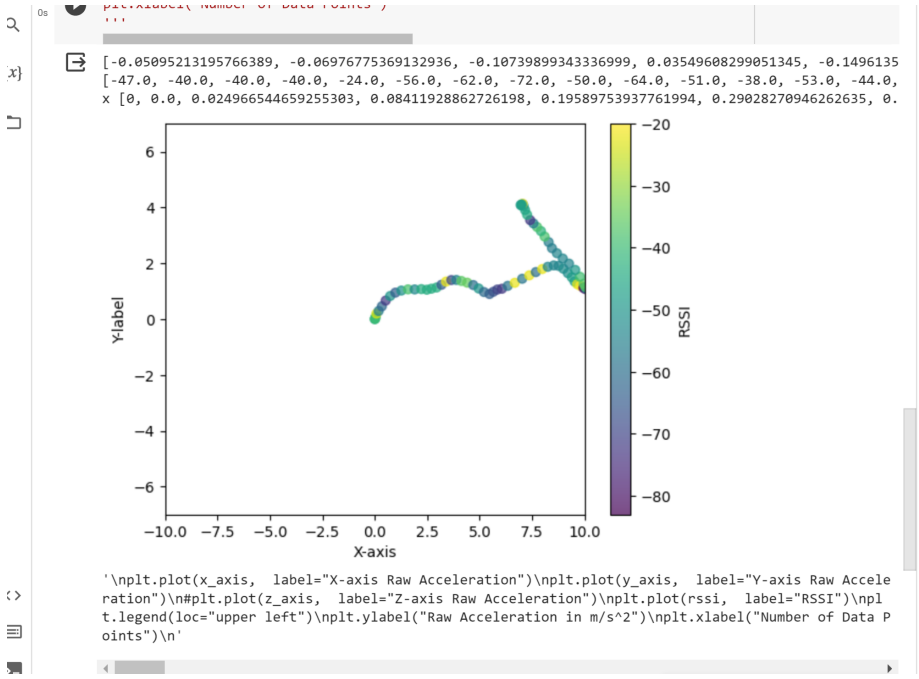
CS437 PostLab3 and Checkpoint 2

Joseph Stack
Prithvi Balaji

Checkpoint 1: checked off in class

Checkpoint 2: not checked off in class

We merged the RSSI collection script from the previous part and the IMU collection code to simultaneously collect both data types and merge them based on timestamp and save a new file with <x,y,RSSI, timestamp> tuples. We walked from the corner of the room and visualized a scatter x-y plot with RSSI as the color code of the points.



Scatter Plot:

Explanation of scatter plot:

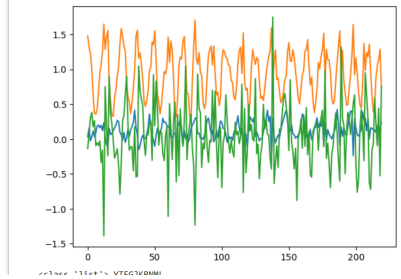
The scatter plot shows the direction of what we were walking in. We began at the origin (0,0) and walked straight in the x-direction and then made a sharp left turn in the positive y-direction. We color coded the RSSI values onto every packet that we observed from the 3 RPI's that were transmitting packets. The RPIs were placed at different locations in the room, which is why the RSSI signals had a pretty strong variance. However, we did notice that there were a handful of packets transmitted that had strong RSSI values at around the point (7.5,1.75), which means that at that location at least one of the Raspberry Pis was most likely there.

