Student name: Truong Khang Thinh Nguyen

Student ID: 223446545

SIT225: Data Capture Technologies

Activity 8.1: Using smartphone to capture sensor data

The **Arduino IoT Remote** phone application lets you control and monitor all of your dashboards in the Arduino Cloud. With the app, you can also access your phone's internal sensors such as GPS data, light sensor, IMU and more (depending on what phone you have).

The phone's sensor data is automatically stored in Cloud variables, which you can also synchronize with other Things such as custom thing in Python board. This means your phone can become a part of your IoT system, acting as another node in your network.

In this activity, you will enable your smartphone to work as a custom device (like an Arduino board) and connect to your smartphone sensors such as accelerometers and GPS and streaming data to Arduino IoT Cloud dashboard.

Hardware Required

Your smartphone – compatible Android or iPhone

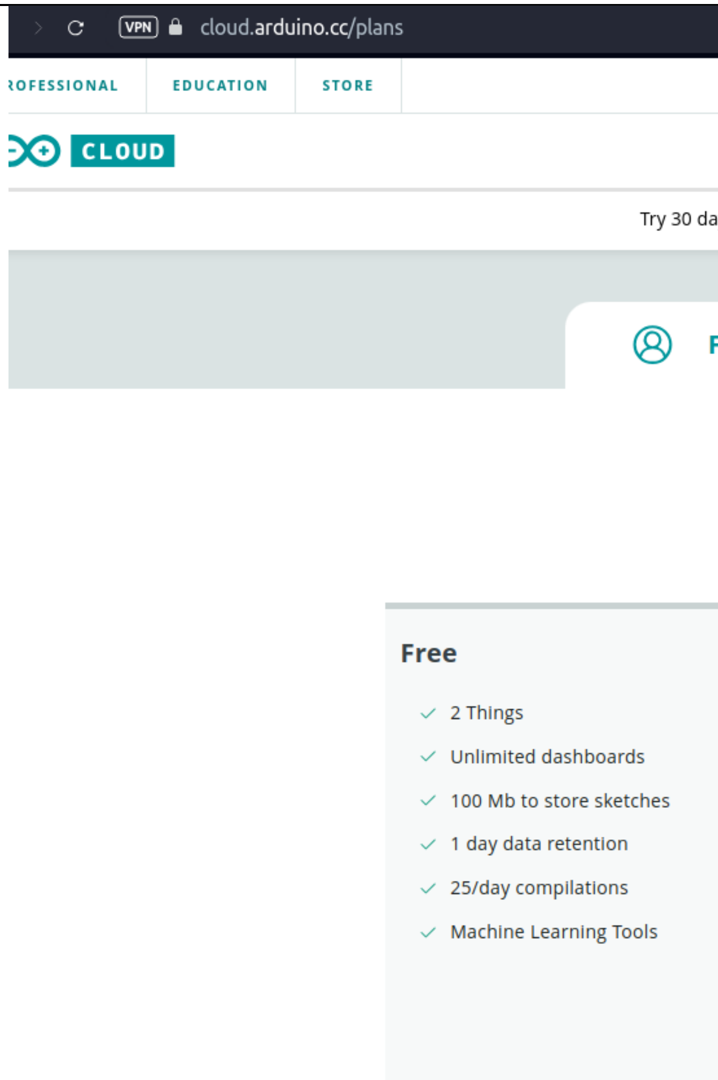
NOTE: *The IoT Remote app requires iOS 12.4 or later for iOS the version. If you are using Android, version 8.0 or later is required. Make sure the iOS or Android version on your device is up to date before downloading the app.*

Software Required

Android / iOS smart phone.
Arduino account
Arduino IoT Remote App (App Store or Google Play)
Python 3 (for custom Python Thing)

Steps

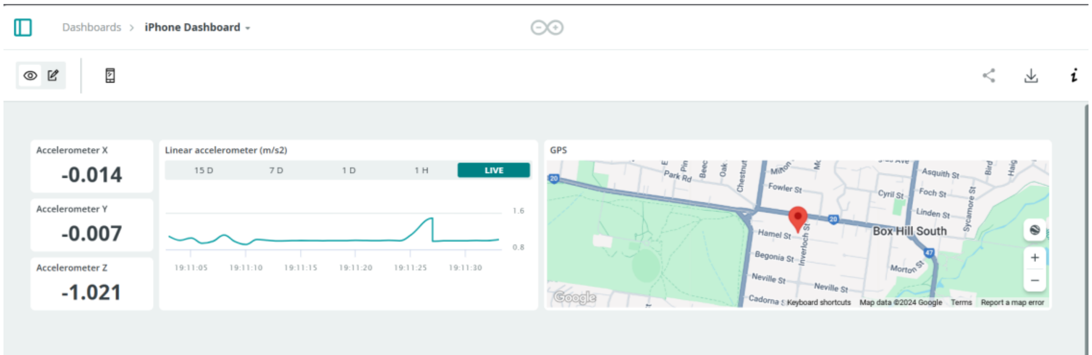
Step	Action
1	**Install App**: To use the Arduino IoT Remote app, visit Google Play / App Store and search for "Arduino IoT Remote". After installing the app, you will need to log in to your Arduino account. After you login, you will discover all your dashboards (if you have any), in the main menu. Based on the app version, home screen may vary. There will be 3 tabs at the bottom – Dashboards, Devices and Activity. You can follow the tutorial (https://docs.arduino.cc/arduino-cloud/iot-remote-app/getting-started).
2	**Add device**: Tap into the Devices tab. You will be able to create a new device. Alternatively, you can your profile (top right corner), in the settings section, you will see “Phone as device” which you can turn ON if it is OFF. There, you can select sensors in your smartphone such as accelerometer linear, accelerometer x/y/z and GPS among others. Note: A free account is enough for this experiment. If you are asked to upgrade your account, you can remove all other Things from your Arduino IoT Cloud account since the Free account allows at most 2 Things to configure, see below image.

	 The add device wizard will allow you to setup your sensor and also create a dashboard which you can see in your smartphone app. If you login to Arduino IoT Cloud in web browser, you can see the dashboard for your smartphone is already created.
3	**Keep your smartphone screen ON for a while**: Keep data coming through your smartphone for 10-15 minutes. During this time, keep moving your smartphone in a pattern so the accelerometer data can be analysed to discover the pattern. You can download data from the Ardiuno Cloud dashboard page by clicking on a download icon at top right corner which shows – Download historic data. A data download link will be sent to your account email from there you can download data.

<table>
<tr>
<td></td>
<td>

Question: Take a screenshot of your Arduino Cloud Dashboard where smartphone data is streaming and paste it here.

Answer: <Your answer>

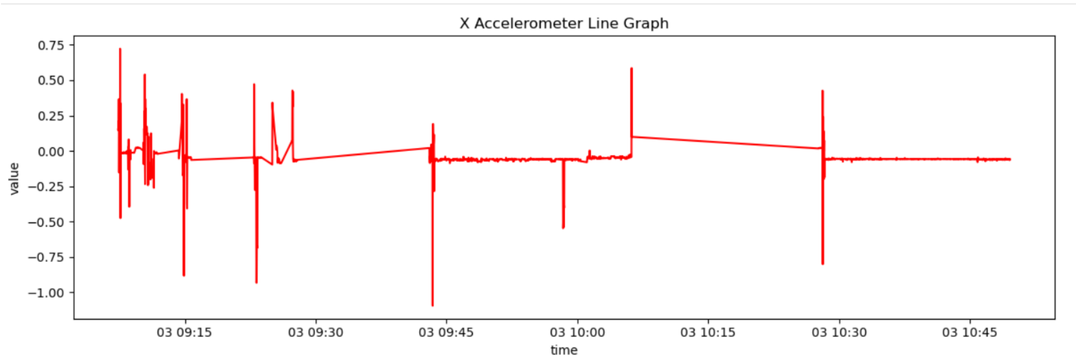


</td>
</tr>
<tr>
<td>4</td>
<td>

Plot accelerometer data:
The zipped data file you downloaded from the cloud contains separate files per variable including accelerometer_linear, accelerometer_x, accelerometer_y, accelerometer_z and Gps. Each file has 2 columns – time and value.

