task5-2d

September 5, 2024

[14]:
```
# Import necessary packages
from pymongo.mongo_client import MongoClient # MongoDB
from pymongo.server_api import ServerApi
import couchdb # CouchDB

import pandas as pd
import csv
import seaborn as sns
import matplotlib.pyplot as plt
import paho.mqtt.client as paho
from paho import mqtt
import json
import datetime
import requests
from requests.auth import HTTPBasicAuth
```

MongoDB

[49]:
```
mongo_uri = "mongodb+srv://thinhtruongkhangnguyen1404:abce12322HH@gyrodata.
↪2kirg.mongodb.net/?retryWrites=true&w=majority&appName=GyroData"
client = MongoClient(mongo_uri)
db = client['MongoDB_Testing']
collection = db['GyroData_Testing']

# MQTT settings
mqtt_broker = "7a783a7c2ba249cc8ca373ec1c6ba990.s1.eu.hivemq.cloud"
mqtt_topic = "Gyro_Data_MongoDB"
mqtt_user = "Ntkt05"
mqtt_password = "abce123@@HH"


# setting callbacks for different events to see if it works, print the message␣
↪etc.
def on_connect(client, userdata, flags, rc, properties=None):
    print("CONNACK received with code %s." % rc)

# with this callback you can see if your publish was successful
def on_publish(client, userdata, mid, properties=None):
```

```
    print("mid: " + str(mid))

# print which topic was subscribed to
def on_subscribe(client, userdata, mid, granted_qos, properties=None):
    print("Subscribed: " + str(mid) + " " + str(granted_qos))

# print message, useful for checking if it was successful
def on_message(client, userdata, message):
    payload = message.payload.decode("utf-8")
    print(f"Received message: {payload}")
    var = payload.split(",")
    doc = {
        "Timestamp" : datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S'),
        "x": var[0],
        "y": var[1],
        "z": var[2]
    }
    # Assuming payload is a JSON string
    try:
        # Insert data into MongoDB
        collection.insert_one(doc)
        print("Data inserted into MongoDB")
    except json.JSONDecodeError:
        print("Error decoding JSON")

# userdata is user defined data of any type, updated by user_data_set()
# client_id is the given name of the client
client = paho.Client(client_id="", userdata=None, protocol=paho.MQTTv5)
client.on_connect = on_connect

# enable TLS for secure connection
client.tls_set(tls_version=mqtt.client.ssl.PROTOCOL_TLS)
# set username and password
client.username_pw_set(mqtt_user, mqtt_password)
# connect to HiveMQ Cloud on port 8883 (default for MQTT)
client.connect(mqtt_broker, 8883)

# setting callbacks, use separate functions like above for better visibility
client.on_subscribe = on_subscribe
client.on_message = on_message
client.on_publish = on_publish

# subscribe to all topics of encyclopedia by using the wildcard "#"
client.subscribe("Gyro_Data_MongoDB" ,qos=1)

client.loop_forever()
```

```
C:\Users\thinh\AppData\Local\Temp\ipykernel_24632\3562746668.py:46:
DeprecationWarning: Callback API version 1 is deprecated, update to latest
version
  client = paho.Client(client_id="", userdata=None, protocol=paho.MQTTv5)

CONNACK received with code Success.
Subscribed: 1 [ReasonCode(Suback, 'Granted QoS 1')]
Received message: 0.31, 0.61, 0.67
Data inserted into MongoDB
Received message: 0.37, 0.61, 0.67
Data inserted into MongoDB
Received message: 0.37, 0.61, 0.67
Data inserted into MongoDB
Received message: 0.37, 0.61, 0.67
Data inserted into MongoDB
Received message: 0.31, 0.61, 0.67
Data inserted into MongoDB
Received message: 0.37, 0.61, 0.67
Data inserted into MongoDB
Received message: 0.37, 0.61, 0.61
Data inserted into MongoDB
Received message: 0.37, 0.55, 0.67
Data inserted into MongoDB
Received message: 0.37, 0.61, 0.67
Data inserted into MongoDB
Received message: 0.31, 0.61, 0.67
Data inserted into MongoDB
Received message: 0.31, 0.61, 0.67
Data inserted into MongoDB
Received message: 0.31, 0.61, 0.67
Data inserted into MongoDB
Received message: 0.37, 0.61, 0.61
Data inserted into MongoDB
Received message: 0.37, 0.61, 0.61
Data inserted into MongoDB
Received message: 0.31, 0.61, 0.67
Data inserted into MongoDB
Received message: 0.31, 0.61, 0.67
Data inserted into MongoDB
Received message: 0.31, 0.61, 0.67
Data inserted into MongoDB
Received message: 0.31, 0.55, 0.67
Data inserted into MongoDB
Received message: 0.31, 0.61, 0.67
Data inserted into MongoDB
Received message: 0.37, 0.61, 0.67
Data inserted into MongoDB
Received message: 0.31, 0.67, 0.61
```

```
Data inserted into MongoDB
Received message: 0.31, 0.61, 0.67
Data inserted into MongoDB
Received message: 0.31, 0.61, 0.67
Data inserted into MongoDB
Received message: 0.37, 0.61, 0.67
Data inserted into MongoDB
Received message: 0.31, 0.61, 0.67
Data inserted into MongoDB
Received message: 0.37, 0.61, 0.67
Data inserted into MongoDB
Received message: 0.31, 0.61, 0.67
Data inserted into MongoDB
Received message: 0.31, 0.61, 0.67
Data inserted into MongoDB
Received message: 0.31, 0.61, 0.67
Data inserted into MongoDB
Received message: 0.37, 0.61, 0.67
Data inserted into MongoDB
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[49], line 64
     61 # subscribe to all topics of encyclopedia by using the wildcard "#"
     62 client.subscribe("Gyro_Data_MongoDB" ,qos=1)
---> 64 client.loop_forever()

File ~\Documents\ANACONDA\Lib\site-packages\paho\mqtt\client.py:2297, in Client
 ↪loop_forever(self, timeout, retry_first_connection)
   2295 rc = MQTTErrorCode.MQTT_ERR_SUCCESS
   2296 while rc == MQTTErrorCode.MQTT_ERR_SUCCESS:
-> 2297     rc = self._loop(timeout)
   2298     # We don't need to worry about locking here, because we've
   2299     # either called loop_forever() when in single threaded mode, or
   2300     # in multi threaded mode when loop_stop() has been called and
   2301     # so no other threads can access _out_packet or _messages.
   2302     if (self._thread_terminate is True
   2303         and len(self._out_packet) == 0
   2304             and len(self._out_messages) == 0):

File ~\Documents\ANACONDA\Lib\site-packages\paho\mqtt\client.py:1663, in Client
 ↪_loop(self, timeout)
   1660     rlist = [self._sock, self._sockpairR]
   1662 try:
-> 1663     socklist = select.select(rlist, wlist, [], timeout)
   1664 except TypeError:
   1665     # Socket isn't correct type, in likelihood connection is lost
```

```
   1666     # … or we called disconnect(). In that case the socket will
   (…)
   1669     # rc != MQTT_ERR_SUCCESS and we don't want state to change from
   1670     # mqtt_cs_disconnecting.
   1671     if self._state not in (_ConnectionState.MQTT_CS_DISCONNECTING,␣
 ↪_ConnectionState.MQTT_CS_DISCONNECTED):

KeyboardInterrupt:
```