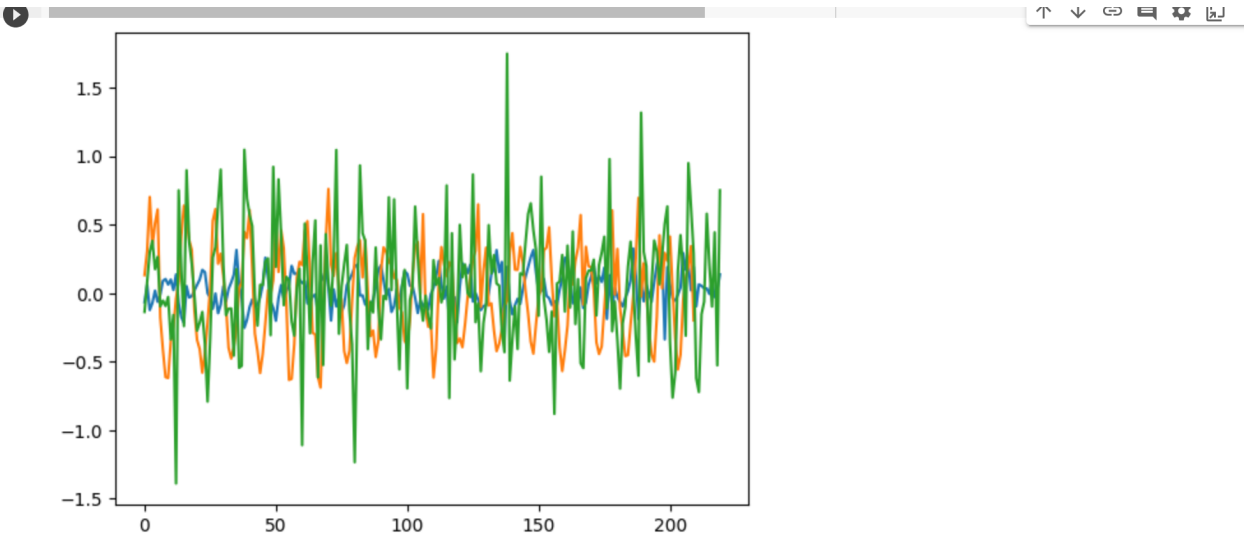
CSV file provided

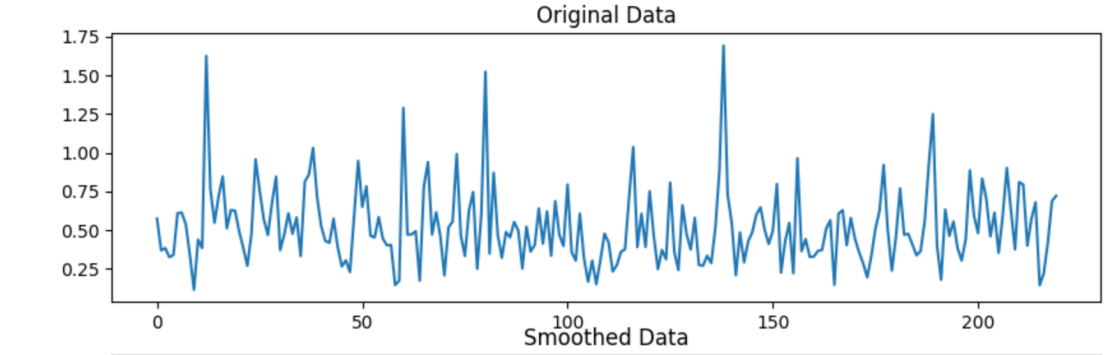
PostLab1:

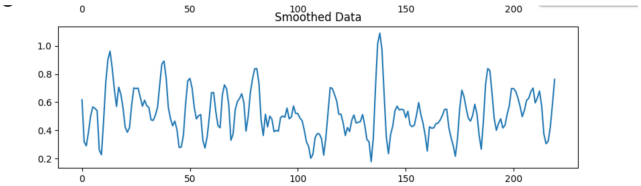
We have the IMUStepDetection code once we create the csv file with the list of tuples from checkpoint 2.

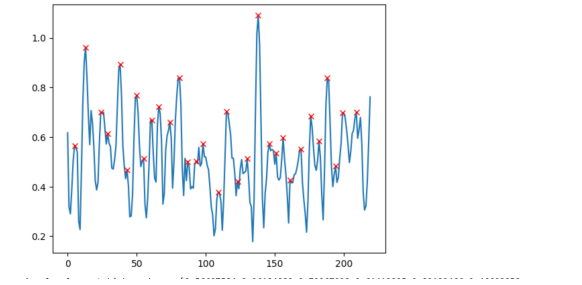
```
print("Trajectory positions are---------------------------->", t)
```

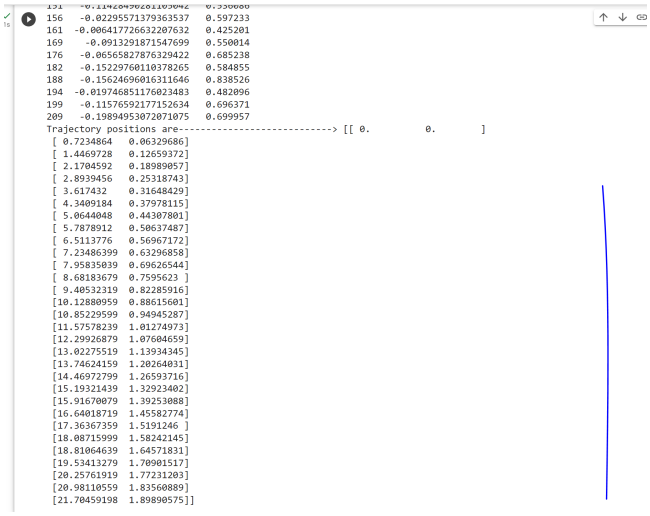












Dead Reckoning Code:

```
def calculate_position(x_accel, y_accel, orientation, step_length):
    # Constants for sensor fusion
    dt = 0.1


    x = 0.0
    y = 0.0


    delta_x = x_accel * (dt ** 2) / 2
    delta_y = y_accel * (dt ** 2) / 2


    rotated_x = delta_x * math.cos(math.radians(orientation)) -
delta_y * math.sin(math.radians(orientation))
    rotated_y = delta_x * math.sin(math.radians(orientation)) +
delta_y * math.cos(math.radians(orientation))

    x += rotated_x
    y += rotated_y


    x *= step_length
    y *= step_length
```

```
    return x, y

To get the fusion method we can use dead reckoning and create the
create_position method:
```

Walking strategies and environment:

We thought about use Kalman filtering and dead reckoning, we figured out that it would be better for Joey to have z-steps (high-walking) to use IMUDetection.py. We set up the environment to take same csv file from Checkpoint 2 and to include the IMUStepDetection code in the PostLab description. We walked to the raspberry pi a bit forward and the rest to left. We took about 20 steps, which was displayed in the array of "trajectory projections" that was printed at the end.