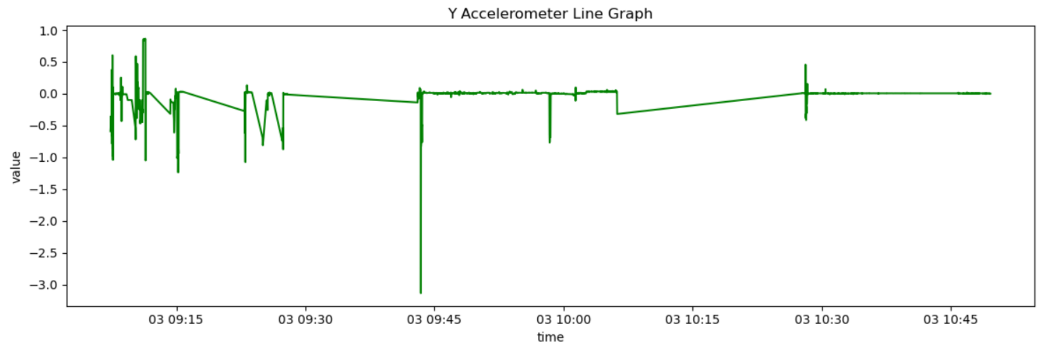
Question: Open Jupyter Notebook by using command line, go to the data folder and write command ($ jupyter lab). Using Pandas, read CSV file and fetch the data column for accelerometer_x and plot it using Python plotting library (matplotlib or any other convenient for you). Repeat the plotting process for accelerometer y and z to have 3 separate graphs. Now create a fourth graph with all 3 variables x, y and z. Screenshot the 4 graphs and paste here.

Answer: <Your answer>
X Accelerometer

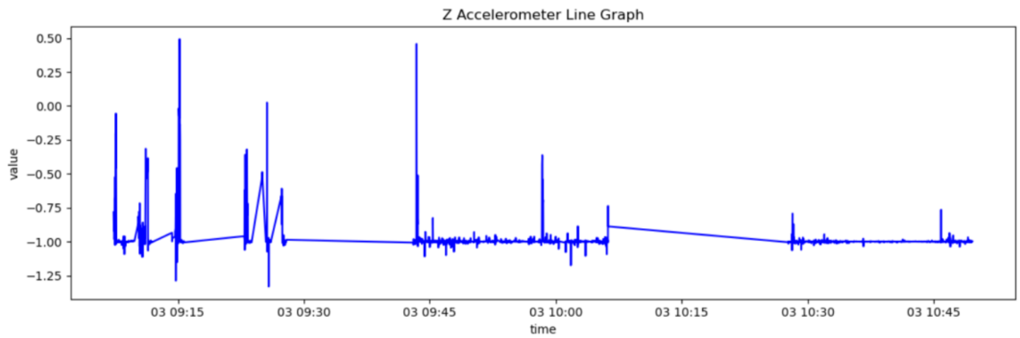


Y Accelerometer

</td>
</tr>
</table>

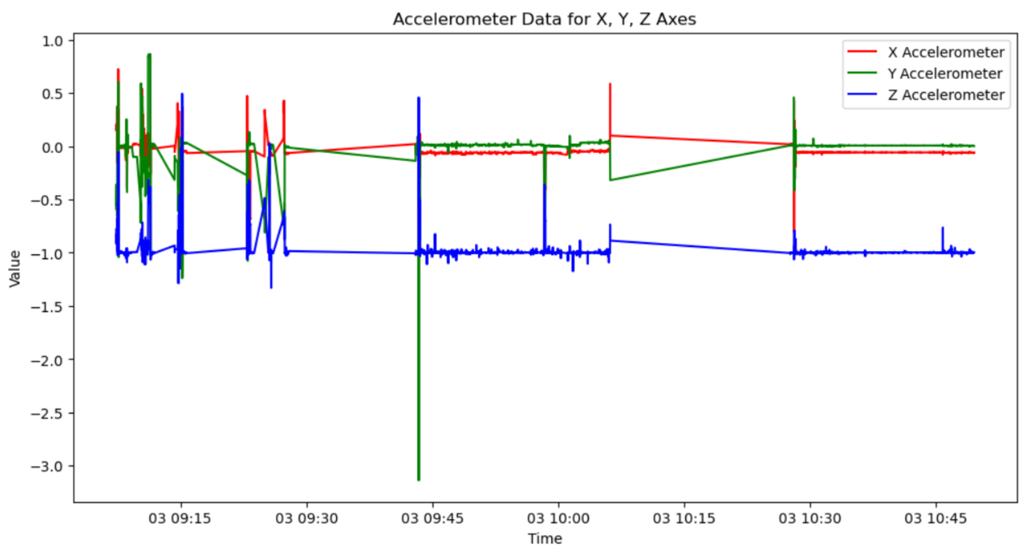
<table>
<tr>
<td></td>
<td>


Z Accelerometer



</td>
</tr>
<tr>
<td>5</td>
<td>
Question: Analyse accelerometer variables to find any repeating pattern. Remember that you were repeatedly moving your phone in a single pattern which should be manifested in the graphs. Justify your answer.

Answer: <Your answer>



As expectation, because I put the phone lying flat on its back , screen facing upward so the X accelerometer and Y accelerometer values remain at 0.0 and Z accelerometer values remain at -1.0
</td>
</tr>
</table>

Activity 8.2: Receive smartphone sensor data from Python script

You can connect anything to Arduino Cloud including a wide range of compatible Arduino boards such as Arduino Nano 33 IoT or a third-party device that speaks Python. In activity 3.2, you have configured custom Python board and created a cloud variable that was synced to your Arduino Thing such as DHT22 sensor variables.

In this activity, you will need to synchronise smartphone's accelerometer x, y and z variables to Python script. If you can recall, you have already done a similar function in Activity 3.2.

Steps:

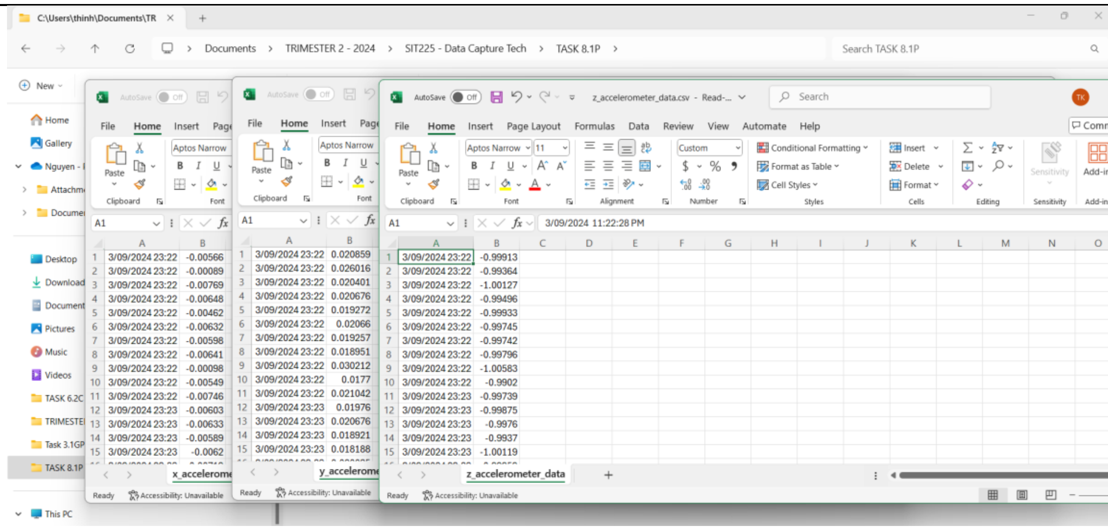
Step	Action
1	Configure Python board in Arduino Cloud and create a Thing where define 3 variables at a time and sync to corresponding accelerometer variable of smartphone Thing.
2	Write Python script to keep listening to data from the 3 variables to come through. You may need to create 3 call-back functions – a single function per variable (x, y and z).
3	Question: Keep storing each variable data in a separate file. Append each value with a timestamp so each data reading forms a comma separated line - <timestamp>, <data-value>. New data is written in a separate line. Keep storing them in a CSV file, where there will be 3 separate files. Screenshot your Python script here and screenshot the files opened side-by-side you have created and paste it here. Answer: <your answer>.