[ ]:
```
data_use = collection.find()

with open('Mongodb_Demonstrate.csv', 'w', newline='') as csvfile:
    writer = csv.DictWriter(csvfile, fieldnames=['Id', 'Timestamp', 'x', 'y',␣
 ↪'z'])
    writer.writeheader()

    for document in data_use:
        writer = csv.writer(csvfile)
        # Write the data rows
        writer.writerow(document.values())
```

[25]:
```
# Analyse the recored CSV file
mongo_df = pd.read_csv("Mongo.csv")
mongo_df
```

[25]:

	_id	Timestamp	x	y	z
0	66c97d2f00cbf1ca6a82dc67	2024-08-24 16:26:55.480000	0.31	1.10	0.49
1	66c97d3100cbf1ca6a82dc68	2024-08-24 16:26:57.507000	0.37	0.31	0.49
2	66c97d3300cbf1ca6a82dc69	2024-08-24 16:26:59.491000	0.31	0.31	0.49
3	66c97d3500cbf1ca6a82dc6a	2024-08-24 16:27:01.496000	0.18	0.37	0.49
4	66c97d3700cbf1ca6a82dc6b	2024-08-24 16:27:03.504000	0.37	0.49	0.49
…	…	…	…	…	…
1813	66c98b6200cbf1ca6a82e37c	2024-08-24 17:27:30.668000	-0.37	0.55	0.49
1814	66c98b6400cbf1ca6a82e37d	2024-08-24 17:27:32.669000	-0.73	2.69	0.49
1815	66c98b6700cbf1ca6a82e37e	2024-08-24 17:27:35.132000	0.43	0.79	0.49
1816	66c98b6800cbf1ca6a82e37f	2024-08-24 17:27:36.683000	0.12	0.98	0.55
1817	66c98b6a00cbf1ca6a82e380	2024-08-24 17:27:38.684000	0.18	0.24	0.49

[1818 rows x 5 columns]

[26]:
```
# Preprocessing the Data
mongo_df.drop(columns = ["_id"], inplace = True)
mongo_df["Timestamp"] = pd.to_datetime(mongo_df["Timestamp"], errors='coerce')
mongo_df
```

[26]:

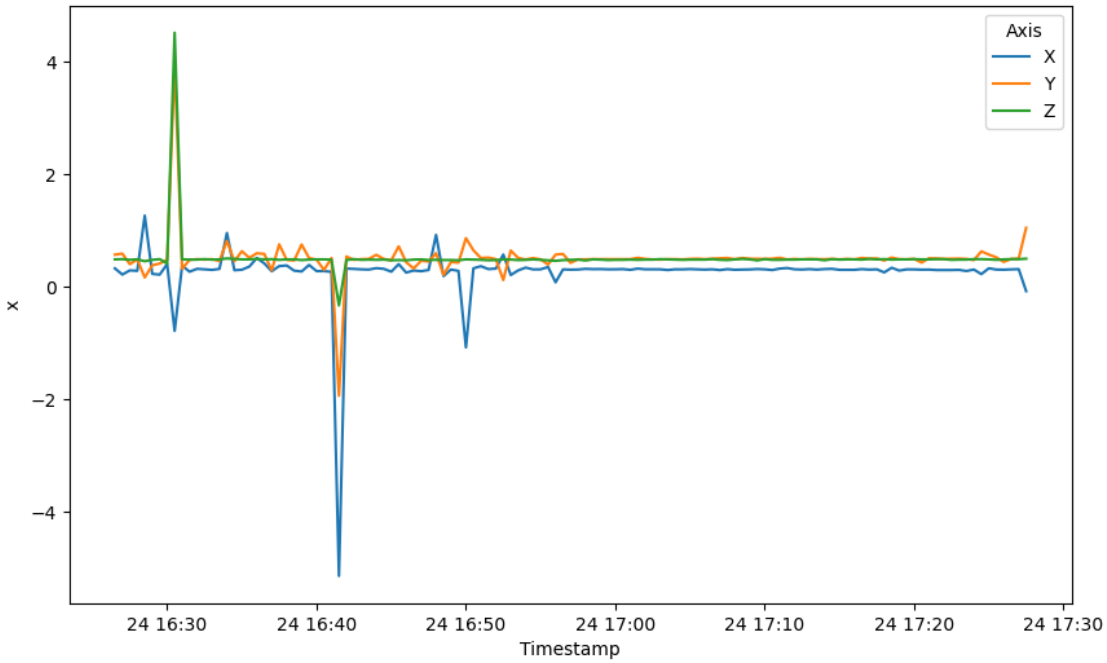
	Timestamp	x	y	z
0	2024-08-24 16:26:55.480	0.31	1.10	0.49
1	2024-08-24 16:26:57.507	0.37	0.31	0.49
2	2024-08-24 16:26:59.491	0.31	0.31	0.49
3	2024-08-24 16:27:01.496	0.18	0.37	0.49
4	2024-08-24 16:27:03.504	0.37	0.49	0.49
…	…	…	…	…
1813	2024-08-24 17:27:30.668	-0.37	0.55	0.49
1814	2024-08-24 17:27:32.669	-0.73	2.69	0.49
1815	2024-08-24 17:27:35.132	0.43	0.79	0.49
1816	2024-08-24 17:27:36.683	0.12	0.98	0.55
1817	2024-08-24 17:27:38.684	0.18	0.24	0.49

[1818 rows x 4 columns]

[27]:
```
mongo_df.set_index("Timestamp", inplace=True)
mongo_filter_df = mongo_df.resample('30S').mean()
```

[30]:
```
# Plotting
plt.figure(figsize=(10, 6))
sns.lineplot(data = mongo_filter_df , x = "Timestamp" , y = "x" , label = 'X')
sns.lineplot(data = mongo_filter_df , x = "Timestamp" , y = "y", label = 'Y')
sns.lineplot(data = mongo_filter_df , x = "Timestamp" , y = "z", label = 'Z')
plt.legend(title = 'Axis')
```

[30]: <matplotlib.legend.Legend at 0x1eb1162c910>



As we can see fron the generated graph, the X variable in the gyroscope data shows the most noticeable change at a specific point in time, while for the rest of the period, its values remain relatively consistent. Just like the X variable, both the Y and Z variables follow the same general trend as X, with fluctuations at certain points, but their values tend to stay stable throughout the observation period. However, unlike X, the values for Y and Z tend to be larger than X's overall.