CSV file included:

PostLab2:

Should be zero when we walk to Rpi and then come back it x is 0.

Here we decided to keep track of the maximum RSSI value by storing it in a tuple that also contains the x and y coordinates that our RPI was at when it received the corresponding RSSI signal. Every time we receive a stronger RSSI value, we replace the tuple with the stronger tuple, and when the program is finished running we are able to observe the x and y coordinates where the maximum RSSI value was noted, which can help us detect the area where a spy camera could possibly be located. We tested on all 3 RPIs and for each one we filtered out the other MAC addresses on the other cameras, so we would only analyze transmitted packets from one of the RPIs. The one we have displayed below was the RPI

closest to the door. Our walking strategy during this was to begin near the back left corner of the room as you walk in, and walk towards the TV, turn left, and walk towards the RPI closest to the door. Once we reach that camera we turn around. As we can see in the image of the terminal below, we observe that we obtained the maximum RSSI value in the middle of the test, as the values grew weaker as we walked away from the camera. The graph also displays this as the furthest in the x direction that we walked, we obtained the greatest (strongest) RSSI value.

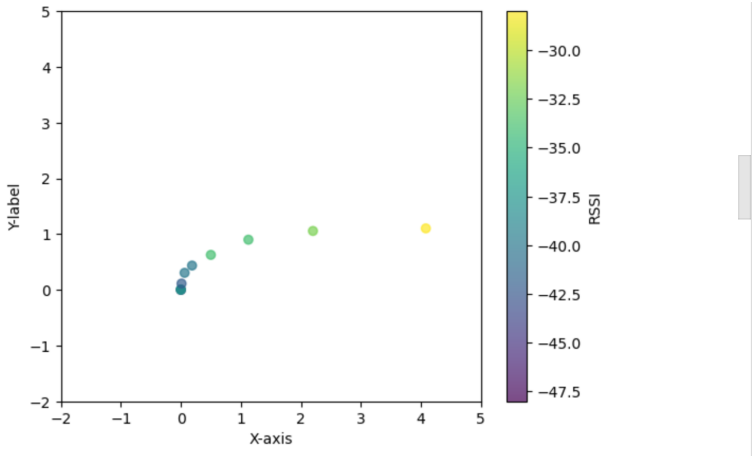
Using the same data cleansing sliding window algorithm we used in Prelab5, Lab3, we were able to smooth our x and y values of our data. Along with this, we attempted to use a spatial interpolation algorithm which is listed below.

```
from scipy.interpolate import griddata

# Assuming data is a list of tuples (x, y, RSSI)
data = [(x1, y1, rssi1), (x2, y2, rssi2), ...]

# Define the grid for interpolation
x_grid, y_grid = np.meshgrid(np.linspace(min_x, max_x, num_points),
np.linspace(min_y, max_y, num_points))

# Interpolate the RSSI values
rssi_interpolated = griddata((x, y), rssi, (x_grid, y_grid),
method='cubic')
```




```
'\nplt.plot(x_axis,  label="X-axis Raw Acceleration")\nplt.plot(y_axis,  label="Y-axis Raw
Acceleration")\n#plt.plot(z_axis,  label="Z-axis Raw Acceleration")\nplt.plot(rssi,  label
="RSSI")\nplt.legend(loc="upper left")\nplt.ylabel("Raw Acceleration in m/s^2")\nplt.xlabe
l("Number of Data Points")\n'
```

PostLab3:

In order to only receive packets from only the motion detection camera, we will use the MAC addresses on the spy cameras and only investigate the RSSI signals that correspond with the motion detection cameras MAC address. By doing this, we are able to filter out transmissions from the other cameras, which will result in a much lower RSSI sampling rate. One way we could utilize the senseHat LED to guide the user to move around the space in order to maximize data collection is by having our LED light on our Raspberry Pi blink faster when we are facing the motion camera, and slower when we are facing in the opposite direction. For instance, when our back is turned to the camera it could block some of the radio frequency signals. Therefore, our RPI wouldn't be able to receive as many RSSI signals, or they will be weaker, and the LED blinking rate would be reduced, notifying us that we are facing in the opposite direction of the spy camera. To implement this in our Raspberry Pi, we decided to use different coloring on our SenseHat to depict how strong the RSSI signals are from the packets that we are receiving. We ordered the colors from red, orange, yellow, and green – green being the strongest and red being the weakest. Using the motion detection camera we constantly created motion in front of the lens, and had it transmit as many packets as possible and walked around with our raspberry pi with the SenseHat on in order to test how close we were to the spy camera.