Activity_8.2.py > on_z_acc_changed

```python
import sys
import traceback
from arduino_iot_cloud import ArduinoCloudClient
import asyncio
from datetime import datetime
import time

DEVICE_ID = "d2d8a108-d4c3-4794-bf15-271885269210"
SECRET_KEY = "YyuZ6b3CQZHTJC#TGX@@H7AiP"

# Open files for each variable in append mode at the start
x_file = open('x_accelerometer_data.csv', 'a')
y_file = open('y_accelerometer_data.csv', 'a')
z_file = open('z_accelerometer_data.csv', 'a')

# Callback function on x_acc variable change event
def on_x_acc_changed(client, value):
    timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    csv_string = f"{timestamp}, {value}\n"
    x_file.write(csv_string)
    x_file.flush()
    print(f"New X accelerometer data logged: {csv_string.strip()}")


# Callback function on y_acc variable change event
def on_y_acc_changed(client, value):
    timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    csv_string = f"{timestamp}, {value}\n"
    y_file.write(csv_string)
    y_file.flush()
    print(f"New Y accelerometer data logged: {csv_string.strip()}")


# Callback function on z_acc variable change event
def on_z_acc_changed(client, value):
    timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    csv_string = f"{timestamp}, {value}\n"
    z_file.write(csv_string)
    z_file.flush()
    print(f"New Z accelerometer data logged: {csv_string.strip()}")

def main():
    print("main() function")
    try:
        # Instantiate Arduino cloud client
        client = ArduinoCloudClient(
            device_id=DEVICE_ID, username=DEVICE_ID, password=SECRET_KEY
        )
```

```python
        )

        # Register with 'x_acc', 'y_acc', and 'z_acc' cloud variables
        # and listen to their value changes in corresponding callback functions
        client.register(
            "x", value=None,
            on_write=on_x_acc_changed)

        client.register(
            "y", value=None,
            on_write=on_y_acc_changed)

        client.register(
            "z", value=None,
            on_write=on_z_acc_changed)

        # Start the cloud client
        client.start()

        # Keep the script running for 10 minutes to log data
        time.sleep(600)

    except Exception as e:
        print(f"Exception occurred: {e}")
        traceback.print_exc()

    finally:
        # Close all files at the end of the execution
        x_file.close()
        y_file.close()
        z_file.close()

if __name__ == "__main__":
    try:
        main()  # main function which runs in an internal infinite loop
    except:
        exc_type, exc_value, exc_traceback = sys.exc_info()
        traceback.print_tb(exc_type, file=print)

```

X , y and z respectively



Terminal Output

```
New X accelerometer data logged: 2024-09-03 23:22:28, -0.0056610107421875
New Z accelerometer data logged: 2024-09-03 23:22:28, -0.9991302490234375
New Y accelerometer data logged: 2024-09-03 23:22:28, 0.0208587646484375
New X accelerometer data logged: 2024-09-03 23:22:30, -0.000885009765625
New Y accelerometer data logged: 2024-09-03 23:22:31, 0.0260162353515625
New Z accelerometer data logged: 2024-09-03 23:22:32, -0.9936370849609375
New X accelerometer data logged: 2024-09-03 23:22:33, -0.0076904296875
New Y accelerometer data logged: 2024-09-03 23:22:34, 0.0204010009765625
New Z accelerometer data logged: 2024-09-03 23:22:35, -1.0012664794921875
New X accelerometer data logged: 2024-09-03 23:22:36, -0.0064849853515625
New Y accelerometer data logged: 2024-09-03 23:22:37, 0.0206756591796875
New Z accelerometer data logged: 2024-09-03 23:22:38, -0.994964599609375
New X accelerometer data logged: 2024-09-03 23:22:39, -0.0046234130859375
```

4

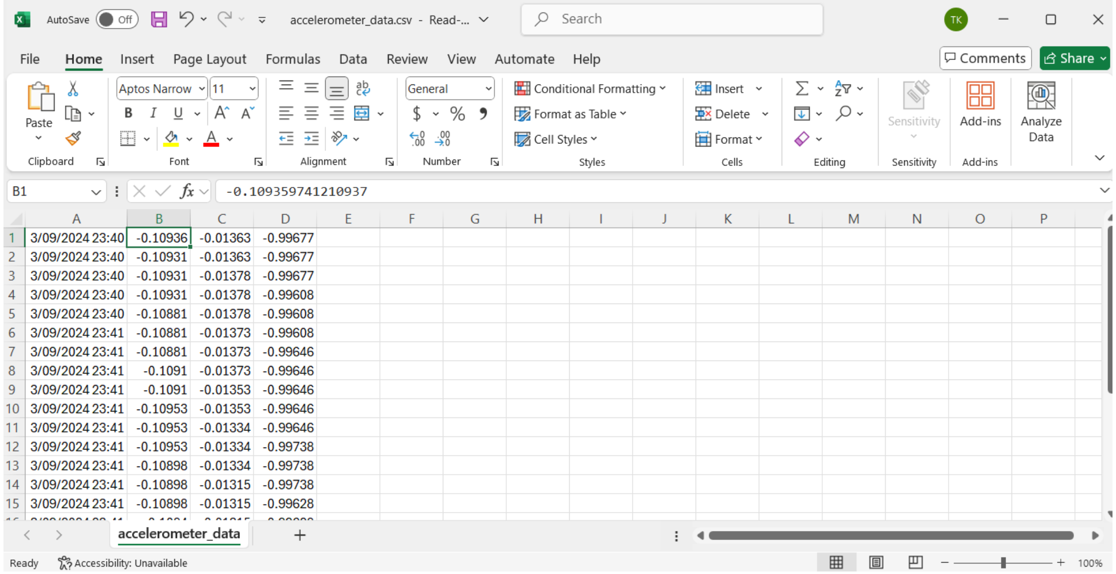
Question: Now manage 3 variable data so they can be stored in a single CSV file where each line consists of comma separated sensor values with a timestamp - <timestamp>, <x>, <y>, <z>. Store data once you gather 3 variables and repeat the process. Screenshot your Python script here and screenshot the file you have created opened and paste it here.

Answer: <your answer>.

