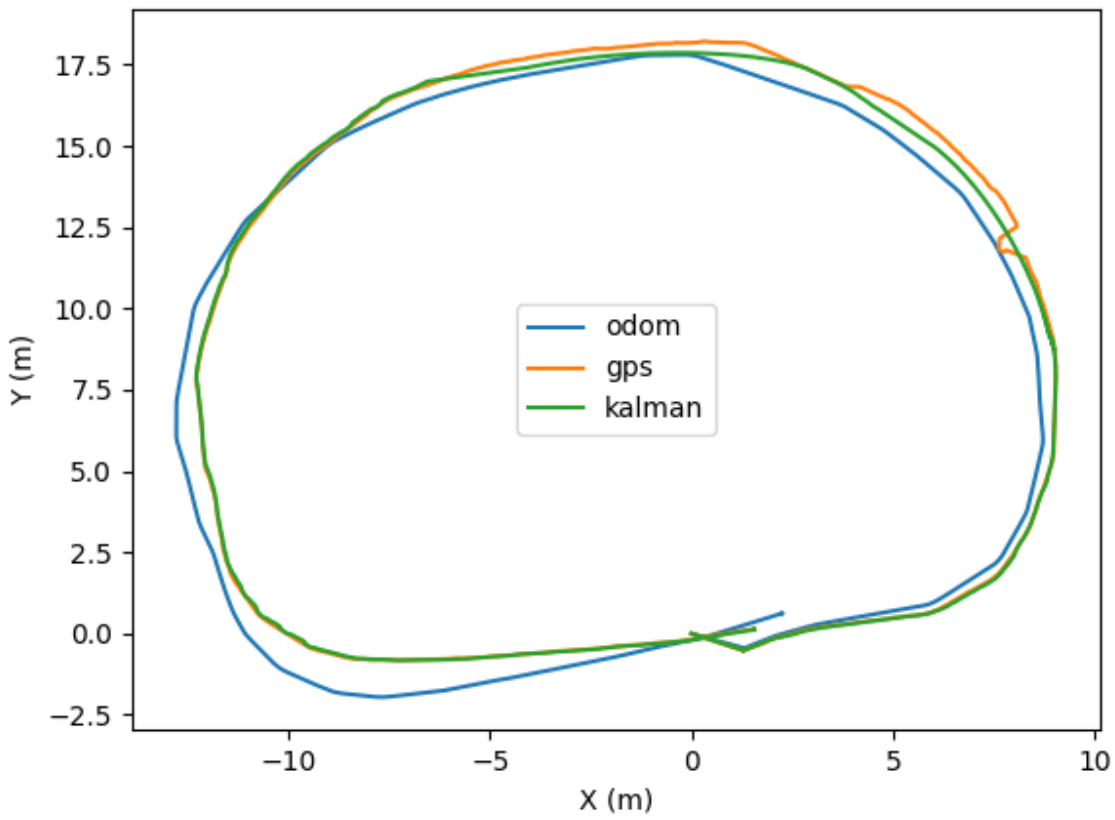
Ryan Hoffman
CPE470
Project 1
February 28, 2022

