Jonathan Nusantara (jan265)
AMR
HW 4

Dynamics and measurement functions

1. The g function is a function that will be used to estimate the current robot pose based on previous pose and user control. The control that is given is the odometry of distance and angle turned based on the control. Since we are given the odometry function that we can use is integrateOdom(), as it will integrate the odometry values and incorporate it to the previous pose, and then provide an estimated current pose.

$$\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} x_{t-1} + \frac{\text{distance}}{\phi}(\sin(\theta+\phi) - \sin(\theta)) \\ y_{t-1} - \frac{\text{distance}}{\phi}(\cos(\theta+\phi) - \cos(\theta)) \\ \theta_t + \phi \end{bmatrix}$$

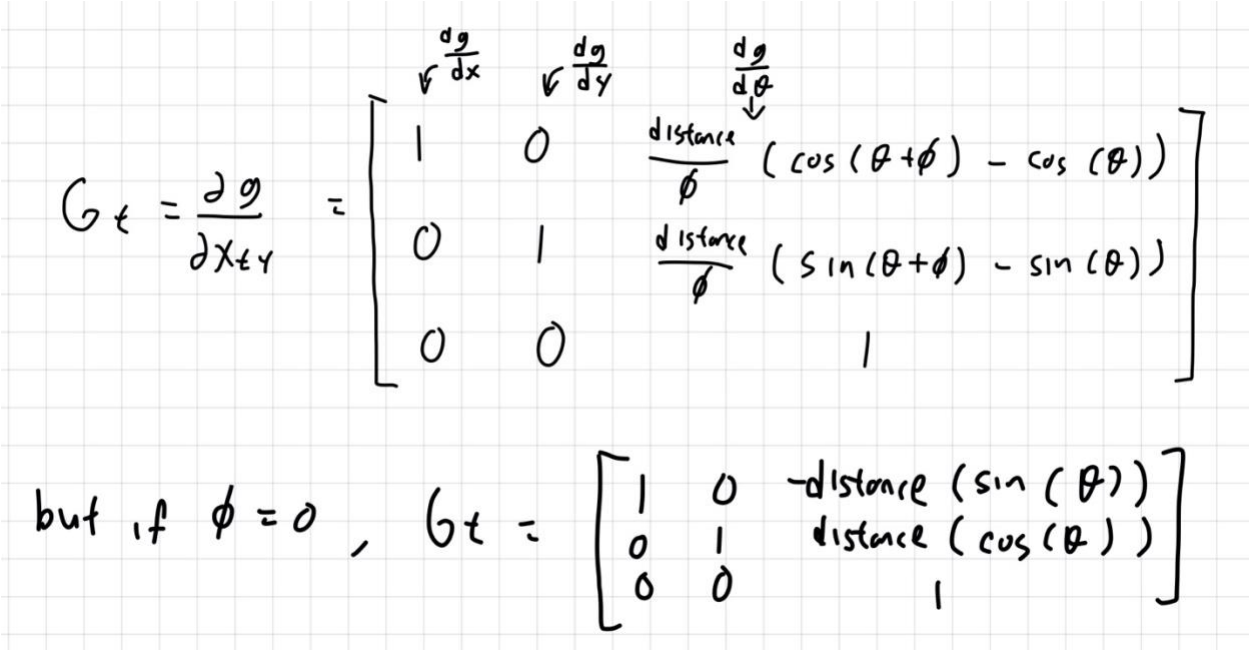
$$\text{but if } \phi = 0, \quad \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} x_{t-1} + \text{distance}\cos(\theta) \\ y_{t-1} + \text{distance}\sin(\theta) \\ \theta_t \end{bmatrix}$$

2. h(xt) is a function that we use to predict a measurement based on the current estimated robot pose, that is previously estimated using g(). Since the measurement that is given is depth measurement, the function that we can use to predict depth measurement in depthPredict().
3. hGPS is a function that will predict a "GPS" like information, which is a measurement of the true robot pose. If we are given a prediction of current robot pose from g(), then our GPS like prediction will literally be equal to the robot pose. So the function is just multiplying a 3x3 eye matrix with the robot pose.