CoachDB

[50]:
```
# MQTT settings
mqtt_broker = "7a783a7c2ba249cc8ca373ec1c6ba990.s1.eu.hivemq.cloud"
mqtt_topic = "Gyro_Data_CoachDB"
mqtt_user = "Ntkt05"
mqtt_password = "abce123@@HH"


# CouchDB configuration
COUCHDB_URL = "http://localhost:5984/gyrocoachdb"  # Replace with your CouchDB␣
 ↪URL and database name
COUCHDB_USERNAME = "admin"
COUCHDB_PASSWORD = "admin"

# Define the MQTT callback function
def on_message(client, userdata, message):
    # Decode the MQTT message payload
    payload = message.payload.decode('utf-8')

    # Convert the payload to a Python dictionary
    var = payload.split(",")
    doc = {
        "Timestamp" : datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S'),
        "x": var[0],
        "y": var[1],
        "z": var[2]
    }

    # Send the data to CouchDB
    response = requests.post(
        COUCHDB_URL,
        headers={"Content-Type": "application/json"},
        data=json.dumps(doc),
        auth=HTTPBasicAuth(COUCHDB_USERNAME, COUCHDB_PASSWORD)
    )

    if response.status_code == 201:
        print("Data successfully saved to CouchDB")
```

```
    else:
        print(f"Failed to save data: {response.text}")

        # setting callbacks for different events to see if it works, print the␣
 ↪message etc.
def on_connect(client, userdata, flags, rc, properties=None):
    print("CONNACK received with code %s." % rc)

# with this callback you can see if your publish was successful
def on_publish(client, userdata, mid, properties=None):
    print("mid: " + str(mid))

# print which topic was subscribed to
def on_subscribe(client, userdata, mid, granted_qos, properties=None):
    print("Subscribed: " + str(mid) + " " + str(granted_qos))
# client_id is the given name of the client
client = paho.Client(client_id="5.2D", userdata=None, protocol=paho.MQTTv5)
client.on_connect = on_connect

# enable TLS for secure connection
client.tls_set(tls_version=mqtt.client.ssl.PROTOCOL_TLS)
# set username and password
client.username_pw_set(mqtt_user, mqtt_password)
# connect to HiveMQ Cloud on port 8883 (default for MQTT)
client.connect(mqtt_broker, 8883)

# setting callbacks, use separate functions like above for better visibility
client.on_subscribe = on_subscribe
client.on_message = on_message
client.on_publish = on_publish

# subscribe to all topics of encyclopedia by using the wildcard "#"
client.subscribe("Gyro_Data_CouchDB", qos = 1)

client.loop_forever()
```

```
C:\Users\thinh\AppData\Local\Temp\ipykernel_24632\3418206977.py:52:
DeprecationWarning: Callback API version 1 is deprecated, update to latest
version
  client = paho.Client(client_id="5.2D", userdata=None, protocol=paho.MQTTv5)
```

```
CONNACK received with code Success.
Subscribed: 1 [ReasonCode(Suback, 'Granted QoS 1')]
Data successfully saved to CouchDB
Data successfully saved to CouchDB
Data successfully saved to CouchDB
Data successfully saved to CouchDB
Data successfully saved to CouchDB
```

```
Data successfully saved to CouchDB
Data successfully saved to CouchDB
Data successfully saved to CouchDB
Data successfully saved to CouchDB
Data successfully saved to CouchDB
Data successfully saved to CouchDB
Data successfully saved to CouchDB
Data successfully saved to CouchDB
Data successfully saved to CouchDB
```

```
---------------------------------------------------------------------------
ConnectionRefusedError                    Traceback (most recent call last)
File ~\Documents\ANACONDA\Lib\site-packages\urllib3\util\connection.py:85, in␣
 ↪create_connection(address, timeout, source_address, socket_options)
     84     sock.bind(source_address)
---> 85 sock.connect(sa)
     86 return sock

ConnectionRefusedError: [WinError 10061] No connection could be made because th␣
 ↪target machine actively refused it

During handling of the above exception, another exception occurred:

KeyboardInterrupt                         Traceback (most recent call last)
Cell In[50], line 70
     67 # subscribe to all topics of encyclopedia by using the wildcard "#"
     68 client.subscribe("Gyro_Data_CouchDB", qos = 1)
---> 70 client.loop_forever()

File ~\Documents\ANACONDA\Lib\site-packages\paho\mqtt\client.py:2297, in Client
 ↪loop_forever(self, timeout, retry_first_connection)
   2295 rc = MQTTErrorCode.MQTT_ERR_SUCCESS
   2296 while rc == MQTTErrorCode.MQTT_ERR_SUCCESS:
-> 2297     rc = self._loop(timeout)
   2298     # We don't need to worry about locking here, because we've
   2299     # either called loop_forever() when in single threaded mode, or
   2300     # in multi threaded mode when loop_stop() has been called and
   2301     # so no other threads can access _out_packet or _messages.
   2302     if (self._thread_terminate is True
   2303         and len(self._out_packet) == 0
   2304             and len(self._out_messages) == 0):

File ~\Documents\ANACONDA\Lib\site-packages\paho\mqtt\client.py:1686, in Client
 ↪_loop(self, timeout)
   1683     return MQTTErrorCode.MQTT_ERR_UNKNOWN