Activity_8.2.py > main

```python
import sys
import traceback
from arduino_iot_cloud import ArduinoCloudClient
import asyncio
from datetime import datetime
import time

DEVICE_ID = "d2d8a108-d4c3-4794-bf15-271885269210"
SECRET_KEY = "YyuZ6b3CQZHTJC#TGX@@H7AiP"

# Open a single file for all variables in append mode
file = open('accelerometer_data.csv', 'a')

# Dictionary to store the latest values of x, y, z
latest_values = {
    'x': None,
    'y': None,
    'z': None
}

def write_to_file():
    timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    csv_string = f"{timestamp}, {latest_values['x']}, {latest_values['y']}, {latest_values['z']}\n"
    file.write(csv_string)
    file.flush()
    print(f"Data logged: {csv_string.strip()}")

# Callback function on x_acc variable change event
def on_x_acc_changed(client, value):
    latest_values['x'] = value
    if all(v is not None for v in latest_values.values()):
        write_to_file()

# Callback function on y_acc variable change event
def on_y_acc_changed(client, value):
    latest_values['y'] = value
    if all(v is not None for v in latest_values.values()):
        write_to_file()

# Callback function on z_acc variable change event
def on_z_acc_changed(client, value):
    latest_values['z'] = value
    if all(v is not None for v in latest_values.values()):
        write_to_file()
```

```python
async def main():
    print("main() function")
    try:
        # Instantiate Arduino cloud client
        client = ArduinoCloudClient(
            device_id=DEVICE_ID, username=DEVICE_ID, password=SECRET_KEY
        )

        # Register with 'x_acc', 'y_acc', and 'z_acc' cloud variables
        # and listen to their value changes in corresponding callback functions
        client.register("x", value=None, on_write=on_x_acc_changed)
        client.register("y", value=None, on_write=on_y_acc_changed)
        client.register("z", value=None, on_write=on_z_acc_changed)

        # Start the cloud client asynchronously with interval and backoff
        interval = 10  # Polling interval in seconds
        backoff = 2     # Backoff factor for retrying connection
        await client.run(interval=interval, backoff=backoff)

        # Keep the script running for 10 minutes to log data
        await asyncio.sleep(600)

    except Exception as e:
        print(f"Exception occurred: {e}")
        traceback.print_exc()

    finally:
        # Close the file at the end of the execution
        file.close()

if __name__ == "__main__":
    try:
        # Run the main() asynchronously
        asyncio.run(main())
    except:
        exc_type, exc_value, exc_traceback = sys.exc_info()
        traceback.print_tb(exc_type, file=print)
```



	A	B	C	D
1	3/09/2024 23:40	-0.10936	-0.01363	-0.99677
2	3/09/2024 23:40	-0.10931	-0.01363	-0.99677
3	3/09/2024 23:40	-0.10931	-0.01378	-0.99677
4	3/09/2024 23:40	-0.10931	-0.01378	-0.99608
5	3/09/2024 23:40	-0.10881	-0.01378	-0.99608
6	3/09/2024 23:41	-0.10881	-0.01373	-0.99608
7	3/09/2024 23:41	-0.10881	-0.01373	-0.99646
8	3/09/2024 23:41	-0.1091	-0.01373	-0.99646
9	3/09/2024 23:41	-0.1091	-0.01353	-0.99646
10	3/09/2024 23:41	-0.10953	-0.01353	-0.99646
11	3/09/2024 23:41	-0.10953	-0.01334	-0.99646
12	3/09/2024 23:41	-0.10953	-0.01334	-0.99738
13	3/09/2024 23:41	-0.10898	-0.01334	-0.99738
14	3/09/2024 23:41	-0.10898	-0.01315	-0.99738
15	3/09/2024 23:41	-0.10898	-0.01315	-0.99628

Terminal Output

```
Data logged: 2024-09-03 23:43:01, -0.1106414794921875, -0.0130615234375, -0.9967498779296875
Data logged: 2024-09-03 23:43:02, -0.1106414794921875, -0.0130615234375, -0.99725341796875
Data logged: 2024-09-03 23:43:03, -0.1109619140625, -0.0130615234375, -0.99725341796875
Data logged: 2024-09-03 23:43:03, -0.1109619140625, -0.0119476318359375, -0.99725341796875
Data logged: 2024-09-03 23:43:05, -0.1109619140625, -0.0119476318359375, -0.99664306640625
Data logged: 2024-09-03 23:43:06, -0.110626220703125, -0.0119476318359375, -0.99664306640625
Data logged: 2024-09-03 23:43:07, -0.110626220703125, -0.0128326416015625, -0.99664306640625
Data logged: 2024-09-03 23:43:08, -0.110626220703125, -0.0128326416015625, -0.9952239990234375
Data logged: 2024-09-03 23:43:09, -0.110076904296875, -0.0128326416015625, -0.9952239990234375
```

task-objective-8-1p

September 5, 2024

[107]:
```
import sys
import traceback
from arduino_iot_cloud import ArduinoCloudClient
import asyncio
from datetime import datetime

import plotly.express as px
import seaborn as sns
import matplotlib.pyplot as plt
import dash
from dash.dependencies import Input, Output
from dash import dcc, html
import time
```

[108]:
```
DEVICE_ID = "d2d8a108-d4c3-4794-bf15-271885269210"
SECRET_KEY = "YyuZ6b3CQZHTJC#TGX@@H7AiP"
```

[109]:
```
arduino = ArduinoCloudClient(
            device_id=DEVICE_ID, username=DEVICE_ID, password=SECRET_KEY
        )
```

[110]:
```
# 2 buffer data one for temporary and the one for plotting
buffer_data = []
plot_data = []
```

[111]:
```
# Plotly dash
# Initialize the Dash app
app = dash.Dash(__name__)

# Define the layout of the app
app.layout = html.Div([
    dcc.Graph(id='update-graph'),
    dcc.Interval(id='interval-component', interval=1000, n_intervals=0)
])

@app.callback(
    Output('update-graph', 'figure'),
```

```
    Input('interval-component', 'n_intervals')
)
def update_graph(n_intervals):
    global plot_data
    # Convert the current data to a DataFrame
    df = pd.DataFrame(plot_data, columns=['index', 'Timestamp', 'X', 'Y', 'Z'])

    # Create a line plot with Plotly Express
    figure = px.line(df, x='Timestamp', y=['X', 'Y', 'Z'], title='Accelerometer␣
↪Data using SmartPhone')

    return figure
```

