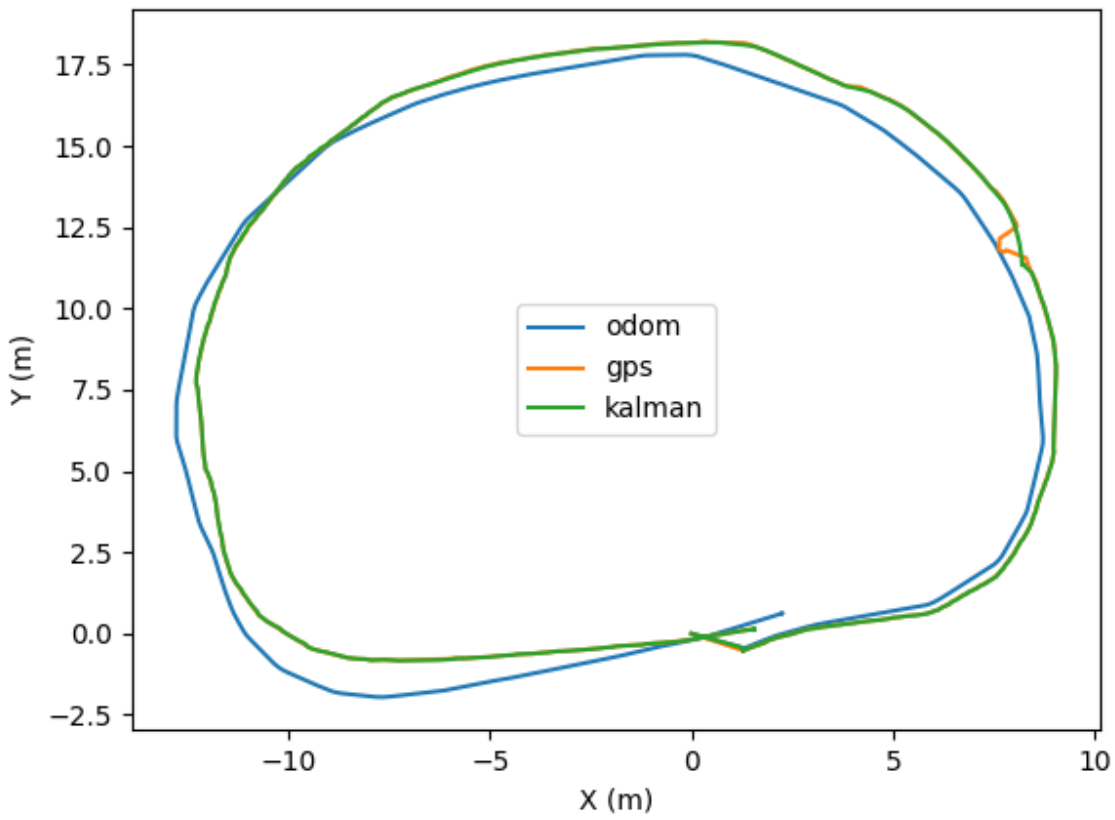
Ryan Hoffman
CPE470
Project 1
February 28, 2022

