to a certain topic. The benefit of it has over serial communication is the ability to work under unreliable networks and the ability to scale for many devices across large networks. The only downside of MQTT protocol is that the message are not inherently real-time, meaning there is a slight delay compare to Serial communication (P, P.K., 2021).

Q3.

Arduino code to connect to HiveMQ

```
#include <Arduino_LSM6DS3.h>

#include <PubSubClient.h>

#include <SPI.h>

#include <WiFiNINA.h>

#include <ArduinoJson.h>


// WiFi Credentials

char ssid[] = "";

char pass[] = "";


int status = WL_IDLE_STATUS;


// Initialize WiFi client

WiFiSSLClient wifi_client;


// Connect to Wifi Access Point

void connectToAP(){

  while (status != WL_CONNECTED){

      Serial.print("Attempting to connect to SSID: ");

      Serial.println(ssid);


      // Connect to WPA/WP2 network

      status = WiFi.begin(ssid, pass);


      // wait 5 second for connection

      delay(5000);
```

```
      if (status == WL_CONNECTED){
          Serial.println("Connected to WiFi");
          printWifiStatus();
        }else{
            Serial.println("Failed to connect. Retrying...");
          }
      Serial.println("Connected");
    }
  }

// Print results to serial monitor
void printWifiStatus(){
  // Network SSID
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  // Device IP address
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);
  }

// Gyroscope axis
float x, y, z;



/******* MQTT Broker Connection Details *******/
const char* mqtt_server = "943103f9e51648f9b2e42b0741f78511.s1.eu.hivemq.cloud";
const char* mqtt_username = "";
```

```
const char* mqtt_password = "";
const int mqtt_port =8883;

/**** MQTT Client Initialisation Using WiFi Connection *****/
PubSubClient client(wifi_client);

unsigned long lastMsg = 0;
#define MSG_BUFFER_SIZE (50)
char msg[MSG_BUFFER_SIZE];


/************* Connect to MQTT Broker ***********/
void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    String clientId = "LongID";   // Create a random client ID
    //clientId += String(random(0xffff), HEX);
    // Attempt to connect
    if (client.connect(clientId.c_str(), mqtt_username, mqtt_password)) {
      Serial.println("connected");

     //
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");   // Wait 5 seconds before
retrying
      delay(5000);
```

```
    }
  }
}



/***** Call back Method for Receiving MQTT messages ****/

void callback(char* topic, byte* payload, unsigned int length) {
  String incommingMessage = "";
  for (int i = 0; i < length; i++) incommingMessage+=(char)payload[i];

  Serial.println("Message arrived ["+String(topic)+"]"+incommingMessage);

  //--- check the incomming message
    if( strcmp(topic,"led_state") == 0){
     //
  }

}



/**** Method for Publishing MQTT Messages **********/
void publishMessage(const char* topic, String payload , boolean retained){
  if (client.publish(topic, payload.c_str(), true)){
      Serial.println("Message publised ["+String(topic)+"]: "+payload);
    } else{
        Serial.println("Message publish failed");
      }
}
```

```
void setup() {
  Serial.begin(9600); // set baud rate
  while (!Serial);  // wait for port to init

  if (!IMU.begin()) {
    while (1);
  }

  // Check for the WiFi module
  if (WiFi.status() == WL_NO_MODULE){
      Serial.println("WiFi module failed!");
      while (true);
    }

  // wifi connection
  connectToAP();
  printWifiStatus();


  //mqtt connection
  client.setServer(mqtt_server, mqtt_port);
  client.setCallback(callback);
}

void loop() {
  // check if client is connected
  if (!client.connected()) reconnect(); // check if client is connected
```

```
  client.loop();

  // read accelero data
  if (IMU.accelerationAvailable()) {
    IMU.readAcceleration(x, y, z);
  }

  Serial.println(String(x) + "," + String(y) + "," + String(z));

  DynamicJsonDocument doc(1024);
  doc["x"] = x;
  doc["y"] = y;
  doc["z"] = z;

  char mqtt_message[128];
  serializeJson(doc, mqtt_message);

  publishMessage("data", mqtt_message, true); // send topic and data



  delay(5000); // delay 5s
}
```

This code works by connecting the Arduino Nano 33 to the internet and connecting to the MQTT broker to send our topic and message. The code for connecting to the internet I got it from DroneBot Workshop (2021) and for the HiveMQ connection it’s from Kudzai Manditereza (2023). There are comments for each functions, like how is the connection and status to the internet, the connection, callback, publish function for the MQTT.