[112]:
```
app.run_server(debug_mode = True, jupyter_tab = True)
```

<IPython.lib.display.IFrame at 0x180e214d410>

[151]:
```
x, y, z = 0, 0, 0
time = 0
num_threshold = 10

def on_x_changed(client, value):
    global x
    x = value
def on_y_changed(client, value):
    global y
    y = value
def on_z_changed(client, value):
    global z
    z = value
if __name__ == "__main__":
    client = ArduinoCloudClient(device_id=DEVICE_ID, username=DEVICE_ID,
                                password=SECRET_KEY, sync_mode = True)
    # Register the callback functions
    client.register("x", value=None, on_write=on_x_changed)
    client.register("y", value=None, on_write=on_y_changed)
    client.register("z", value=None, on_write=on_z_changed)
    client.start()

    while True:
        # Check if all the variables x y and z are all recorded
        if (x is not None ) and (y is not None) and (z is  not None) :
            if time < num_threshold :

                time = time + 1
                current_timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
                buffer_data.append([time, current_timestamp, x,y,z])
```

```
                # Print out to check the number of times the data has been␣
↪recorded
                print([time,current_timestamp,x,y,z])

                # Set x,y,z back to None
                x, y, z = None, None, None
            else :
                current_timestamp = datetime.now()
                format_time = current_timestamp.strftime('%Y-%m-%d-%H-%M-%S')
                df = pd.DataFrame(buffer_data,
                                  columns=['Index','Timestamp', 'X_Acce',␣
↪'Y_Acce', 'Z_Acce'])
                filename = f"csv_{format_time}.csv"
                df.to_csv(filename)

                # Reset the time to 0 becasue it has reached the defined␣
↪threshold
                time = 0
                plot_data = buffer_data.copy()

                # Clear the buffer data for the next time usage
                buffer_data.clear()
        client.update()
```

```
[1, '2024-09-05 13:36:39', 0, 0, 0]
[2, '2024-09-05 13:36:39', 0.0134735107421875, 0.0346221923828125,
-1.00335693359375]
[3, '2024-09-05 13:36:40', 0.0043182373046875, 0.05413818359375,
-0.999053955078125]
[4, '2024-09-05 13:36:41', -0.0029296875, 0.054351806640625, -0.9996337890625]
[5, '2024-09-05 13:36:42', 0.0146484375, 0.0453948974609375,
-0.9985504150390625]
[6, '2024-09-05 13:36:43', 0.0056610107421875, 0.04461669921875,
-0.9996490478515625]
[7, '2024-09-05 13:36:44', 0.0093841552734375, 0.040740966796875,
-0.9973907470703125]
[8, '2024-09-05 13:36:45', 0.00830078125, 0.0403289794921875,
-0.9998626708984375]
[9, '2024-09-05 13:36:46', 0.4925079345703125, 0.1922760009765625,
-0.8449249267578125]
[10, '2024-09-05 13:36:47', 0.3591156005859375, 0.2521820068359375,
-0.899139404296875]
[1, '2024-09-05 13:36:48', 0.36370849609375, 0.2473602294921875,
-0.8879547119140625]
[2, '2024-09-05 13:36:49', 0.369415283203125, 0.283966064453125,
-0.875457763671875]
[3, '2024-09-05 13:36:50', 0.3466796875, 0.2802886962890625,
```

-0.8928985595703125]
[4, '2024-09-05 13:36:51', 0.35430908203125, 0.2727813720703125,
-0.9009246826171875]
[5, '2024-09-05 13:36:52', 0.32342529296875, 0.273773193359375,
-0.8963470458984375]
[6, '2024-09-05 13:36:53', 0.336669921875, 0.2750701904296875,
-0.9058990478515625]
[7, '2024-09-05 13:36:54', 0.330230712890625, 0.2708587646484375,
-0.904541015625]
[8, '2024-09-05 13:36:55', 0.3173065185546875, 0.2705230712890625,
-0.90179443359375]
[9, '2024-09-05 13:36:56', 0.3245849609375, 0.2742462158203125,
-0.89874267578125]
[10, '2024-09-05 13:36:57', 0.3207244873046875, 0.269073486328125,
-0.9063568115234375]
[1, '2024-09-05 13:36:59', 0.3250885009765625, 0.2791748046875,
-0.906524658203125]
[2, '2024-09-05 13:36:59', 0.3097076416015625, 0.271026611328125,
-0.9055023193359375]
[3, '2024-09-05 13:37:00', 0.3223114013671875, 0.2745208740234375,
-0.9078826904296875]
[4, '2024-09-05 13:37:02', 0.3016510009765625, 0.27825927734375,
-0.9047088623046875]
[5, '2024-09-05 13:37:03', 0.31884765625, 0.2796173095703125, -0.91314697265625]
[6, '2024-09-05 13:37:04', 0.3039398193359375, 0.2711181640625,
-0.90997314453125]
[7, '2024-09-05 13:37:05', 0.3169708251953125, 0.2846527099609375,
-0.9066162109375]
[8, '2024-09-05 13:37:06', 0.3129425048828125, 0.27471923828125,
-0.909820556640625]
[9, '2024-09-05 13:37:07', 0.3190155029296875, 0.268768310546875,
-0.9081878662109375]
[10, '2024-09-05 13:37:08', 0.3446197509765625, 0.2545013427734375,
-0.8961029052734375]
[1, '2024-09-05 13:37:09', 0.3502044677734375, 0.2769012451171875,
-0.8955078125]
[2, '2024-09-05 13:37:10', 0.5144805908203125, 0.163604736328125,
-0.8065032958984375]
[3, '2024-09-05 13:37:11', 0.6300506591796875, 0.25634765625,
-0.7377166748046875]
[4, '2024-09-05 13:37:12', 0.620361328125, 0.255859375, -0.7347259521484375]
[5, '2024-09-05 13:37:13', 0.6232452392578125, 0.2537078857421875,
-0.7368927001953125]
[6, '2024-09-05 13:37:14', 0.6241912841796875, 0.2664337158203125,
-0.732391357421875]
[7, '2024-09-05 13:37:15', 0.625640869140625, 0.2592620849609375,
-0.7308197021484375]
[8, '2024-09-05 13:37:16', 0.6181182861328125, 0.29150390625, -0.72955322265625]

[9, '2024-09-05 13:37:17', 0.6093292236328125, 0.2862548828125,
-0.727264404296875]
[10, '2024-09-05 13:37:18', 0.6259765625, 0.2877349853515625,
-0.7279205322265625]
[1, '2024-09-05 13:37:19', 0.599853515625, 0.2743988037109375,
-0.7391204833984375]
[2, '2024-09-05 13:37:20', 0.6020050048828125, 0.2803192138671875,
-0.7350311279296875]
[3, '2024-09-05 13:37:21', 0.6294403076171875, 0.30810546875,
-0.7139434814453125]
[4, '2024-09-05 13:37:22', 0.61944580078125, 0.3067779541015625,
-0.731353759765625]
[5, '2024-09-05 13:37:23', 0.59906005859375, 0.3043212890625,
-0.7378692626953125]
[6, '2024-09-05 13:37:24', 0.6063079833984375, 0.3041229248046875,
-0.7340850830078125]
[7, '2024-09-05 13:37:25', 0.59857177734375, 0.308563232421875,
-0.7346649169921875]
[8, '2024-09-05 13:37:26', 0.5958709716796875, 0.303436279296875,
-0.7350006103515625]
[9, '2024-09-05 13:37:27', 0.6024169921875, 0.301910400390625,
-0.735382080078125]
[10, '2024-09-05 13:37:28', 0.601531982421875, 0.299346923828125,
-0.7359466552734375]
[1, '2024-09-05 13:37:29', 0.6076507568359375, 0.30865478515625,
-0.7327117919921875]
[2, '2024-09-05 13:37:30', 0.5962371826171875, 0.3112640380859375,
-0.7375640869140625]
[3, '2024-09-05 13:37:31', 0.5860748291015625, 0.3047943115234375,
-0.745330810546875]
[4, '2024-09-05 13:37:32', 0.5961761474609375, 0.314239501953125,
-0.73797607421875]
[5, '2024-09-05 13:37:33', 0.5971527099609375, 0.3134918212890625,
-0.73870849609375]
[6, '2024-09-05 13:37:34', 0.5957794189453125, 0.3145599365234375,
-0.737579345703125]
[7, '2024-09-05 13:37:35', 0.5691070556640625, 0.3188323974609375,
-0.747283935546875]
[8, '2024-09-05 13:37:36', 0.57467651367187 5, 0.3409576416015625,
-0.7394561767578125]
[9, '2024-09-05 13:37:37', 0.5840606689453125, 0.3109588623046875,
-0.74188232421875]
[10, '2024-09-05 13:37:38', 0.5546722412109375, 0.368133544921875,
-0.7373504638671875]
[1, '2024-09-05 13:37:40', 0.546875, 0.3621673583984375, -0.740264892578125]
[2, '2024-09-05 13:37:40', 0.5980224609375, 0.4151458740234375,
-0.757049560546875]
[3, '2024-09-05 13:37:42', -0.009735107421875, -0.0822906494140625,

```
-1.1266326904296875]
[4, '2024-09-05 13:37:43', 0.17095947265625, 0.0203094482421875,
-1.0006561279296875]
[5, '2024-09-05 13:37:44', -0.0009765625, -0.0084228515625, -0.99957275390625]
[6, '2024-09-05 13:37:45', -0.001678466796875, -0.00921630859375,
-0.9995880126953125]
[7, '2024-09-05 13:37:46', -0.0018310546875, -0.0096282958984375,
-1.00042724609375]
[8, '2024-09-05 13:37:47', -0.000579833984375, -0.009521484375,
-0.9969940185546875]
[9, '2024-09-05 13:37:48', -0.0008544921875, -0.010711669921875,
-0.999725341796875]
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[151], line 50
     48         # Clear the buffer data for the next time usage
     49         buffer_data.clear()
---> 50 client.update()

File ~\Documents\ANACONDA\Lib\site-packages\arduino_iot_cloud\ucloud.py:473, in
 ↪ArduinoCloudClient.update(self)
    470 self.poll_records()
    472 try:
--> 473     self.poll_mqtt()
    474 except Exception as e:
    475     self.connected = False

File ~\Documents\ANACONDA\Lib\site-packages\arduino_iot_cloud\ucloud.py:377, in
 ↪ArduinoCloudClient.poll_mqtt(self, aiot, args)
    376 def poll_mqtt(self, aiot=None, args=None):
--> 377     self.mqtt.check_msg()
    378     if self.thing_id is not None:
    379         self.senmlpack.clear()

File ~\Documents\ANACONDA\Lib\site-packages\arduino_iot_cloud\umqtt.py:244, in
 ↪MQTTClient.check_msg(self)
    243 def check_msg(self):
--> 244     r, w, e = select.select([self.sock], [], [], 0.05)
    245     if len(r):
    246         return self.wait_msg()

KeyboardInterrupt:
```