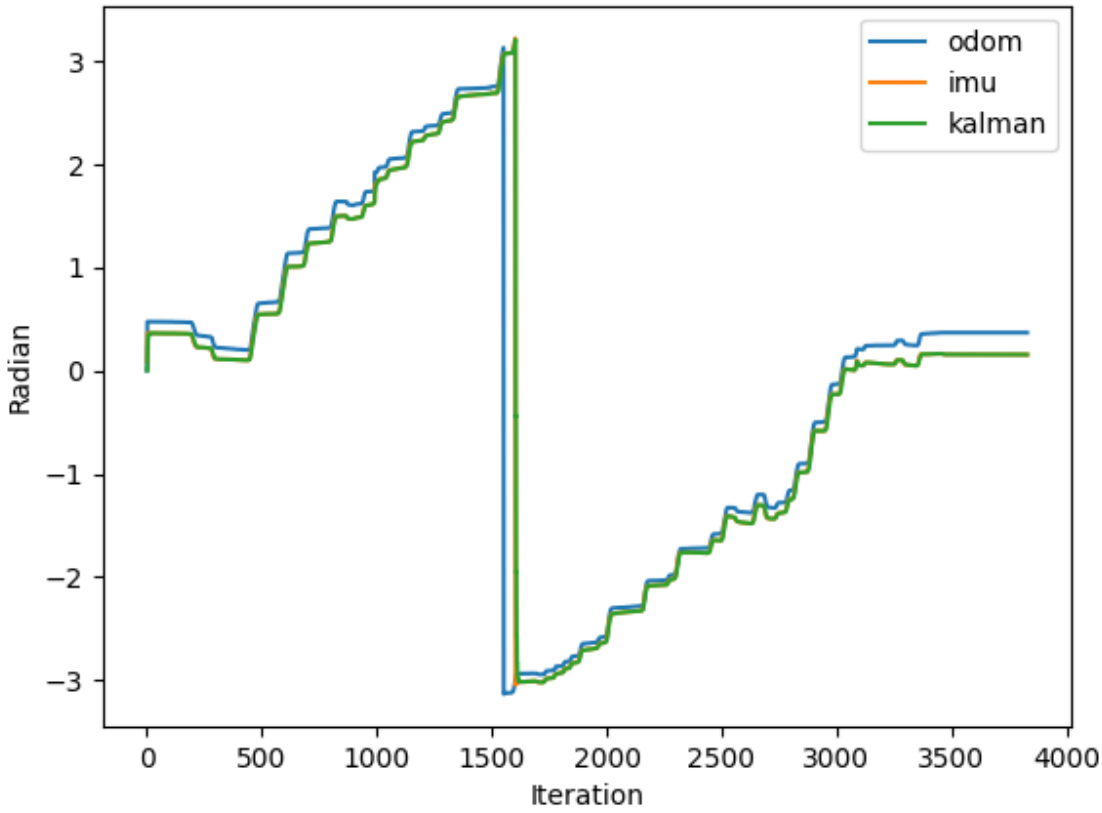
3a. The Kalman Filter outperforms individual data from the GPS and IMU because of its ability to leverage both data sets to find a measurement that eliminates noise and other uncertainties that come from the sensors themselves. The GPS sensor may have a very small amount of error until the robot moves under some sort of object (like a tree) where the GPS signal is weaker, in which case the measurements from the GPS sensor may become very noisy. On the other hand, the IMU sensor may not see an increase in noise as it goes under a tree, however, small errors from the calibration in the IMU may add up over time, leading to increased noise in the final measurements. The Kalman Filter uses both measurements at each iteration to find an updated set of measurements where the noise is smoothed out and errors are decreased.

3b. Below are two graphs (Graph #1 and Graph #2) displaying the position and heading data from my Kalman Filter written in Python. Looking at the first graph, we can see that the kalman output stays very close to the GPS data until the robot navigates under a tree where the GPS data becomes noisy. You can see that the kalman filter returns a much smoother path where the noise from the GPS sensor is eliminated. In the second graph, the kalman heading remains almost identical to the IMU heading.

Graph #1:

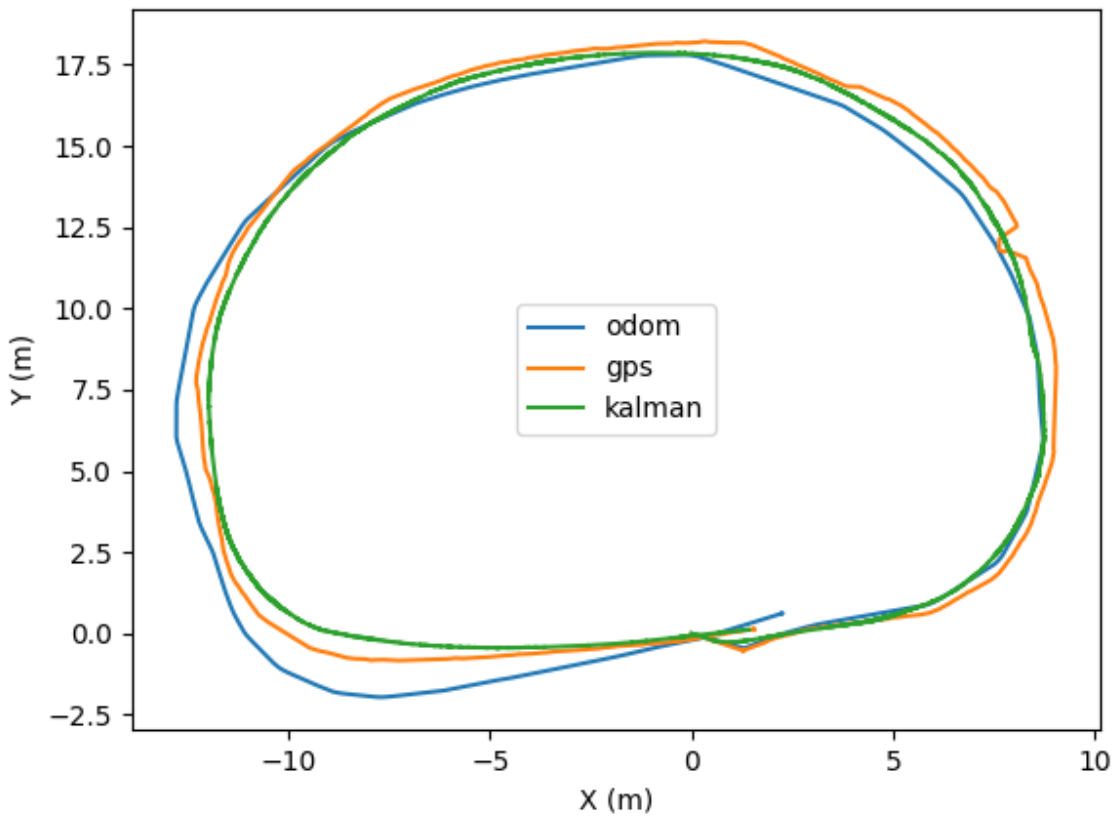


Graph #2:

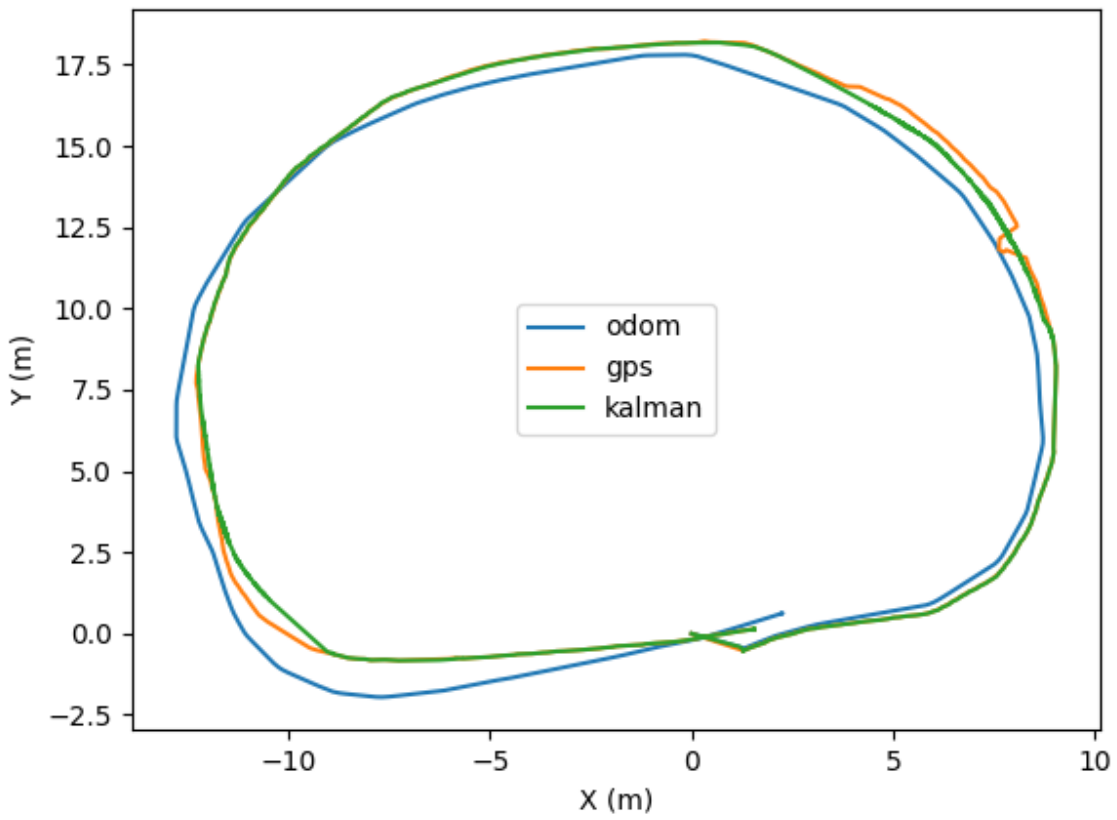


3c. This section contains two graphs (Graph #3, Graph #4). In Graph #3, I random noise to the GPS covariance data by adding a random decimal between the values of 1 and 2 to the covariance data. This shows how a noisy covariance value can result in a Kalman filter path that jumps closer and farther away from the GPS data, resulting in a more inconsistent looking path. In Graph #4, I used the same random noise values from the previous graph, however this time the noise was only added in two ranges of the entire loop. The sections of the graph that have noise are the bottom left and top right sections. In these areas, the graph deviates away from the GPS sensor data and in the areas where noise isn't added, the Kalman path remains similar to the GPS data.