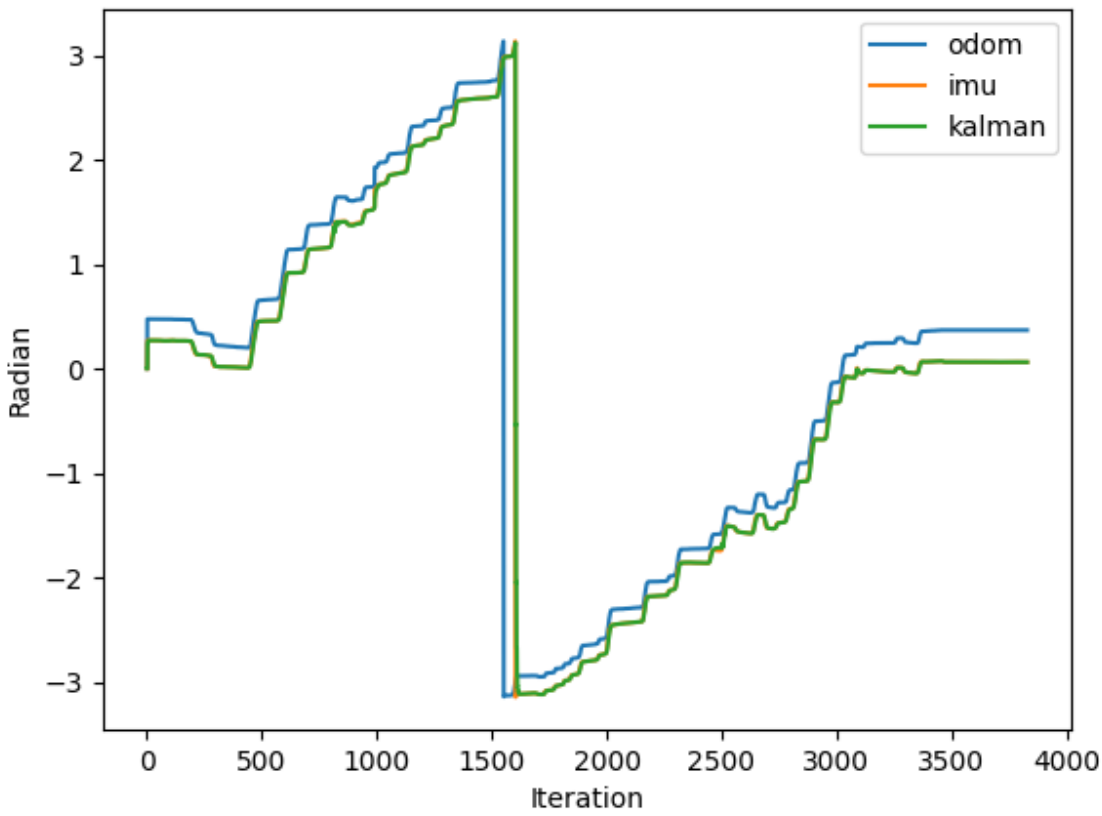
3a. The Kalman Filter outperforms individual data from the GPS and IMU because of its ability to leverage both data sets to find a measurement that eliminates noise and other uncertainties that come from the sensors themselves. The GPS sensor may have a very small amount of error until the robot moves 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:

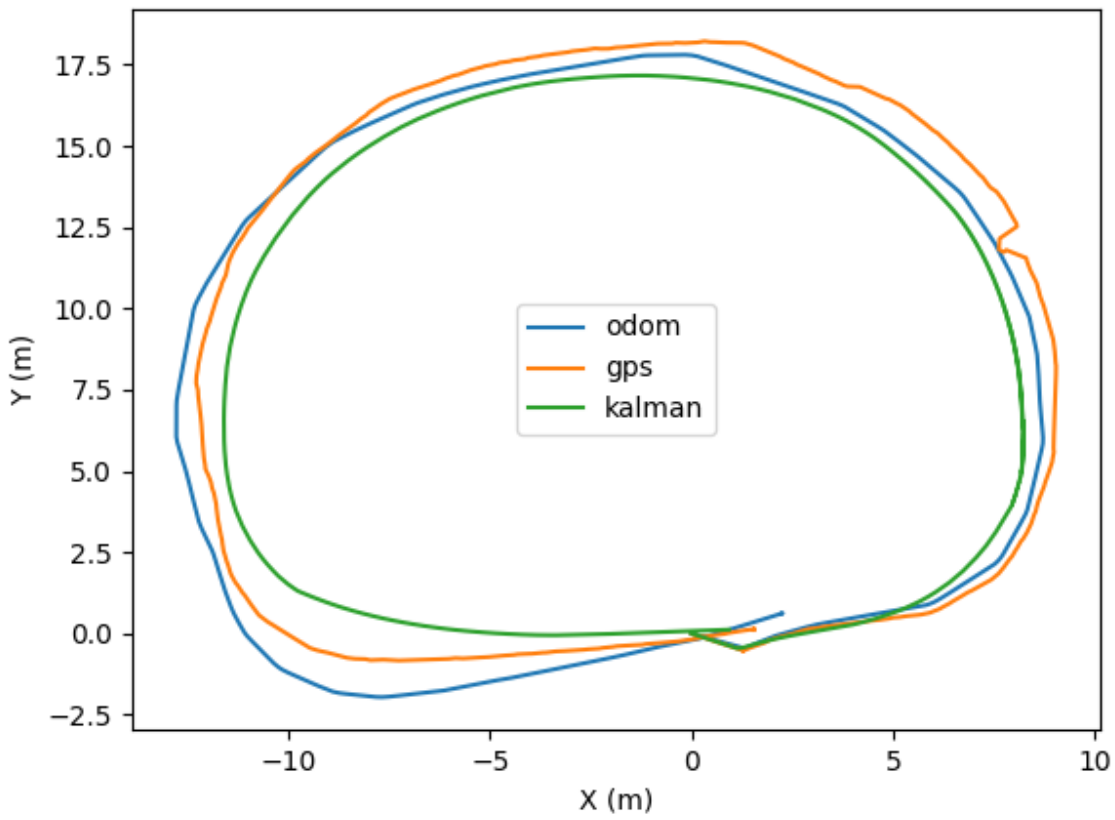


Graph #2:

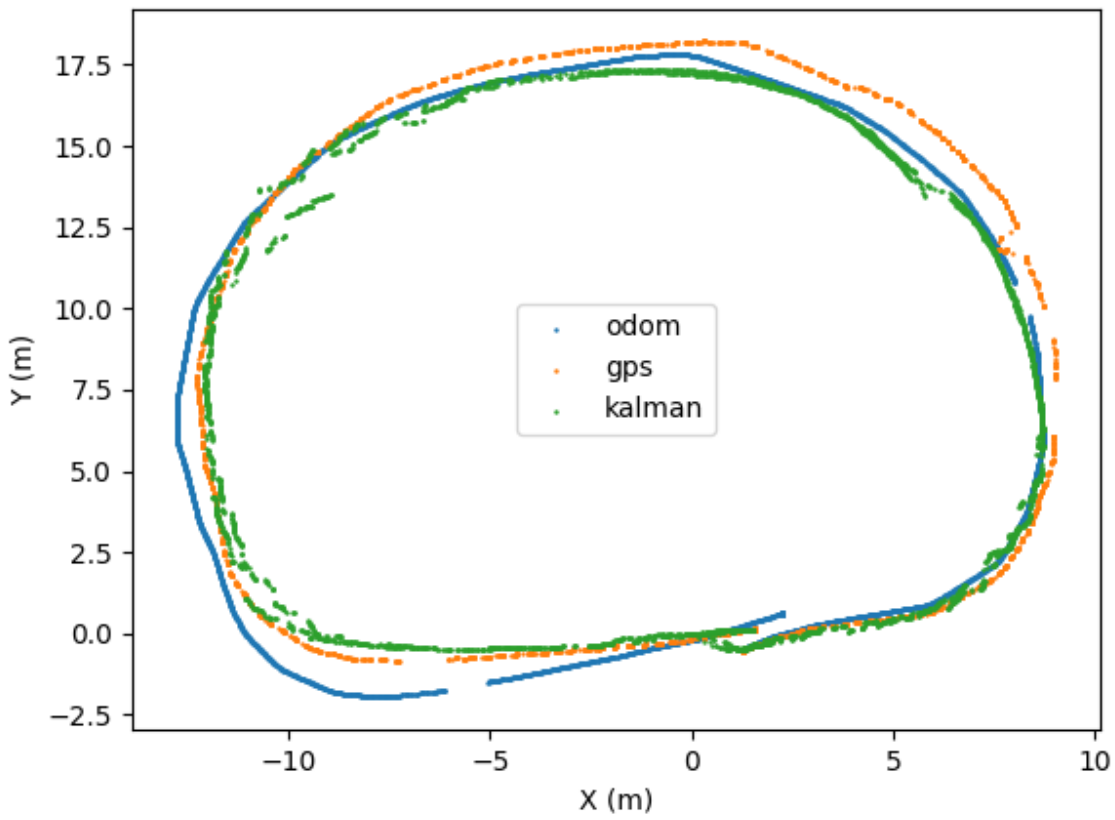


3c. This section contains three graphs (Graph #3, Graph #4, and Graph #5). In Graph #3, I increased the value of the X and Y covariance data (GPS sensor) at each iteration by 0.1. The resulting graph shows the Kalman filter taking a completely different path than what was recorded on the GPS and odometer. This shows that with a very high covariance between data measurements, the Kalman filter doesn't work. In Graph #4, I kept the GPS covariance data increased by 0.1 like the last graph, but also added random noise by adding a random decimal between the values of -0.1 and 0.1 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 #5, I used the same changed covariance value with random noise, 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. We can see that the rest of the graph still has the wrong path (like Graph #3), but is much more consistent in position.