[123]:
```python
# Read recorded CSV file
record_df = pd.read_csv("csv_2024-09-05-12-23-36.csv")
record_df
```

[123]:
```
     Unnamed: 0  Index            Timestamp    X_Acce    Y_Acce    Z_Acce
0             0      1  2024-09-05 12:04:59  0.000000  0.000000  0.000000
1             1      2  2024-09-05 12:04:59  0.001022 -0.011246 -1.000183
2             2      3  2024-09-05 12:05:00  0.000809 -0.011078 -0.998886
3             3      4  2024-09-05 12:05:01 -0.027695  0.123444 -1.147537
4             4      5  2024-09-05 12:05:02 -0.472336 -0.384857 -0.318985
..          …    …                  …       …       …       …
995         995    996  2024-09-05 12:23:30  0.003296 -0.009781 -1.000153
996         996    997  2024-09-05 12:23:32  0.002762 -0.010239 -0.999359
997         997    998  2024-09-05 12:23:33  0.003098 -0.010010 -1.001007
998         998    999  2024-09-05 12:23:34  0.003952 -0.010376 -0.999588
999         999   1000  2024-09-05 12:23:35  0.003189 -0.009689 -1.000381

[1000 rows x 6 columns]
```

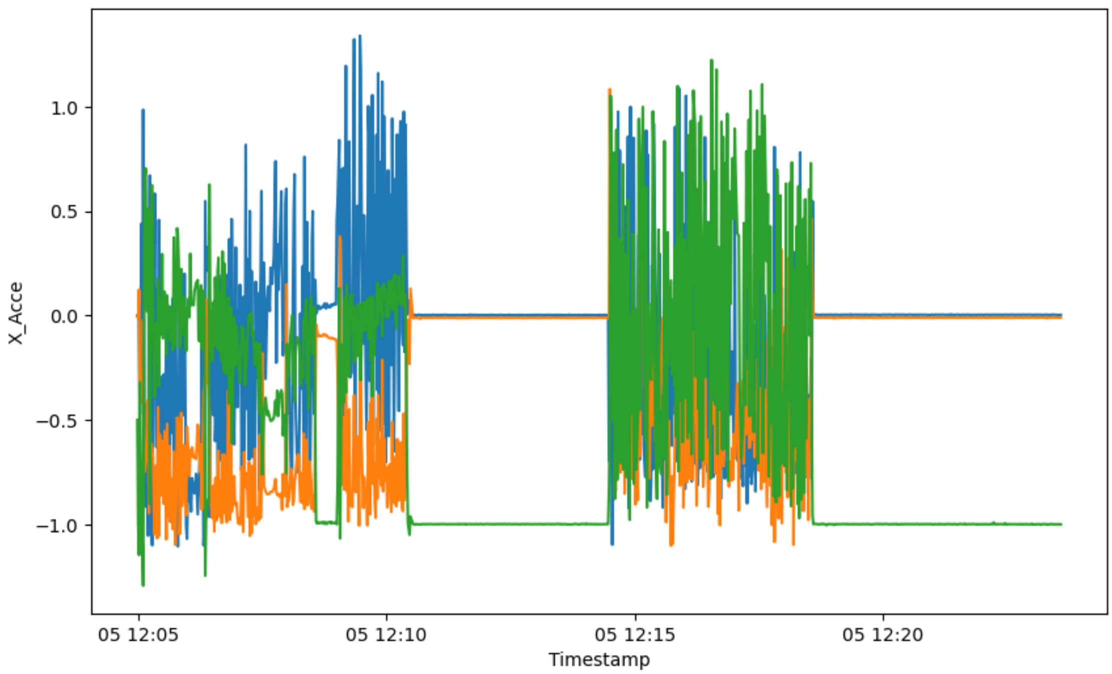
[124]:
```python
# Preprocess the data
record_df.drop(columns = ["Unnamed: 0" , "Index"], inplace = True)
record_df
```

[124]:
```
               Timestamp    X_Acce    Y_Acce    Z_Acce
0    2024-09-05 12:04:59  0.000000  0.000000  0.000000
1    2024-09-05 12:04:59  0.001022 -0.011246 -1.000183
2    2024-09-05 12:05:00  0.000809 -0.011078 -0.998886
3    2024-09-05 12:05:01 -0.027695  0.123444 -1.147537
4    2024-09-05 12:05:02 -0.472336 -0.384857 -0.318985
..                   …       …       …       …
995  2024-09-05 12:23:30  0.003296 -0.009781 -1.000153
996  2024-09-05 12:23:32  0.002762 -0.010239 -0.999359
997  2024-09-05 12:23:33  0.003098 -0.010010 -1.001007
998  2024-09-05 12:23:34  0.003952 -0.010376 -0.999588
999  2024-09-05 12:23:35  0.003189 -0.009689 -1.000381

[1000 rows x 4 columns]
```

[125]:
```python
# Convert datetime column
record_df["Timestamp"] =  pd.to_datetime(record_df["Timestamp"])
```

[126]:
```python
# Plotting
plt.figure(figsize=(10, 6))
sns.lineplot(data = record_df , x = "Timestamp" , y = "X_Acce")
sns.lineplot(data = record_df , x = "Timestamp" , y = "Y_Acce")
sns.lineplot(data = record_df , x = "Timestamp" , y = "Z_Acce")
```