Graph #3:

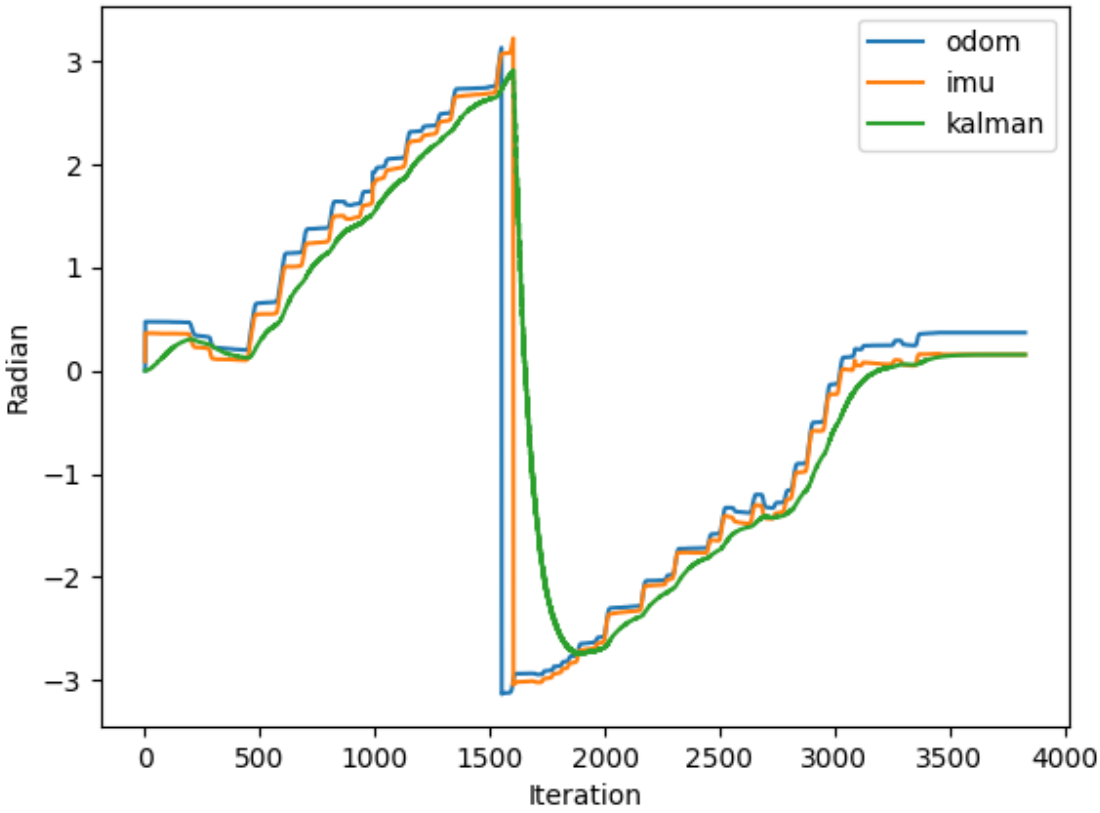


Graph #4:

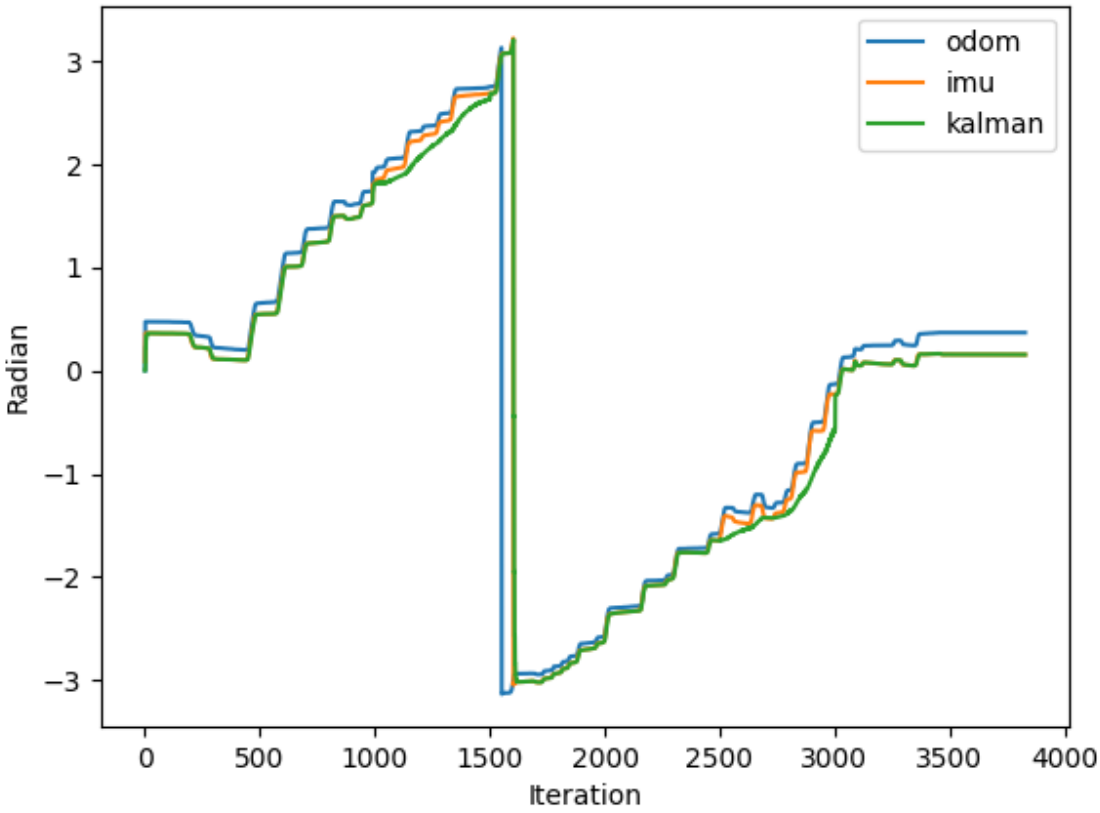


3d. The next two graphs (Graph #5 and Graph #6) show how increasing covariance and adding noise to IMU measurements affect the kalman filter updates. In Graph #5, random noise is added to the entire IMU covariance dataset by adding a random decimal between 1 and 2. Adding noise here disrupts the Kalman Filter data similarly to how adding noise to the GPS covariance did. The KF heading deviates away from the IMU and odometry heading. In Graph #6, noise was only added to two ranges of the graph, both in the upper sections of each upward trend on the graph. You can see in those ranges where the kalman updates move farther away from the actual IMU heading. Adding noise to the IMU covariance causes the Kalman Filter to not work at predicting and correcting the IMU heading.

Graph #5:

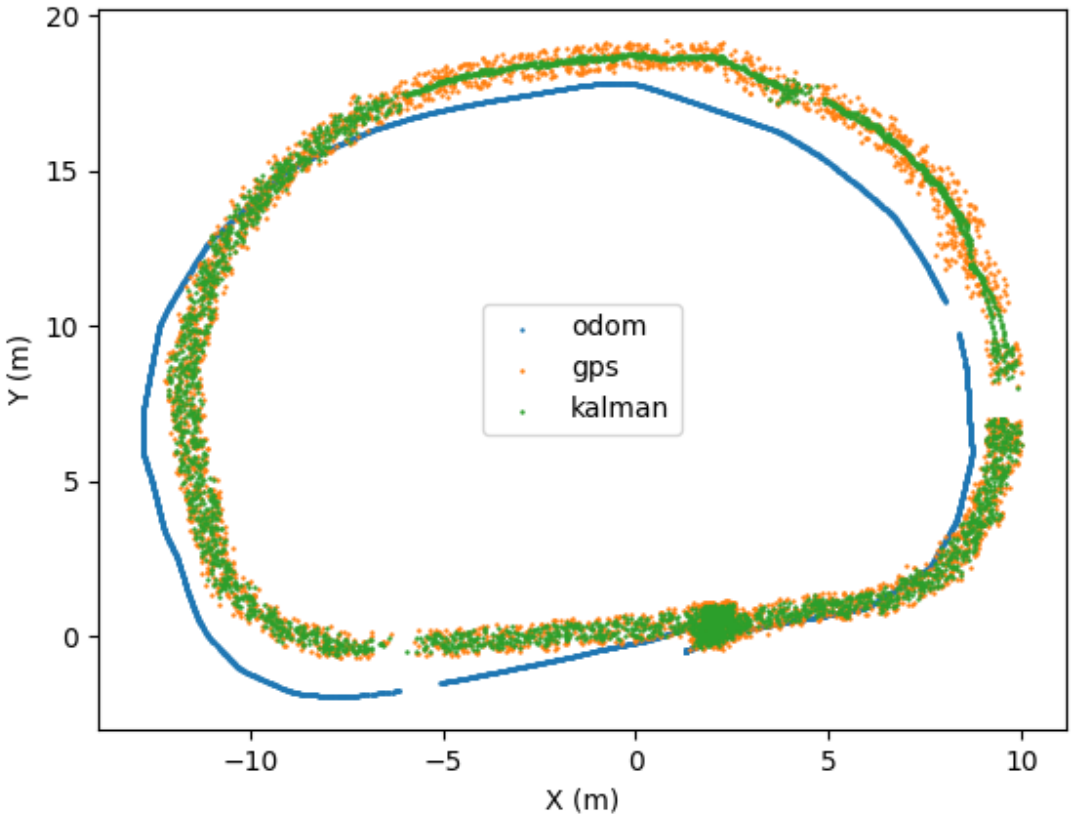

odom
imu
kalman
Radian
Iteration
3
2
1
0
−1
−2
−3
0
500
1000
1500
2000
2500
3000
3500
4000