   1685 if self._sock in socklist[0] or pending_bytes > 0:
-> 1686     rc = self.loop_read()
```

```
   1687     if rc or self._sock is None:
   1688         return rc

File ~\Documents\ANACONDA\Lib\site-packages\paho\mqtt\client.py:2100, in Client
 ↪loop_read(self, max_packets)
   2098 if self._sock is None:
   2099     return MQTTErrorCode.MQTT_ERR_NO_CONN
-> 2100 rc = self._packet_read()
   2101 if rc > 0:
   2102     return self._loop_rc_handle(rc)

File ~\Documents\ANACONDA\Lib\site-packages\paho\mqtt\client.py:3142, in Client
 ↪_packet_read(self)
   3140 # All data for this packet is read.
   3141 self._in_packet['pos'] = 0
-> 3142 rc = self._packet_handle()
   3144 # Free data and reset values
   3145 self._in_packet = {
   3146     "command": 0,
   3147     "have_remaining": 0,
   (…)
   3153     "pos": 0,
   3154 }

File ~\Documents\ANACONDA\Lib\site-packages\paho\mqtt\client.py:3808, in Client
 ↪_packet_handle(self)
   3806     return self._handle_pubackcomp("PUBCOMP")
   3807 elif cmd == PUBLISH:
-> 3808     return self._handle_publish()
   3809 elif cmd == PUBREC:
   3810     return self._handle_pubrec()

File ~\Documents\ANACONDA\Lib\site-packages\paho\mqtt\client.py:4145, in Client
 ↪_handle_publish(self)
   4143 message.timestamp = time_func()
   4144 if message.qos == 0:
-> 4145     self._handle_on_message(message)
   4146     return MQTTErrorCode.MQTT_ERR_SUCCESS
   4147 elif message.qos == 1:

File ~\Documents\ANACONDA\Lib\site-packages\paho\mqtt\client.py:4501, in Client
 ↪_handle_on_message(self, message)
   4499 with self._in_callback_mutex:
   4500     try:
-> 4501         on_message(self, self._userdata, message)
   4502     except Exception as err:
   4503         self._easy_log(
   4504             MQTT_LOG_ERR, 'Caught exception in on_message: %s', err)
```

```
Cell In[50], line 28, in on_message(client, userdata, message)
     20 doc = {
     21     "Timestamp" : datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')
     22     "x": var[0],
     23     "y": var[1],
     24     "z": var[2]
     25 }
     27 # Send the data to CouchDB
---> 28 response = requests.post(
     29     COUCHDB_URL,
     30     headers={"Content-Type": "application/json"},
     31     data=json.dumps(doc),
     32     auth=HTTPBasicAuth(COUCHDB_USERNAME, COUCHDB_PASSWORD)
     33 )
     35 if response.status_code == 201:
     36     print("Data successfully saved to CouchDB")

File ~\Documents\ANACONDA\Lib\site-packages\requests\api.py:115, in post(url,␣
 ↪data, json, **kwargs)
    103 def post(url, data=None, json=None, **kwargs):
    104     r"""Sends a POST request.
    105
    106     :param url: URL for the new :class:`Request` object.
   (…)
    112     :rtype: requests.Response
    113     """
--> 115     return request("post", url, data=data, json=json, **kwargs)

File ~\Documents\ANACONDA\Lib\site-packages\requests\api.py:59, in␣
 ↪request(method, url, **kwargs)
     55 # By using the 'with' statement we are sure the session is closed, thus␣
 ↪we
     56 # avoid leaving sockets open which can trigger a ResourceWarning in som
     57 # cases, and look like a memory leak in others.
     58 with sessions.Session() as session:
---> 59     return session.request(method=method, url=url, **kwargs)

File ~\Documents\ANACONDA\Lib\site-packages\requests\sessions.py:589, in Sessio
 ↪request(self, method, url, params, data, headers, cookies, files, auth,␣
 ↪timeout, allow_redirects, proxies, hooks, stream, verify, cert, json)
    584 send_kwargs = {
    585     "timeout": timeout,
    586     "allow_redirects": allow_redirects,
    587 }
    588 send_kwargs.update(settings)
--> 589 resp = self.send(prep, **send_kwargs)
    591 return resp
```

```
File ~\Documents\ANACONDA\Lib\site-packages\requests\sessions.py:703, in Sessio
 ↪send(self, request, **kwargs)
    700 start = preferred_clock()
    702 # Send the request
--> 703 r = adapter.send(request, **kwargs)
    705 # Total elapsed time of the request (approximately)
    706 elapsed = preferred_clock() - start

File ~\Documents\ANACONDA\Lib\site-packages\requests\adapters.py:486, in␣
 ↪HTTPAdapter.send(self, request, stream, timeout, verify, cert, proxies)
    483     timeout = TimeoutSauce(connect=timeout, read=timeout)
    485 try:
--> 486     resp = conn.urlopen(
    487         method=request.method,
    488         url=url,
    489         body=request.body,
    490         headers=request.headers,
    491         redirect=False,
    492         assert_same_host=False,
    493         preload_content=False,
    494         decode_content=False,
    495         retries=self.max_retries,
    496         timeout=timeout,
    497         chunked=chunked,
    498     )
    500 except (ProtocolError, OSError) as err:
    501     raise ConnectionError(err, request=request)

File ~\Documents\ANACONDA\Lib\site-packages\urllib3\connectionpool.py:714, in␣
 ↪HTTPConnectionPool.urlopen(self, method, url, body, headers, retries,␣