Python code for Mongo DB connection

```
# MongoDB connection

mongo_username = quote_plus("")

mongo_password = quote_plus("")

mongo_uri =
f"mongodb+srv://{mongo_username}:{mongo_password}@cluster0.950if.mongodb.net/" #
MongoDB connection URI

mongo_client = MongoClient(mongo_uri)

db = mongo_client["sensor_data"]

collection = db["accelerometer"]


# MQTT brokers details

broker = "943103f9e51648f9b2e42b0741f78511.s1.eu.hivemq.cloud"

port = 8883

username = ""

password = ""

topic = "data" # topic to subscribe to


# Function to get the current time

def timestamp():

    return datetime.now().strftime('%Y%m%d%H%M%S')


# Callback when the client connects to the broker

def on_connect(client, userdata, flags, rc):

    if rc == 0:

        print("Connected to broker")

        client.subscribe(topic)

    else:

        print(f"Failed to connect, return code {rc}")
```

```
# Callback when a message is received
def on_message(client, userdata, msg):
    payload = msg.payload.decode()
    print(f"Message received: {msg.topic} {payload}")

    # Parse the data
    data = json.loads(payload)
    formatted_data = {
        "Timestamp": timestamp(),
        "x": data["x"],
        "y": data["y"],
        "z": data["z"]
    }

    # Save to CSV file
    with open("data.csv", mode='a', newline='') as csv_file:
        csv_writer = csv.writer(csv_file)
        csv_writer.writerow(formatted_data.values())

    # Save to JSON file
    df = pd.read_csv("data.csv", header=None, names=['Timestamp', 'x', 'y', 'z'])
# read csv file
    # json_data = df.to_json(orient='records') # convert dataframe to json
dictionaries
    # with open('data.json', 'w') as file2:
    #     json.dump(json_data, file2, indent=4)

    # Upload data to MongoDB
    collection.insert_one(formatted_data)
```

```
    print("Data inserted into MongoDB")


# Initialize client

client = mqtt.Client()


# Set callbacks

client.on_connect = on_connect

client.on_message = on_message


# Set username and passords

client.username_pw_set(username, password)


# Enable SSL/TLS

client.tls_set()


# Connect to the broker

client.connect(broker, port)


# Start the loop to process received messages

client.loop_forever()
```

The code works as follows. First, make a connection to MongoDB, then define the details for MQTT brokers, define some functions for callbacks. When the message is received, I parse the message, save the data in a csv for later analysis, and upload it the MongoDB. The code for Redis is quite similar to this.

Q4.

https://www.youtube.com/watch?v=C8xQnIwlwL8

Q5.

https://github.com/tomadonna1/SIT225_2024T2/tree/main/Distinction%20Task%20Store%20data%20to%20non-Firebase%20cloud%20database

References:

CONTEC. (2022). *Serial communication Basic Knowledge -RS-232C/RS-422/RS-485- | CONTEC*. [online] Available at: https://www.contec.com/support/basic-knowledge/daq-control/serial-communicatin/#:~:text=and%20synchronous%20communication-.

DroneBot Workshop (2021). *Arduino Nano 33 IoT - Getting Started*. [online] www.youtube.com. Available at: https://www.youtube.com/watch?v=KjETPxv_VR4.

Kudzai Manditereza (2023). *Getting Started with MQTT on Arduino Using NodeMCU ESP8266*. [online] Hivemq.com. Available at: https://www.hivemq.com/blog/mqtt-on-arduino-nodemcu-esp8266-hivemq-cloud/#heading-setting-up-a-hive-mq-cloud-mqtt-broker.

P, P.K. (2021). *The Modern industrial world is headed to IIoT digital platforms in these days, but the difficulties being faced by industries are the protocols that supports IoT hubs and clouds are limited to some light weight protocols. Most of the industries have still legacy serial communication equipment in the*. [online] Linkedin.com. Available at: https://www.linkedin.com/pulse/serial-mqtt-protocol-conversion-prasanth-k-p/.