Graph #6:

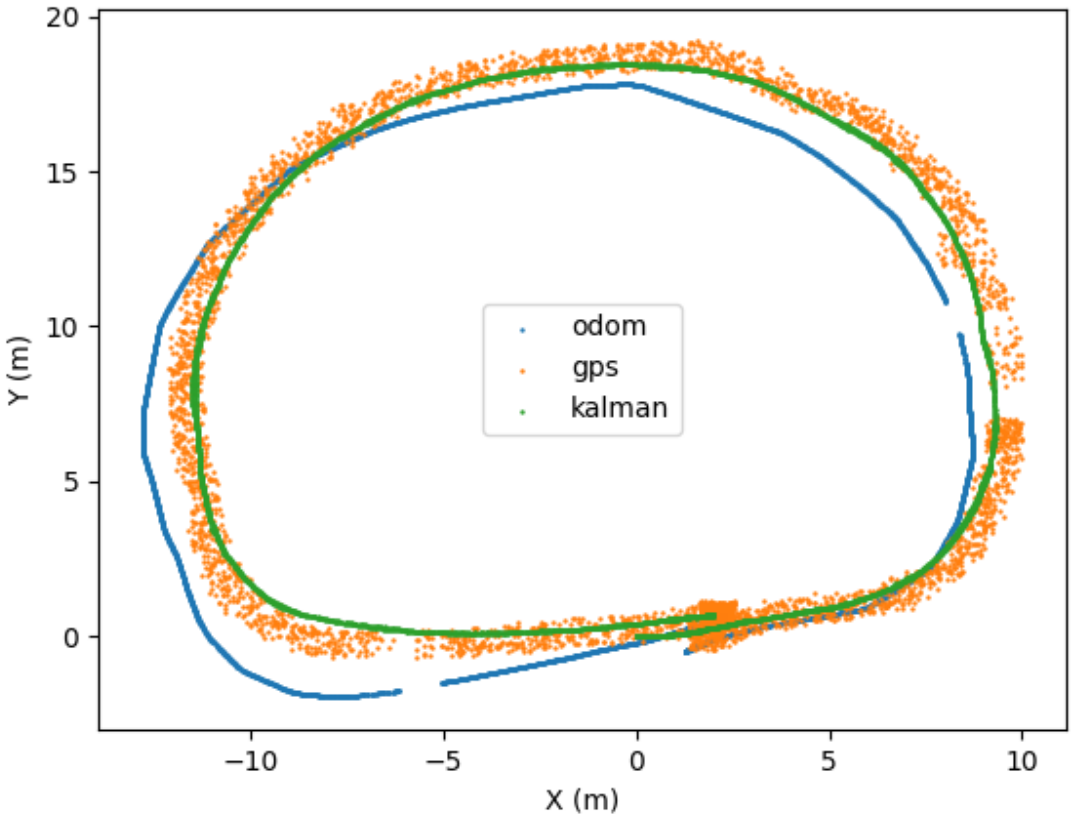


3e. Graphs #7, #8, and #9 show how adding noise to the GPS position data works with the Kalman Filter. In Graph #7, noise is added to the GPS position data for all iterations and the covariance of the GPS sensor is left untouched (not part of the original assignment although I thought it was interesting to observe). I was surprised to see some areas where the Kalman updates are spread almost as far apart as the noisy GPS data and others where the Kalman filter updates are very close together and are able to handle the GPS noise. In Graph #8, noise is once again added to the entire range of the GPS position data along with the noise added to the GPS covariance data. Interestingly, the Kalman Filter output wasn't noisy like the GPS data was. This is possibly due to the higher GPS covariance causing the distance away from the median to be increased. Graph #9 is similar to Graph #8, however the noise was only added to two periods in the graph. The KF path is still off because of the GPS covariance error and continues to remain less noisy in the areas where there's GPS position noise.