 ↪redirect, assert_same_host, timeout, pool_timeout, release_conn, chunked,␣
 ↪body_pos, **response_kw)
    711     self._prepare_proxy(conn)
    713 # Make the request on the httplib connection object.
--> 714 httplib_response = self._make_request(
    715     conn,
    716     method,
    717     url,
    718     timeout=timeout_obj,
    719     body=body,
    720     headers=headers,
    721     chunked=chunked,
    722 )
    724 # If we're going to release the connection in ``finally:``, then
    725 # the response doesn't need to know about the connection. Otherwise
    726 # it will also try to release it and we'll have a double-release
    727 # mess.
```

```
    728 response_conn = conn if not release_conn else None

File ~\Documents\ANACONDA\Lib\site-packages\urllib3\connectionpool.py:415, in␣
 ↪HTTPConnectionPool._make_request(self, conn, method, url, timeout, chunked,␣
 ↪**httplib_request_kw)
    413         conn.request_chunked(method, url, **httplib_request_kw)
    414     else:
--> 415         conn.request(method, url, **httplib_request_kw)
    417 # We are swallowing BrokenPipeError (errno.EPIPE) since the server is
    418 # legitimately able to close the connection after sending a valid␣
 ↪response.
    419 # With this behaviour, the received response is still readable.
    420 except BrokenPipeError:
    421     # Python 3

File ~\Documents\ANACONDA\Lib\site-packages\urllib3\connection.py:244, in␣
 ↪HTTPConnection.request(self, method, url, body, headers)
    242 if "user-agent" not in (six.ensure_str(k.lower()) for k in headers):
    243     headers["User-Agent"] = _get_default_user_agent()
--> 244 super(HTTPConnection, self).request(method, url, body=body,␣
 ↪headers=headers)

File ~\Documents\ANACONDA\Lib\http\client.py:1286, in HTTPConnection.
 ↪request(self, method, url, body, headers, encode_chunked)
   1283 def request(self, method, url, body=None, headers={}, *,
   1284             encode_chunked=False):
   1285     """Send a complete request to the server."""
-> 1286     self._send_request(method, url, body, headers, encode_chunked)

File ~\Documents\ANACONDA\Lib\http\client.py:1332, in HTTPConnection.
 ↪_send_request(self, method, url, body, headers, encode_chunked)
   1328 if isinstance(body, str):
   1329     # RFC 2616 Section 3.7.1 says that text default has a
   1330     # default charset of iso-8859-1.
   1331     body = _encode(body, 'body')
-> 1332 self.endheaders(body, encode_chunked=encode_chunked)

File ~\Documents\ANACONDA\Lib\http\client.py:1281, in HTTPConnection.
 ↪endheaders(self, message_body, encode_chunked)
   1279 else:
   1280     raise CannotSendHeader()
-> 1281 self._send_output(message_body, encode_chunked=encode_chunked)

File ~\Documents\ANACONDA\Lib\http\client.py:1041, in HTTPConnection.
 ↪_send_output(self, message_body, encode_chunked)
   1039 msg = b"\r\n".join(self._buffer)
   1040 del self._buffer[:]
-> 1041 self.send(msg)
```

```
   1043 if message_body is not None:
   1044
   1045     # create a consistent interface to message_body
   1046     if hasattr(message_body, 'read'):
   1047         # Let file-like take precedence over byte-like.  This
   1048         # is needed to allow the current position of mmap'ed
   1049         # files to be taken into account.

File ~\Documents\ANACONDA\Lib\http\client.py:979, in HTTPConnection.send(self,␣
 ↪data)
    977 if self.sock is None:
    978     if self.auto_open:
--> 979         self.connect()
    980     else:
    981         raise NotConnected()

File ~\Documents\ANACONDA\Lib\site-packages\urllib3\connection.py:205, in␣
 ↪HTTPConnection.connect(self)
    204 def connect(self):
--> 205     conn = self._new_conn()
    206     self._prepare_conn(conn)

File ~\Documents\ANACONDA\Lib\site-packages\urllib3\connection.py:174, in␣
 ↪HTTPConnection._new_conn(self)
    171     extra_kw["socket_options"] = self.socket_options
    173 try:
--> 174     conn = connection.create_connection(
    175         (self._dns_host, self.port), self.timeout, **extra_kw
    176     )
    178 except SocketTimeout:
    179     raise ConnectTimeoutError(
    180         self,
    181         "Connection to %s timed out. (connect timeout=%s)"
    182         % (self.host, self.timeout),
    183     )

File ~\Documents\ANACONDA\Lib\site-packages\urllib3\util\connection.py:91, in␣
 ↪create_connection(address, timeout, source_address, socket_options)
     89         err = e
     90         if sock is not None:
---> 91             sock.close()
     92             sock = None
     94 if err is not None:

File ~\Documents\ANACONDA\Lib\socket.py:499, in socket.close(self)
    495 def _real_close(self, _ss=_socket.socket):
    496     # This function should not reference any globals. See issue #808164
    497     _ss.close(self)
```

```
--> 499 def close(self):
    500     # This function should not reference any globals. See issue #808164
    501     self._closed = True
    502     if self._io_refs <= 0:

KeyboardInterrupt:
```