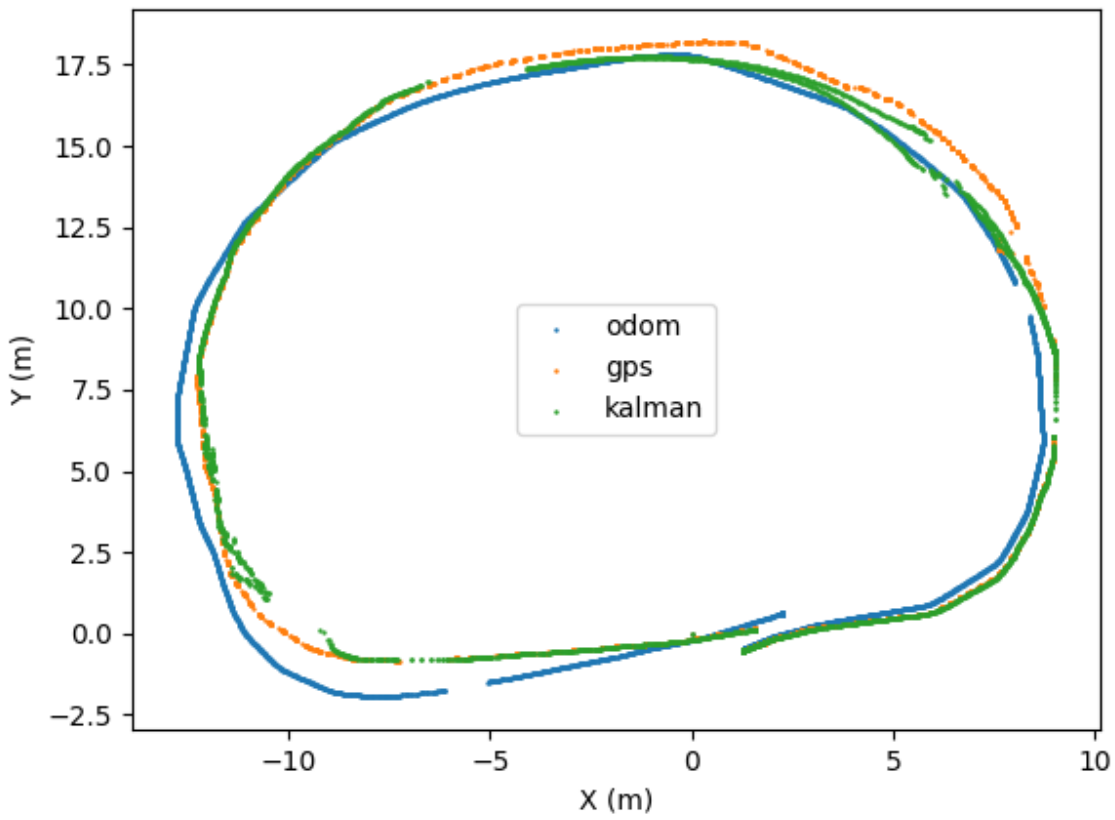
Graph #3:



Graph #4:

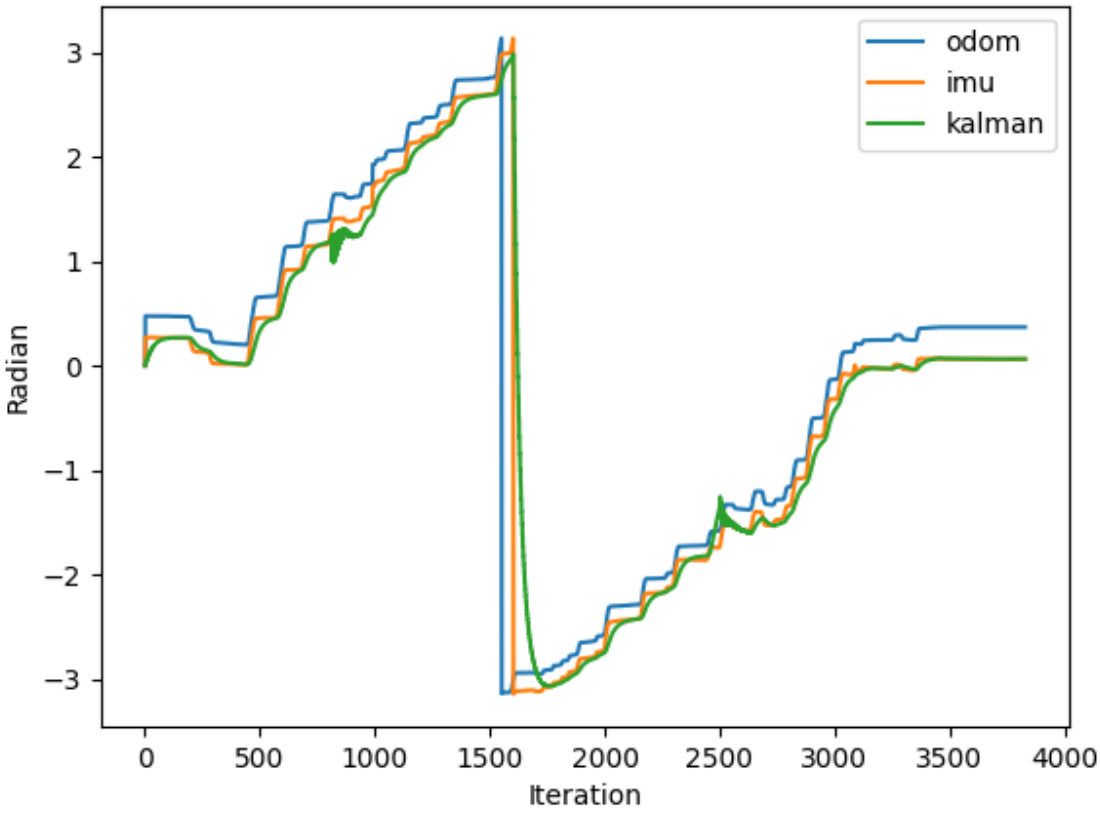


Graph #5:

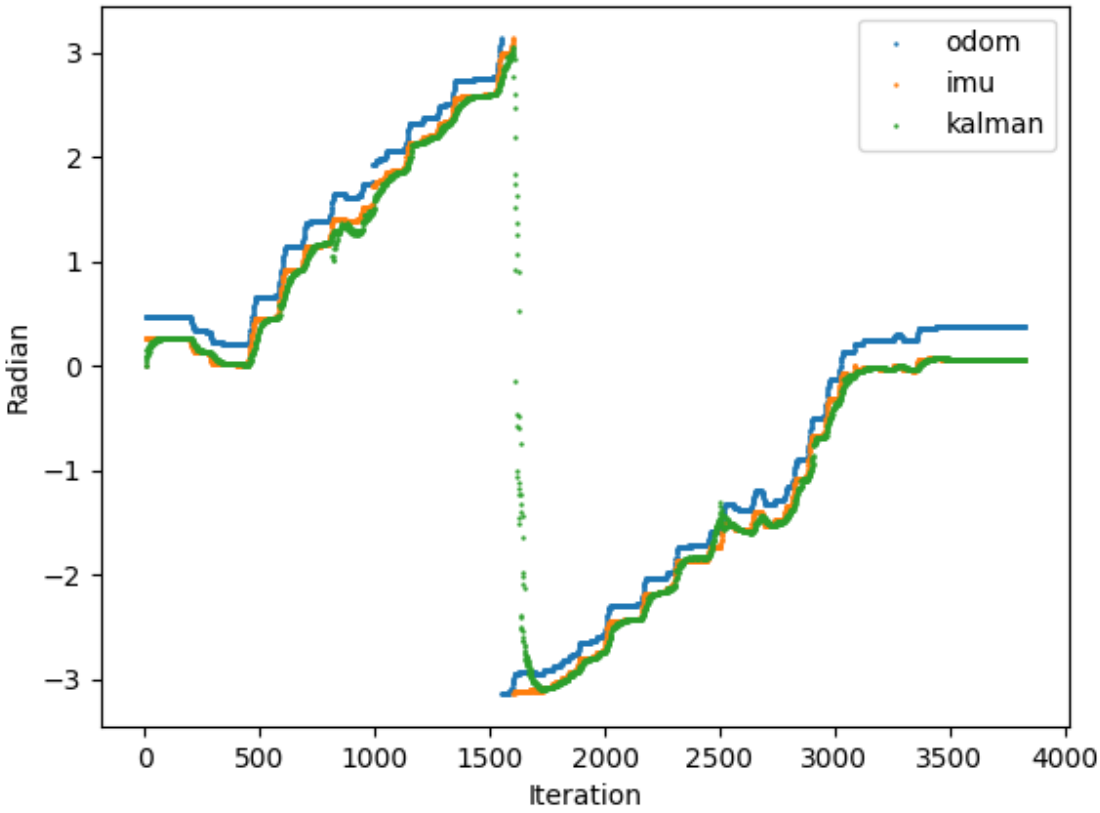


3d. The next three graphs (Graph #6, Graph #7, Graph #8) show how increasing covariance and adding noise to IMU measurements affect the kalman filter updates. In Graph #6, a value of 0.2 was added to the IMU covariance data for all iterations. Similarly to increasing GPS x/y covariance in Graph #3, the updates to the IMU heading cause its graph to stray further from the original IMU data, leading to an incorrect heading prediction. In Graph #7, the IMU covariance is still increased by 0.2 and random noise is added to the entire dataset by adding a random decimal between -0.2 and 0.2. Adding noise here doesn't disrupt the points on the graph nearly as much as adding noise to the gps covariance did. In Graph #8, noise was only added to two ranges of the graph, both in the upper sections of each line on the graph. You can see in those ranges where the kalman updates move farther away from the actual IMU heading.