[126]:
```
<Axes: xlabel='Timestamp', ylabel='X_Acce'>
```



[127]:
```
record_df.set_index("Timestamp", inplace = True)
```

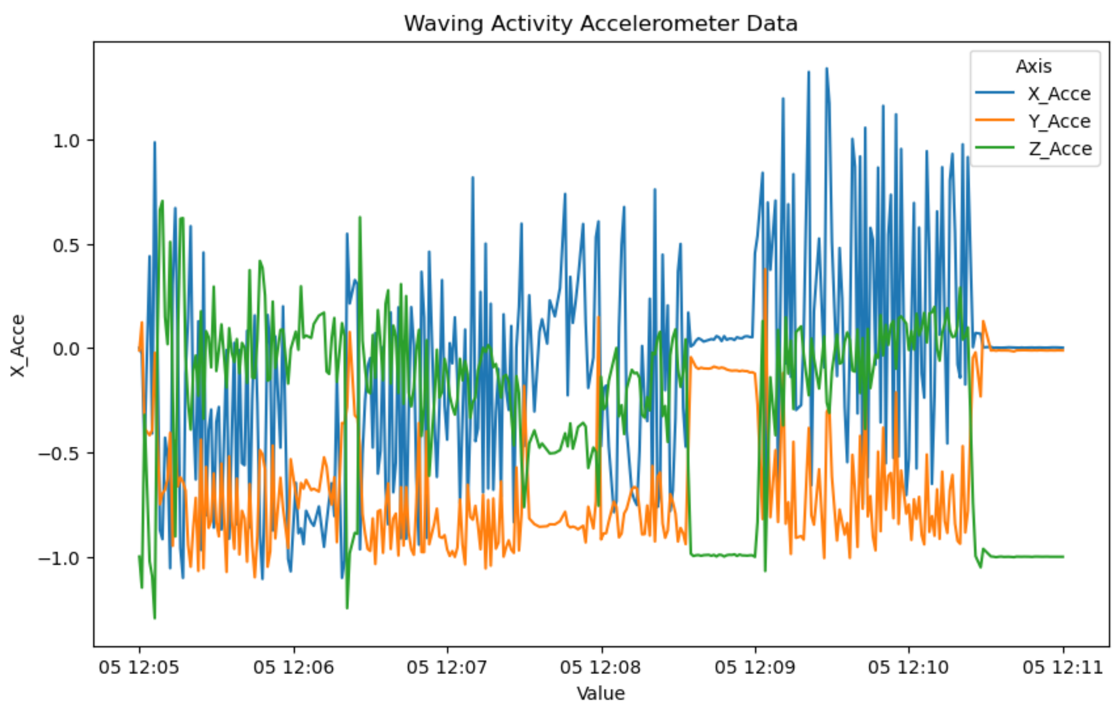
[130]:
```
# Trim out the Activity Waving hands
wave_df = record_df.between_time("12:05", "12:11")
wave_df
```

[130]:
```
                       X_Acce    Y_Acce    Z_Acce
Timestamp
2024-09-05 12:05:00  0.000809 -0.011078 -0.998886
2024-09-05 12:05:01 -0.027695  0.123444 -1.147537
2024-09-05 12:05:02 -0.472336 -0.384857 -0.318985
2024-09-05 12:05:04  0.440201 -0.417160 -1.021774
2024-09-05 12:05:05 -0.408905 -0.393127 -1.093231
…                         …         …         …
2024-09-05 12:10:56  0.003494 -0.011200 -0.999466
2024-09-05 12:10:57  0.003571 -0.011185 -0.999481
2024-09-05 12:10:58  0.003189 -0.010498 -0.999710
2024-09-05 12:10:59  0.002701 -0.010864 -0.998917
2024-09-05 12:11:00  0.002045 -0.010849 -0.999634

[322 rows x 3 columns]
```

[146]:
```
# Plotting
plt.figure(figsize=(10, 6))
plt.title("Waving Activity Accelerometer Data")
```

```
plt.xlabel("Value")
sns.lineplot(data = wave_df , x = wave_df.index , y = "X_Acce" , label =␣
  ↪'X_Acce')
sns.lineplot(data = wave_df , x = wave_df.index , y = "Y_Acce",label = 'Y_Acce'␣
  ↪)
sns.lineplot(data = wave_df , x = wave_df.index , y = "Z_Acce", label =␣
  ↪'Z_Acce')
plt.legend(title = 'Axis')
```

[146]: <matplotlib.legend.Legend at 0x180e7e6e0d0>



[147]:
```
wave_df.describe()
```

[147]:

	X_Acce	Y_Acce	Z_Acce
count	322.000000	322.000000	322.000000
mean	-0.089307	-0.623730	-0.268986
std	0.528154	0.337876	0.431771
min	-1.105911	-1.098038	-1.294266
25%	-0.492493	-0.888271	-0.477131
50%	0.002144	-0.748695	-0.140160
75%	0.170944	-0.407200	0.047398
max	1.340240	0.378006	0.705276

When waving the phone horizontally with its position vertically, the X variable shows the most significant fluctuations because this axis aligns with the direction of our horizontal movement when we move right to left and vice-versa ranging from -1.11 tro 1.34.

The Y variable remains more stable since this axis is oriented vertically and is less influenced by the horizontal waving motion, with values range from -1.1 to 0.38.

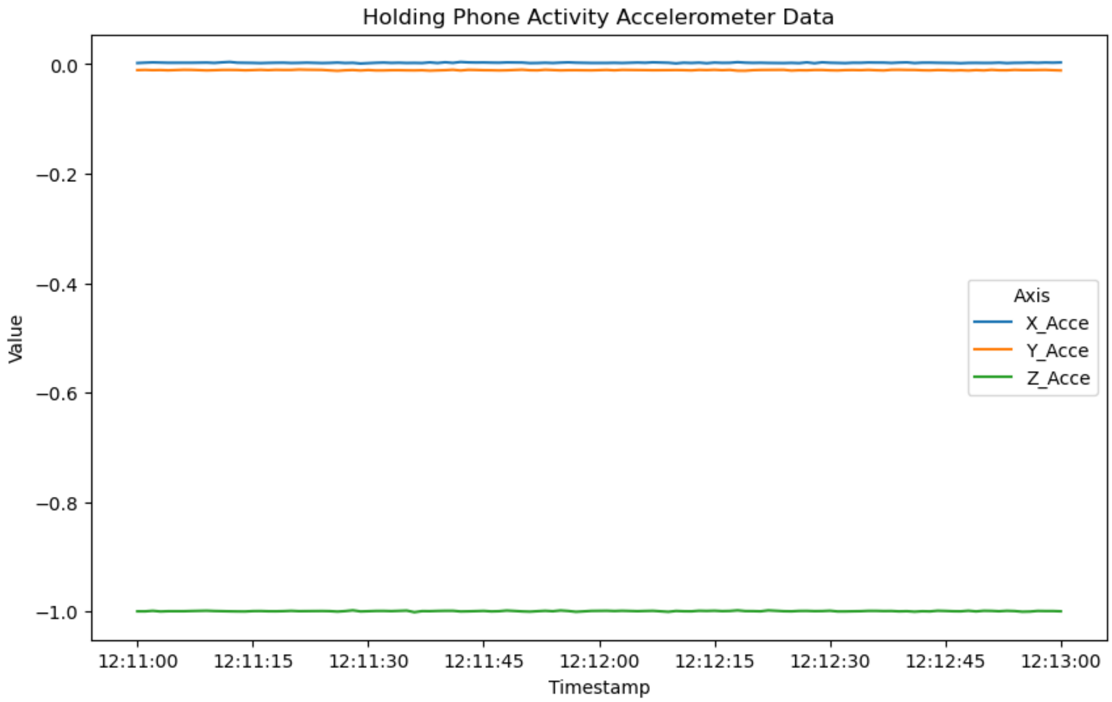
For the Z variable, it fluctuated moderately with values ranging from -1.29 to 0.71, this is due to phone's vertical orientation and the vertical force components experienced during the horizontal movement.