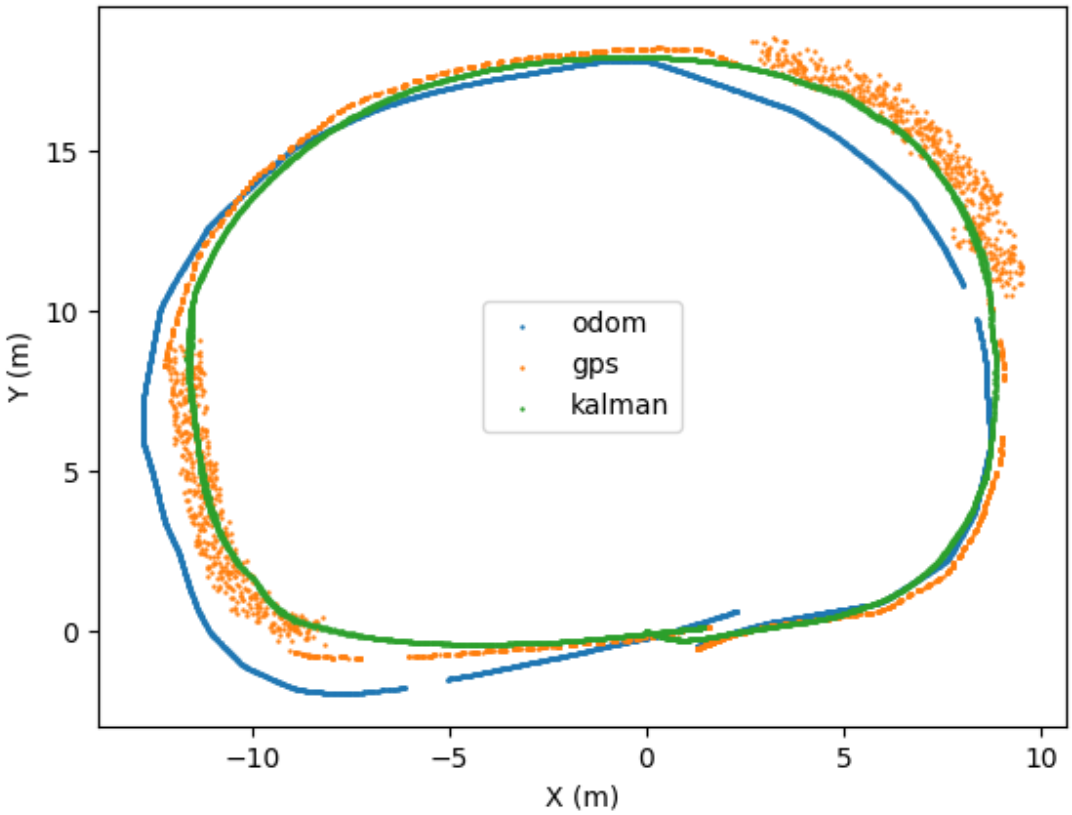
Graph #7:



Graph #8:



Graph #9:



3f. The Python code for this assignment is below. To make things easier, I will also include the actual python file in the zip file of this assignment. You will need python3 as well as the python3 dependencies for the project (**numpy** and **matplotlib)** installed to run the code. To run the code, type **python3 kf.py** in the terminal. The python script will read the data from the **EKF_DATA_circle.txt** file which is expected to be located in the same directory as the kf.py file during runtime.