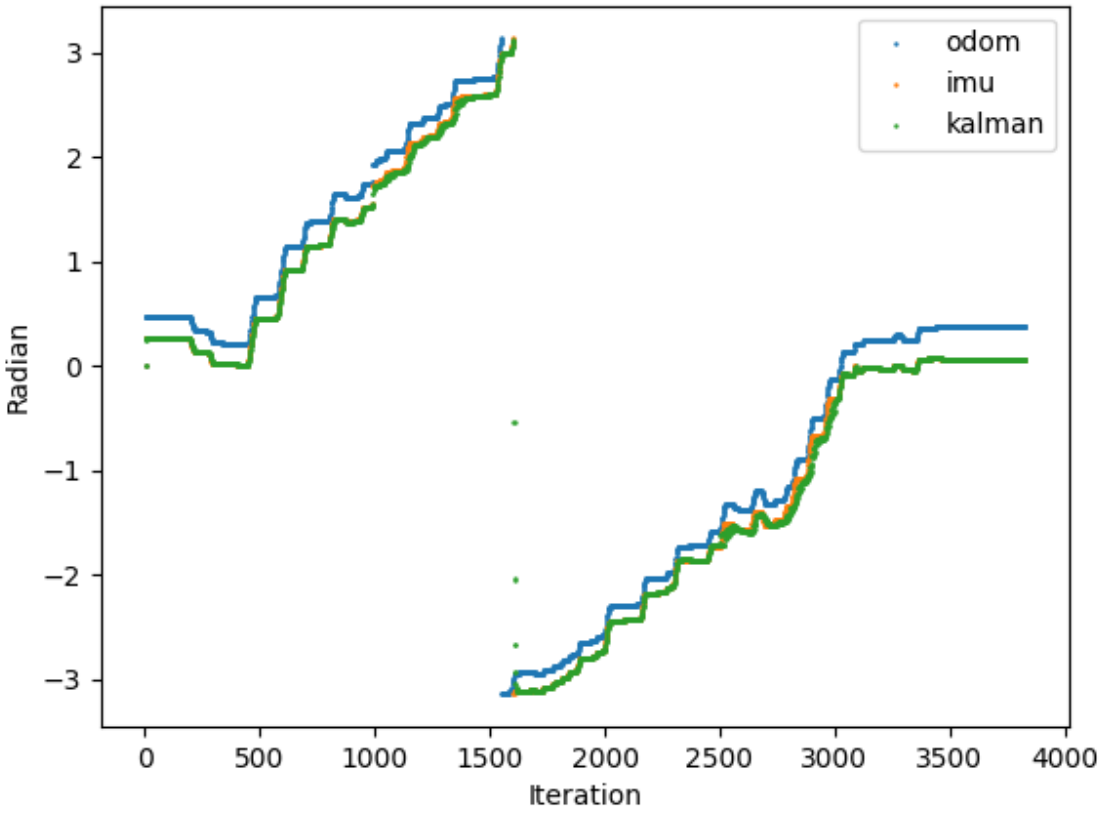
Graph #6:


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

Graph #7:

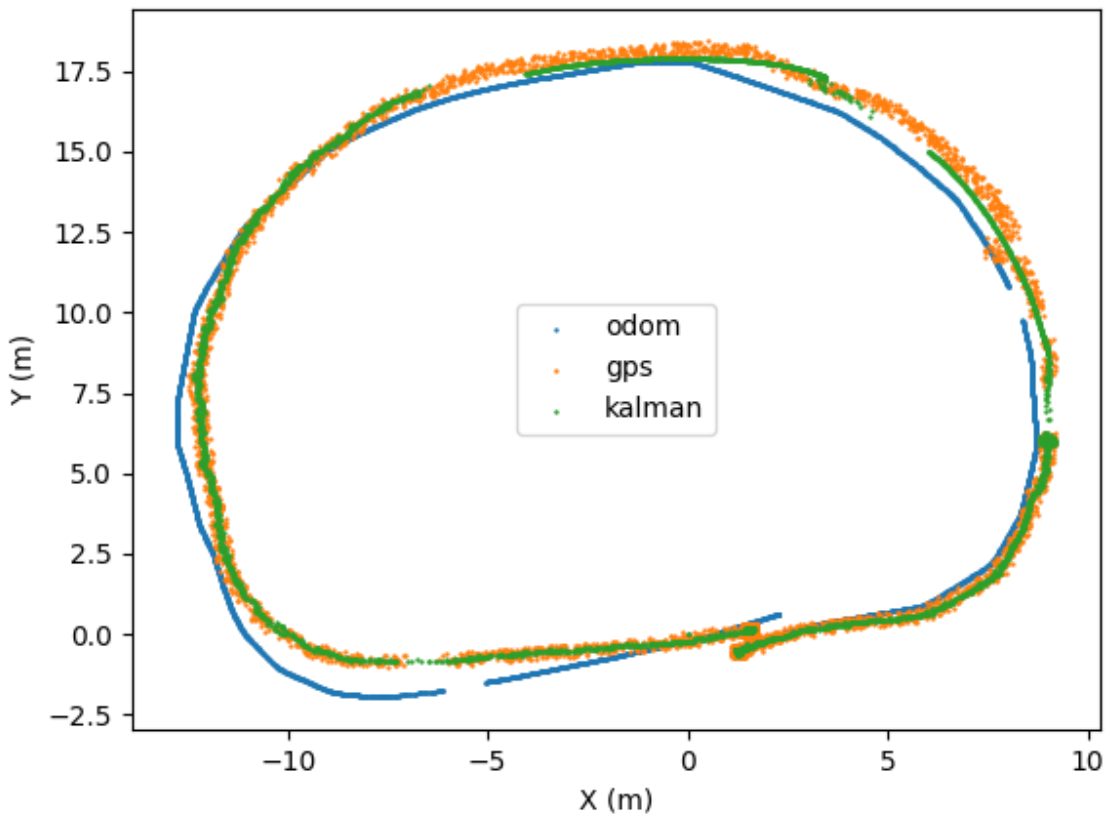

odom
imu
kalman
Radian
Iteration
3
2
1
0
−1
−2
−3
0
500
1000
1500
2000
2500
3000
3500
4000

Graph #8:

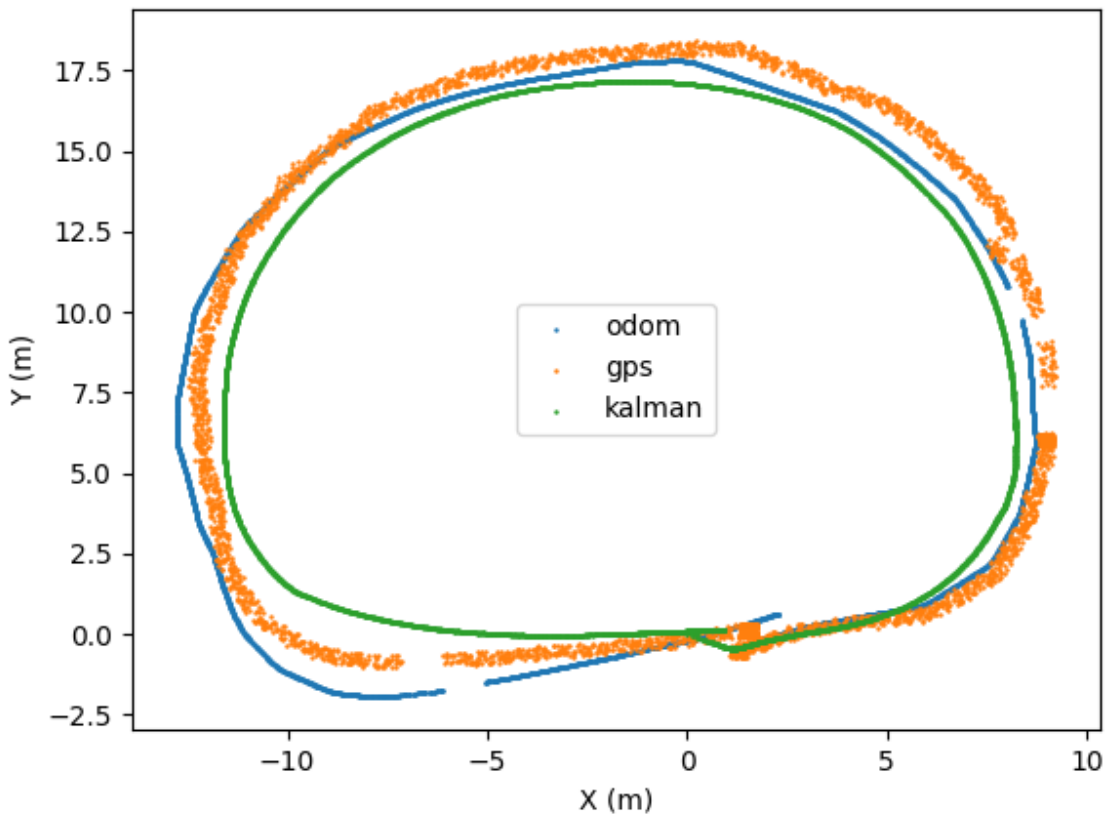


3e. Graphs #9, #10, and #11 shows how adding noise to the GPS position data works with the Kalman Filter. In Graph #9, noise is added to the GPS position data for all iterations and the covariance of the GPS sensor is left untouched. The updates from the Kalman Filter contain coordinates that are closer together than the GPS position coordinates. In Graph #10, the covariance of the GPS data is increased by 0.1 in all iterations and noise is added to the GPS positions in all iterations by adding a random decimal between -0.2 and 0.2. The result is similar to increasing only the covariance of the GPS data, except the original GPS data also has the added noise so the KF data is spread farther apart. Graph #11 is the same as Graph #10, except that noise is only added to the GPS position data in two ranges. You can see the thicker parts of the GPS points where the noise is added. Surprisingly, the difference between adjacent points on the KF plot remains about the same throughout the graph.