[136]:
```python
# Trim out Hold Phone still on surface activity
hold_df = record_df.between_time("12:11", "12:13")
hold_df
```

[136]:
```
                      X_Acce    Y_Acce    Z_Acce
Timestamp
2024-09-05 12:11:00  0.002045 -0.010849 -0.999634
2024-09-05 12:11:01  0.002686 -0.010483 -0.999680
2024-09-05 12:11:02  0.003220 -0.010956 -0.998688
2024-09-05 12:11:03  0.002869 -0.010757 -0.999924
2024-09-05 12:11:04  0.002518 -0.011246 -0.999420
…                        …         …         …
2024-09-05 12:12:56  0.002914 -0.010818 -1.000031
2024-09-05 12:12:57  0.002457 -0.010635 -0.998917
2024-09-05 12:12:58  0.002975 -0.010468 -0.999130
2024-09-05 12:12:59  0.002670 -0.011002 -0.999146
2024-09-05 12:13:00  0.003098 -0.011505 -0.999588

[112 rows x 3 columns]
```

[148]:
```python
# Plotting
plt.figure(figsize=(10, 6))
plt.title("Holding Phone Activity Accelerometer Data")
plt.ylabel("Value")
sns.lineplot(data = hold_df , x = hold_df.index , y = "X_Acce" , label =␣
 ↪'X_Acce')
sns.lineplot(data = hold_df , x = hold_df.index , y = "Y_Acce" , label =␣
 ↪'Y_Acce')
sns.lineplot(data = hold_df , x = hold_df.index , y = "Z_Acce", label =␣
 ↪'Z_Acce')
plt.legend(title = 'Axis')
```

[148]:
```
<matplotlib.legend.Legend at 0x180e5de99d0>
```



[139]:
```
hold_df.describe()
```

[139]:

	X_Acce	Y_Acce	Z_Acce
count	112.000000	112.000000	112.000000
mean	0.002531	-0.010931	-0.999238
std	0.000501	0.000461	0.000566
min	0.001160	-0.012192	-1.001221
25%	0.002209	-0.011246	-0.999596
50%	0.002502	-0.010956	-0.999268
75%	0.002872	-0.010555	-0.998894
max	0.004013	-0.009796	-0.997803

When the phone is placed still on a surface, the accelerometer readings stabilize as expected. The X remains close to zero, indicating minimal movement along this axis, with values around 0.0025. Furthermore, for Y variable, also stays near zero, with values averaging -0.011 reflecting minimal vertical movement.

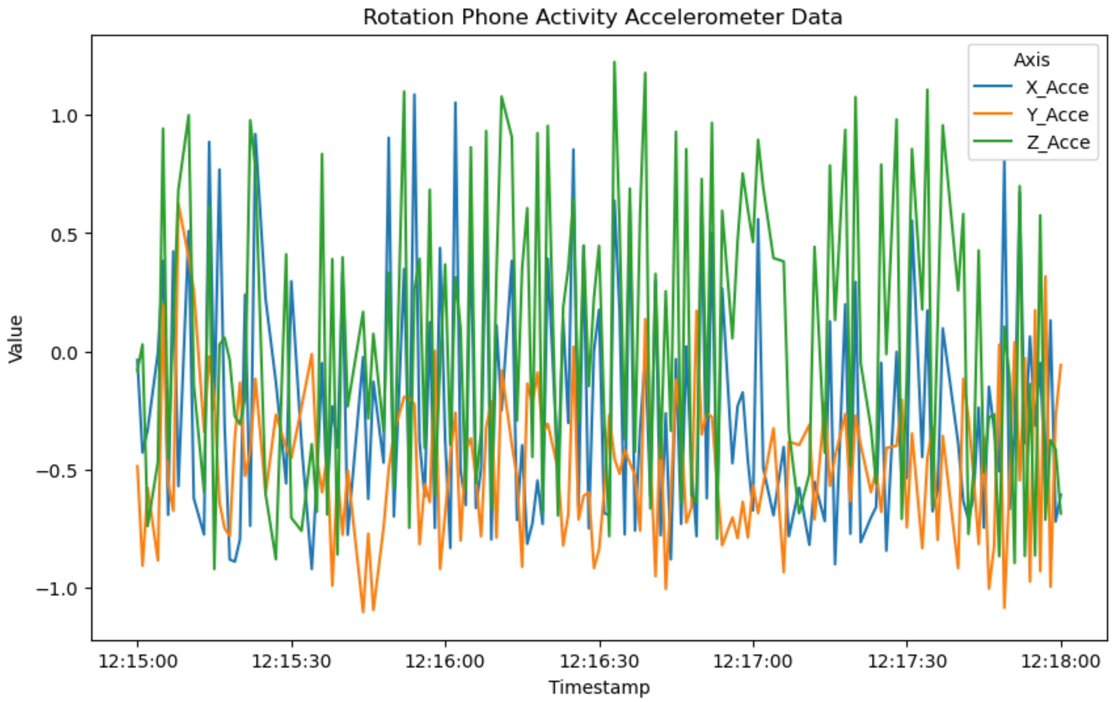
For the Z variable, which remains consistently at -1.0. This stable reading aligns with the expected gravitational pull on the phone when it is held vertically, capturing the vertical force due to gravity

[140]:
```
# Trim out the Rotation Activity
rotate_df = record_df.between_time("12:15", "12:18")
rotate_df
```

[140]:
```
                             X_Acce    Y_Acce    Z_Acce
Timestamp
2024-09-05 12:15:00 -0.036102 -0.485123 -0.081039
2024-09-05 12:15:01 -0.427078 -0.905426  0.029892
2024-09-05 12:15:02 -0.331482 -0.576874 -0.738297
2024-09-05 12:15:04 -0.007309 -0.883804 -0.468002
2024-09-05 12:15:05  0.383759  0.196503  0.942581
…                          …         …         …
2024-09-05 12:17:56 -0.048553 -0.929016  0.575958
2024-09-05 12:17:57 -0.571930  0.318268 -0.711243
2024-09-05 12:17:58  0.131195 -0.994583 -0.374542
2024-09-05 12:17:59 -0.718781 -0.271774 -0.414536
2024-09-05 12:18:00 -0.606125 -0.057449 -0.685089

[155 rows x 3 columns]
```

[149]:
```python
# Plotting
plt.figure(figsize=(10, 6))
plt.title(" Rotation Phone Activity Accelerometer Data")
plt.ylabel("Value")
sns.lineplot(data = rotate_df , x = rotate_df.index , y = "X_Acce", label =␣
 ↪'X_Acce')
sns.lineplot(data = rotate_df , x = rotate_df.index , y = "Y_Acce", label =␣
 ↪'Y_Acce')
sns.lineplot(data = rotate_df , x = rotate_df.index , y = "Z_Acce", label =␣
 ↪'Z_Acce')
plt.legend(title = 'Axis')
```

[149]:
```
<matplotlib.legend.Legend at 0x180e59b15d0>
```



[142]:
```
rotate_df.describe()
```

[142]:

	X_Acce	Y_Acce	Z_Acce
count	155.000000	155.000000	155.000000
mean	-0.252307	-0.473852	0.047857
std	0.507542	0.321925	0.605493
min	-0.919693	-1.102325	-0.919662
25%	-0.672974	-0.709717	-0.474815
50%	-0.403534	-0.496140	0.028763
75%	0.125580	-0.274231	0.583580
max	1.086884	0.624313	1.224472

For the Rotating the Phone Activity, all three variables fluctuated significantly. The X shows a broad range from -0.92 to 1.09, reflecting substantial changes in acceleration due to the phone's rotation. Additionally, The Z also flutuates considerably, with values range from $--0.92$ to 1.22 indicating varying levels of acceleration as the phone rotates.

Just like the other two pervios variables, the Y variable also exhitits substaintial fluctuation, with the values range from -1.1 **to** 0.62**. More interestingly, the Y's values seem to remain slow range of values compared to the other 2 variables.**