[33]:
```
# Fetch data from CouchDB
response = requests.get('http://localhost:5984/gyrocoachdb/_all_docs',
                        params={'include_docs': 'true'},
                        auth=HTTPBasicAuth('admin', 'admin'))

data = response.json()

# Write the data to a CSV file
with open('couchdb.csv', 'w', newline='') as file:
    writer = csv.writer(file)

    # Write header
    writer.writerow(['Timestamp', 'x', 'y', 'z'])

    # Write data rows
    for row in data.get('rows', []):
        doc = row.get('doc', {})
        writer.writerow([doc.get('Timestamp'), doc.get('x'), doc.get('y'), doc.
 ↪get('z')])

print("Data has been written to couchdb.csv")
```

```
Data has been written to couchdb.csv
```

[34]:
```
# Load the CSV file
couch_df = pd.read_csv("couchdb.csv")
couch_df
```

[34]:

	Timestamp	x	y	z
0	2024-08-26 20:10:07	0.31	0.61	0.49
1	2024-08-26 20:10:09	0.12	0.55	0.67
2	2024-08-26 20:10:11	0.61	0.24	0.55
3	2024-08-26 20:10:13	1.10	-0.55	0.55
4	2024-08-26 20:10:15	0.43	0.49	0.55
…	…	…	…	…
2781	2024-08-26 23:05:29	0.24	0.55	0.55
2782	2024-08-26 23:05:31	0.43	0.49	0.49
2783	2024-08-26 23:05:33	0.31	0.55	0.55
2784	2024-08-26 23:05:35	0.31	0.55	0.49
2785	2024-08-26 23:05:37	0.24	0.55	0.49

```
[2786 rows x 4 columns]
```

[36]:
```
# Preprocessing the Data
couch_df["Timestamp"] = pd.to_datetime(couch_df["Timestamp"], errors='coerce')
couch_df
```

[36]:

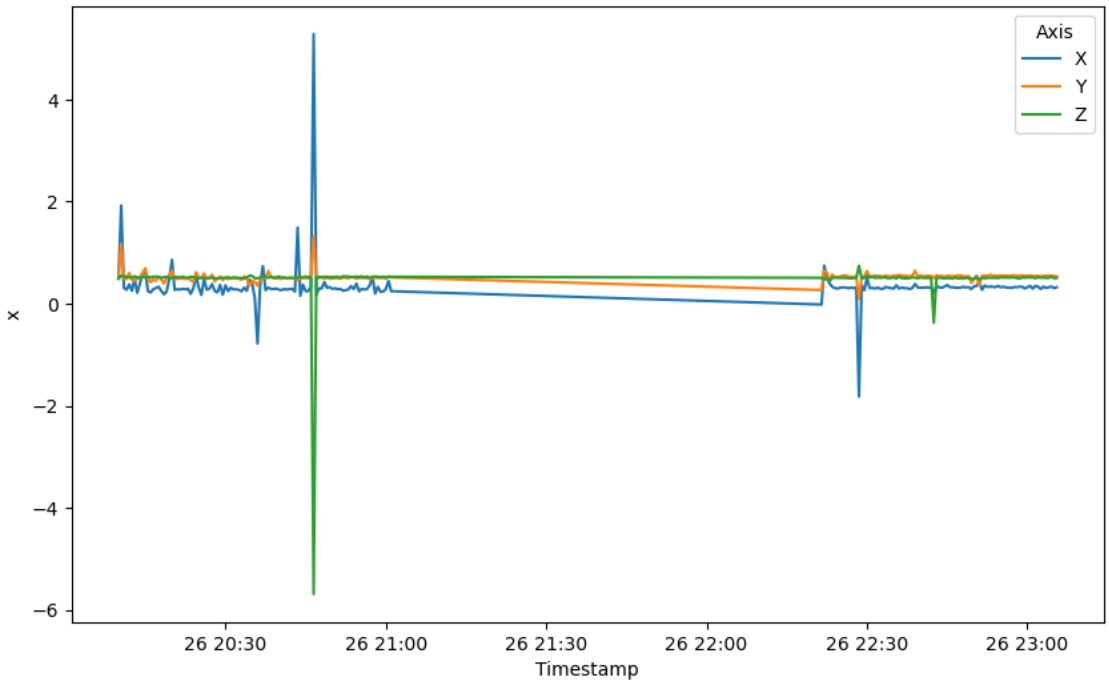
	Timestamp	x	y	z
0	2024-08-26 20:10:07	0.31	0.61	0.49
1	2024-08-26 20:10:09	0.12	0.55	0.67
2	2024-08-26 20:10:11	0.61	0.24	0.55
3	2024-08-26 20:10:13	1.10	-0.55	0.55
4	2024-08-26 20:10:15	0.43	0.49	0.55
…	…	…	…	…
2781	2024-08-26 23:05:29	0.24	0.55	0.55
2782	2024-08-26 23:05:31	0.43	0.49	0.49
2783	2024-08-26 23:05:33	0.31	0.55	0.55
2784	2024-08-26 23:05:35	0.31	0.55	0.49
2785	2024-08-26 23:05:37	0.24	0.55	0.49

```
[2786 rows x 4 columns]
```