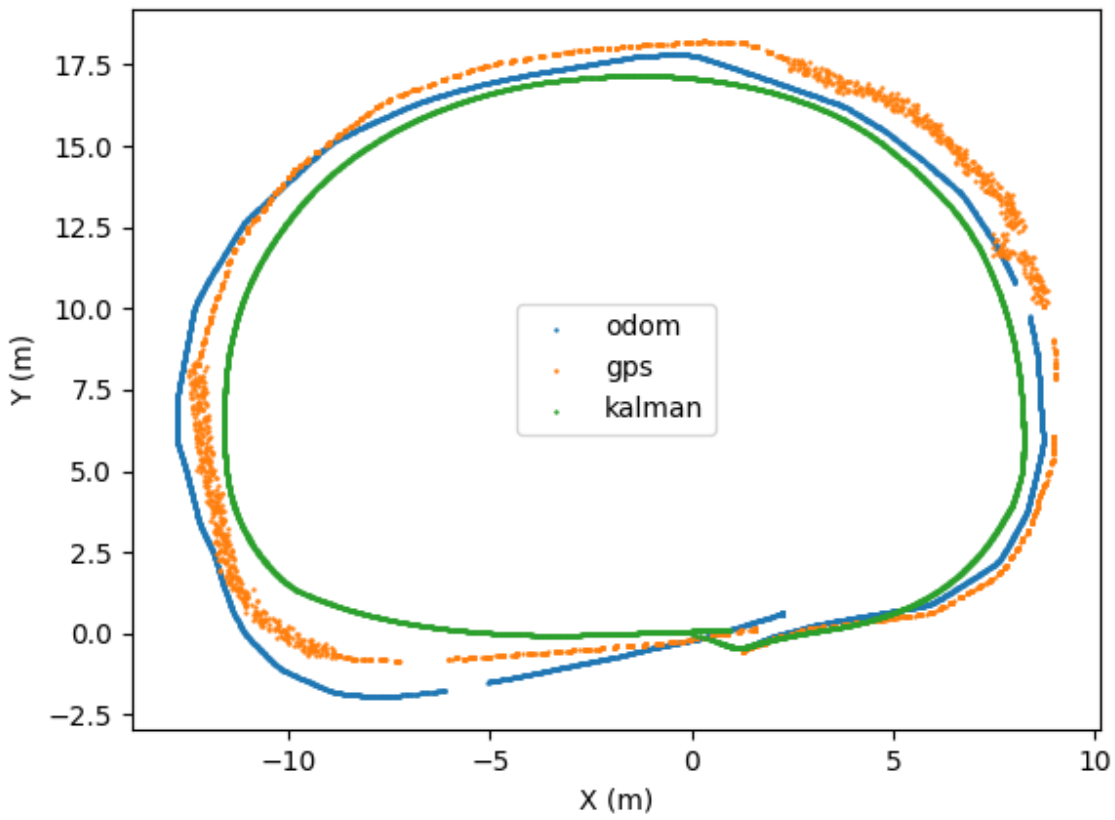
Graph #9:



Graph #10:



Graph #11



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 47-50. To change / add noise to the GPS covariance data, you can uncomment lines 54-57. To change / add noise to IMU covariance data, you can uncomment lines 61-63. To change the position graph to a scatterplot, you can add comments to lines 132-134 and uncomment lines 135-137. To change the orientation graph to a scatterplot, you can add comments to lines 149-151 and uncomment lines 152-154.

```
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()
```

```
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(-0.2, 0.2)
#         gps_y[i] += random.uniform(-0.2, 0.2)
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] += 0.1 + random.uniform(-0.1, 0.1)
#         gps_co_y[i] += 0.1 + random.uniform(-0.1, 0.1)
imu_heading = data["field.I_t"]
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] += 0.2 + random.uniform(-0.2, 0.2)
V = 0.44
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([
   [0.00001, 0, 0, 0, 0],
   [0, 0.00001, 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([
    [0.01, 0, 0, 0, 0],
    [0, 0.01, 0, 0, 0],
    [0, 0, 0.01, 0, 0],
    [0, 0, 0, 0.01, 0],
    [0, 0, 0, 0, 0.01]
])

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

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

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]
    ])

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

    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)

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()
```