The python script in its current state will only save two graphs (Graph #1 and Graph #2 from above). To add noise to GPS position data, you can uncomment lines 49-52. To change / add noise to the GPS covariance data, you can uncomment lines 58-61. To change / add noise to IMU covariance data, you can uncomment lines 71-74. To change the position graph to a scatterplot, you can add comments to lines 147-149 and uncomment lines 135-137. To change the orientation graph to a scatterplot, you can add comments to lines 161-163 and uncomment lines 164-166.

```
import math
import numpy as np
from numpy.linalg import multi_dot
from numpy.linalg import inv
import re
import matplotlib.pyplot as plt
import random

def readFile():
    file = open("EKF_DATA_circle.txt", "r")
    lines = file.readlines()
    file.close()

    header = re.sub("%|\n", "", lines[0]).split(",")
    data = {}
    for i in range(len(header)):
        data[header[i]] = []

    for i in range(1, len(lines)):
        line = re.sub("\n", "", lines[i]).split(",")
        for j in range(len(line)):
            data[header[j]].append(float(line[j]))

    return data

def kalmanFilter(s_i, index):
    X_priori = multi_dot([s_i["A"], s[index - 1]["X"]])

    P_priori = np.add(multi_dot([s_i["A"], s[index - 1]["P"], np.transpose(s_i["A"])]),
Q)

    K = multi_dot([P_priori, np.transpose(H), inv(np.add(multi_dot([H, P_priori,
np.transpose(H)]), s_i["R"]))])

    X = np.add(X_priori, multi_dot([K, np.subtract(s_i["Z"], multi_dot([H,
X_priori]))]))

    P = np.subtract(P_priori, multi_dot([K, H, P_priori]))

    return {"X": X, "P": P}

data = readFile()
```

```
#   INITIALIZATION
odom_x = data["field.O_x"]
odom_y = data["field.O_y"]
odom_theta = data["field.O_t"]
gps_x = data["field.G_x"]
gps_y = data["field.G_y"]

#   USED TO CHANGE / ADD NOISE TO GPS POSITION DATA
# for i in range(len(gps_x)):
#     if (2500 <= i <= 3000) or (1000 <= i <= 1500):
#         gps_x[i] += random.uniform(0.1, 1)
#         gps_y[i] += random.uniform(0.1, 1)

gps_co_x = data["field.Co_gps_x"]
gps_co_y = data["field.Co_gps_y"]

#   USED TO CHANGE / ADD NOISE TO GPS COVARIANCE DATA
# for i in range(len(gps_co_x)):
#     if (2500 <= i <= 3000) or (1000 <= i <= 1500):
#         gps_co_x[i] += random.uniform(1, 2)
#         gps_co_y[i] += random.uniform(1, 2)

imu_heading = data["field.I_t"]

#   CALIBARTING IMU HEADING DATA
for i in range(len(imu_heading)):
   imu_heading[i] += 0.32981-0.237156

imu_co_heading = data["field.Co_I_t"]

#   USED TO CHANGE / ADD NOISE TO IMU COVARIANCE DATA
# for i in range(len(imu_co_heading)):
#     if (2500 <= i <= 3000) or (1000 <= i <= 1500):
#         imu_co_heading[i] += random.uniform(1, 2)

V = 0.14
L = 1
delta_t = 0.001
total = len(odom_x)

H = np.array([
```

```
    [1, 0, 0, 0, 0],
    [0, 1, 0, 0, 0],
    [0, 0, 1, 0, 0],
    [0, 0, 0, 1, 0],
    [0, 0, 0, 0, 1]
])

Q = np.array([
    [0.0004, 0, 0, 0, 0],
    [0, 0.0004, 0, 0, 0],
    [0, 0, 0.001, 0, 0],
    [0, 0, 0, 0.001, 0],
    [0, 0, 0, 0, 0.001]
])

s = [{}] * total

s[0]["P"] = np.array([
    [0.001, 0, 0, 0, 0],
    [0, 0.001, 0, 0, 0],
    [0, 0, 0.001, 0, 0],
    [0, 0, 0, 0.001, 0],
    [0, 0, 0, 0, 0.001]
])

omega = V * math.tan(odom_theta[0]) / L

s[0]["X"] = np.array([odom_x[0], odom_y[0], V, odom_theta[0], omega])

#   KALMAN LOOP
for i in range(total - 1):
    A = np.array([
        [1, 0, delta_t * math.cos(odom_theta[i]), 0, 0],
        [0, 1, delta_t * math.sin(odom_theta[i]), 0, 0],
        [0, 0, 1, 0, 0],
        [0, 0, 0, 1, delta_t],
        [0, 0, 0, 0, 1]
    ])

    R = np.array([
        [gps_co_x[i], 0, 0, 0, 0],
        [0, gps_co_y[i], 0, 0, 0],
```

```
        [0, 0, 0.01, 0, 0],
        [0, 0, 0, imu_co_heading[i], 0],
        [0, 0, 0, 0, 0.01]
    ])

    Z = np.array([gps_x[i], gps_y[i], V, imu_heading[i], omega])

    s[i]["A"] = A
    s[i]["R"] = R
    s[i]["Z"] = Z

    if i > 0:
        s[i + 1] = kalmanFilter(s[i], i)

#   GRAPHING
x, y = [], []
for i in range(len(s)):
    x.append(s[i]['X'][0])
    y.append(s[i]['X'][1])

plt.plot(odom_x, odom_y, label="odom")
plt.plot(gps_x, gps_y, label="gps")
plt.plot(x, y, label="kalman")
# plt.scatter(odom_x, odom_y, label="odom", s=0.5)
# plt.scatter(gps_x, gps_y, label="gps", s=0.5)
# plt.scatter(x, y, label="kalman", s=0.5)
plt.legend()
plt.xlabel("X (m)")
plt.ylabel("Y (m)")
plt.savefig("position.png")
plt.clf()

x, y = [], []
for i in range(len(s)):
    x.append(i+1)
    y.append(s[i]['X'][3])

plt.plot(x, odom_theta, label="odom")
plt.plot(x, imu_heading, label="imu")
plt.plot(x, y, label="kalman")
# plt.scatter(x, odom_theta, label="odom", s=0.5)
# plt.scatter(x, imu_heading, label="imu", s=0.5)
```

```
# plt.scatter(x, y, label="kalman", s=0.5)
plt.legend()
plt.xlabel("Iteration")
plt.ylabel("Radian")
plt.savefig("orientation.png")
plt.clf()
```