[38]:
```
couch_df.set_index("Timestamp", inplace=True)
couch_filter_df = couch_df.resample('30S').mean()
```

[39]:
```
# Plotting
plt.figure(figsize=(10, 6))
sns.lineplot(data = couch_filter_df , x = "Timestamp" , y = "x" , label = 'X')
sns.lineplot(data = couch_filter_df , x = "Timestamp" , y = "y", label = 'Y')
sns.lineplot(data = couch_filter_df , x = "Timestamp" , y = "z", label = 'Z')
plt.legend(title = 'Axis')
```

[39]:
```
<matplotlib.legend.Legend at 0x1eb13b9d050>
```



As expected, the results from the CouchDB dataset are closely the same from those we observed with the MongoDB dataframe in terms of the data trends for the X, Y, and Z variables. The pattern of movement and variation in each axis remains consistent, indicating similar sensor behavior and data capture across both databases

Overall, the X variable saw the most fluctuation, while Y and Z show relatively stable behavior over the period of time. The results reinforce the reliability of the sensor data in capturing motion across different storage systems.

TASK 5.2D – SIT225

TRUONG KHANG THINH NGUYEN

For the task 5.2D, I used MongoDB for cloud-based database and CouchDB for local-based database as two alternatives to Firebase since I want to have different perspectives in terms of storing in local or cloud.

COMPARISON OF DIFFERENT DATABASES

-Firebase

Firebase offers a faster setup method and a fully managed backend. Configuration is really simple, especially when integrating with mobile and web applications, thanks to the full SDKs and detailed documentation.

Firebase offers two main data storage solutions: Realtime Database and Firestore. The Realtime Database is designed for realtime data synchronization, whereas Firestore supports more complex querying, including hierarchical data structures.

In terms of ease of use, Firebase is specifically intended for straightforward and simple usage with an emphasis on rapid development and deployment. Its real-time capabilities and built- in services, such as authentication and cloud operations, make it ideal for developer use, particularly in mobile app development.

-MongoDB

MongoDB, whether self-hosted or managed through MongoDB Atlas, provides greater flexibility and control over configuration. It allows a wide number of deployment options and configurations, however this flexibility may result in increased setup and administrative complexity. But for this task, I chose to implement it in cloud.

MongoDB stores data in BSON documents, which allow for sophisticated searching and indexing capabilities. It allows complicated searches, aggregations, and indexing, making it ideal for applications that require extensive and diversified data operations.

Regarding the difficulty of using it, MongoDB is versatile, but it may require more setup than Firebase, particularly for unique settings. Its broad driver support and documentation make integration easier, but the learning curve is harder for beginners compared to Firebase's more user-friendly approach.

-CouchDB

CouchDB's RESTful HTTP API makes configuration relatively simple. Its emphasis on a schema-free document architecture and ease of replication across distributed contexts make it ideal for applications that require **offline**-first functionality.

CouchDB stores data in JSON documents and queries via MapReduce. Its RESTful interface allows for easy interaction and integration with other online services. However, its querying capabilities are less advanced than MongoDB's.

CouchDB's offline-first architecture and straightforward REST API make it simple to deal with, particularly for dispersed applications. However, its querying and indexing are less sophisticated than MongoDB's, which may limit its ability to handle more complex data operations.

To conclude, Firebase is really good at in terms of usability and integration, making it perfect for rapid development and real-time apps. MongoDB provides strong querying capabilities and flexibility, making it ideal for complicated data requirements. CouchDB offers simplicity and offline capabilities, but only basic querying functions. Each offers advantages depending on the specific requirements of the application.

COMPARISON BETWEEN MQTT AND SERIAL COMMUNICATION

With respect to data transfer, Serial communication is ideal for direct, low-speed data transfer over short distances, which is commonly employed in embedded systems or connecting microcontrollers to sensors. MQTT, on the other hand, is intended for scalable, efficient messaging across networks, making it ideal for IoT and cloud-based applications.

Talking about the parsing and management, Serial communication requires manual parsing and manipulation of data frames, which can be difficult and error prone. MQTT abstracts these issues by providing built-in support for message routing, delivery guarantees, and topic-based subscription, making it easier for developers to conduct their tasks.

And for applications, Serial communication is suitable for direct device-to-device communication in limited settings. MQTT is appropriate for distributed systems, where devices must communicate over a network under changing conditions and message dependability and scalability are critical.

In conclusion, MQTT is generally more efficient for networked applications due to its lightweight design and support for several QoS levels, whereas serial communication's performance is restricted by its simplicity and physical limits. It really depends on the specific requirements of the current applications.