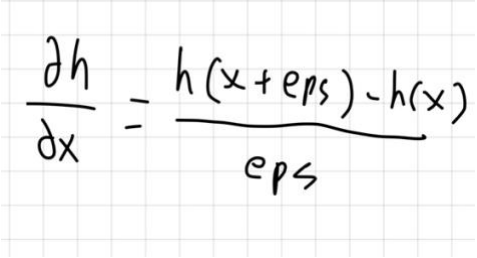
Measurement prediction of GPS, given pose $\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

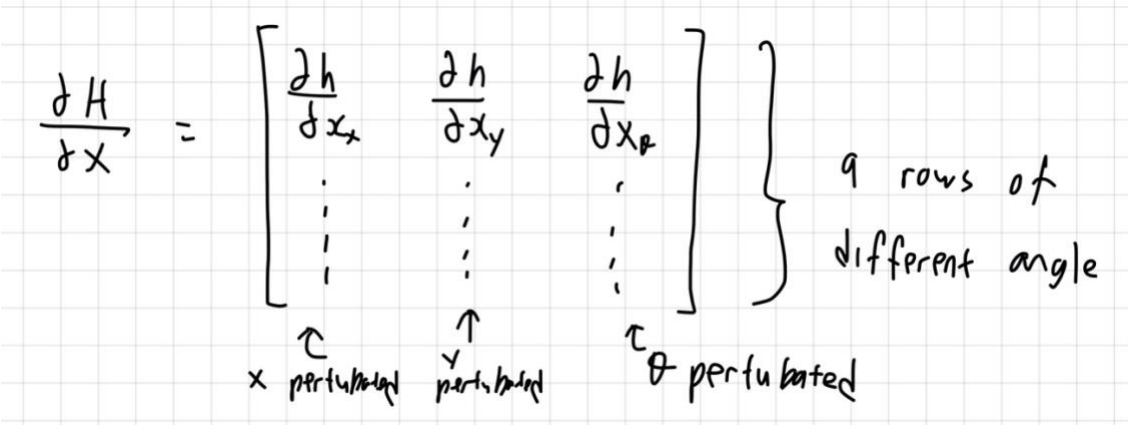
4. Formula of jacobian of g:



5. I agree that differentiating the depthPredict function will be difficult. So, I calculated the finite difference instead. For the jacobian matrix, we have a 9x3 dimension. There will be a calculation of the depth prediction using pertubated value of a pose, subtracted by the depth prediction of the original value of the pose, divided by the amount of perturbation. It can be described using the formula below:

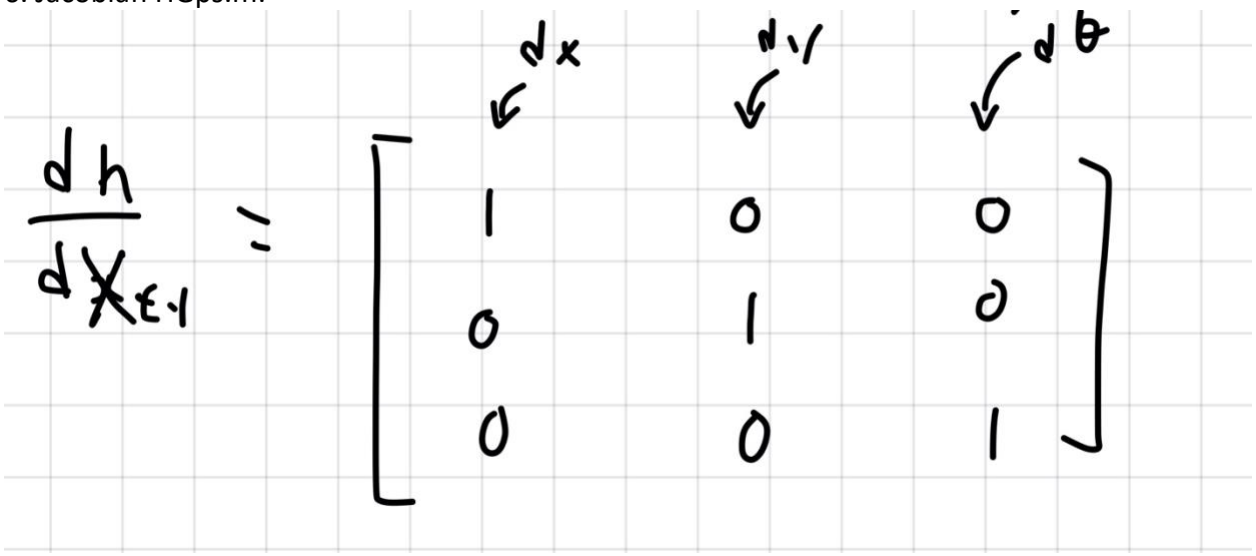


At each column, only one of the pose will be pertubated. Each of the row shows the different angles of the depth predict.



Calculating this difference allows us to get a similar result to differentiating it.

6. Jacobian HGps.m:



Extended Kalman Filter

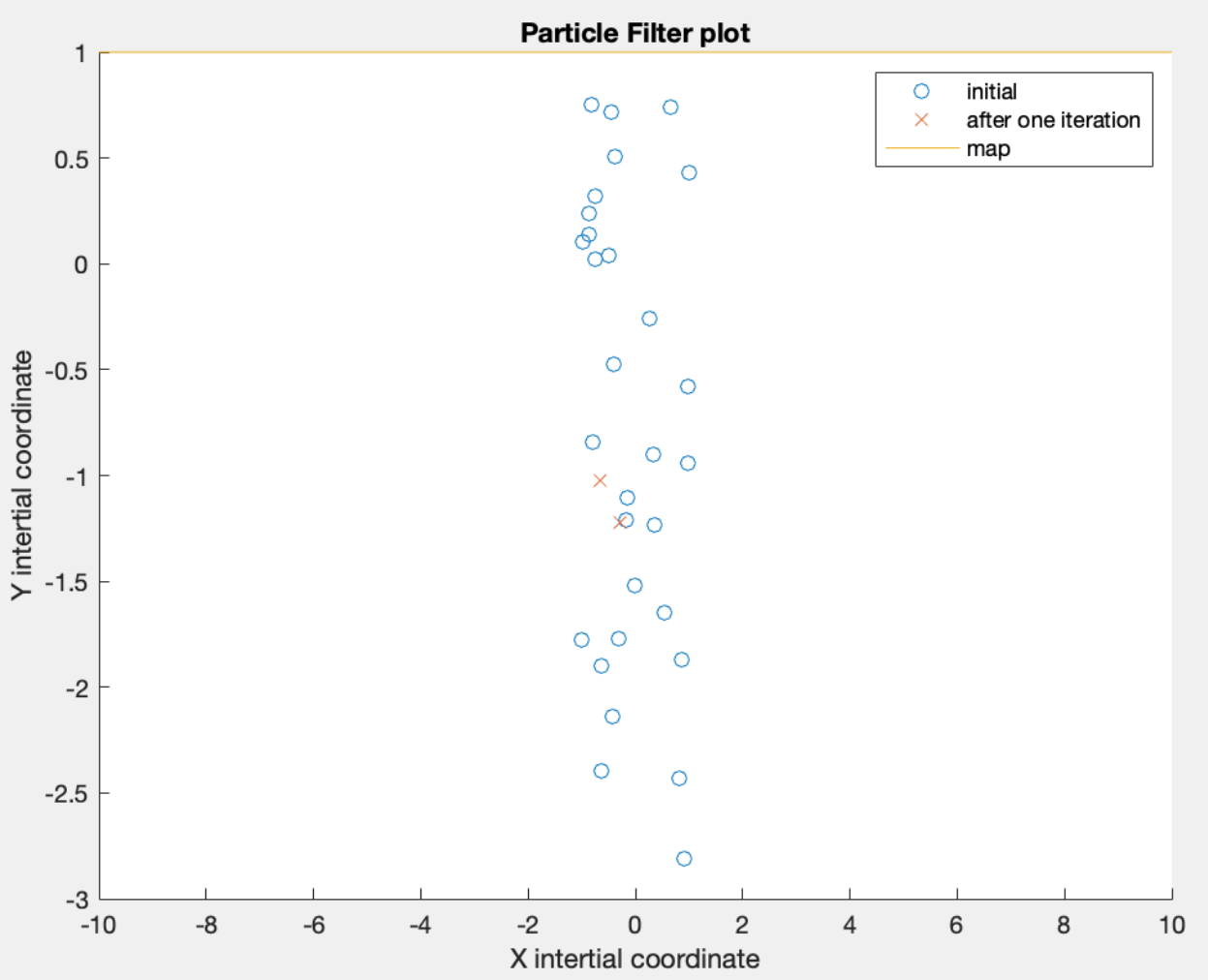
The inputs are as below:

1. Mu_prev: the pose of the robot from previous time
2. U_prev: the control odometry of the robot
3. Sigma_prev: the variance of the pose from previous time
4. Z: the sensor measurements
5. R: the matrix of dynamic or process noise
6. Q: the matric of sensor measurement noise
7. g: the input is a function to predict robot pose based on control
8. Gjac: the function of a jacobian of g function for EKF calculation.
9. H: the input is a function to predict sensor measurement based on predicted robot pose.
10. Hjac: the function of a jacobian of h function for EKF calculation.

The outputs are as below:

1. Mu: the predicted pose of the robot
2. Sigma: the variance of the predicted pose of the robot.

Particle Filter

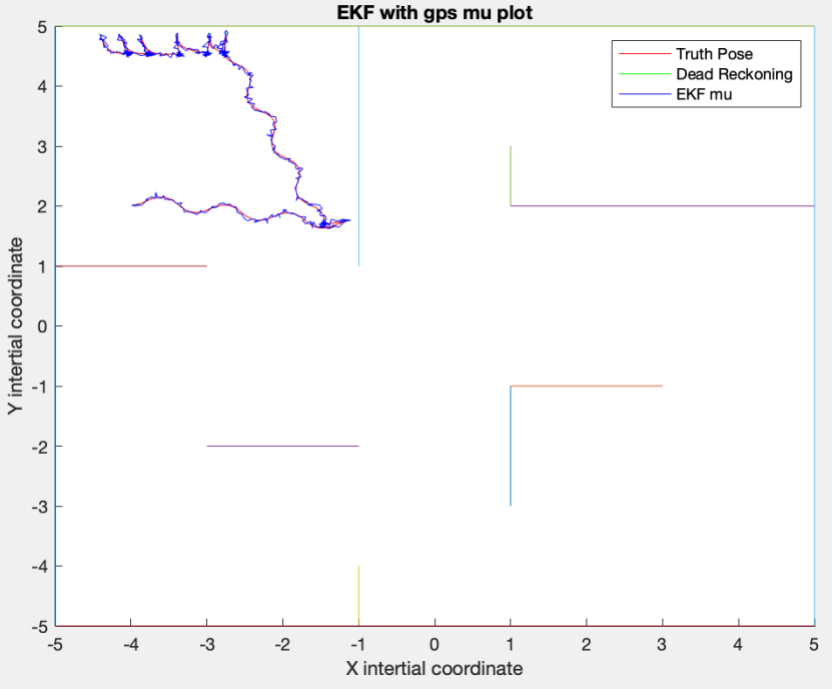


We can see the line of the map in yellow at the top of the plot. There are 30 initial particles shown in blue circles. There are only two location of the 30 particles left shown in red 'x', just after one iteration.

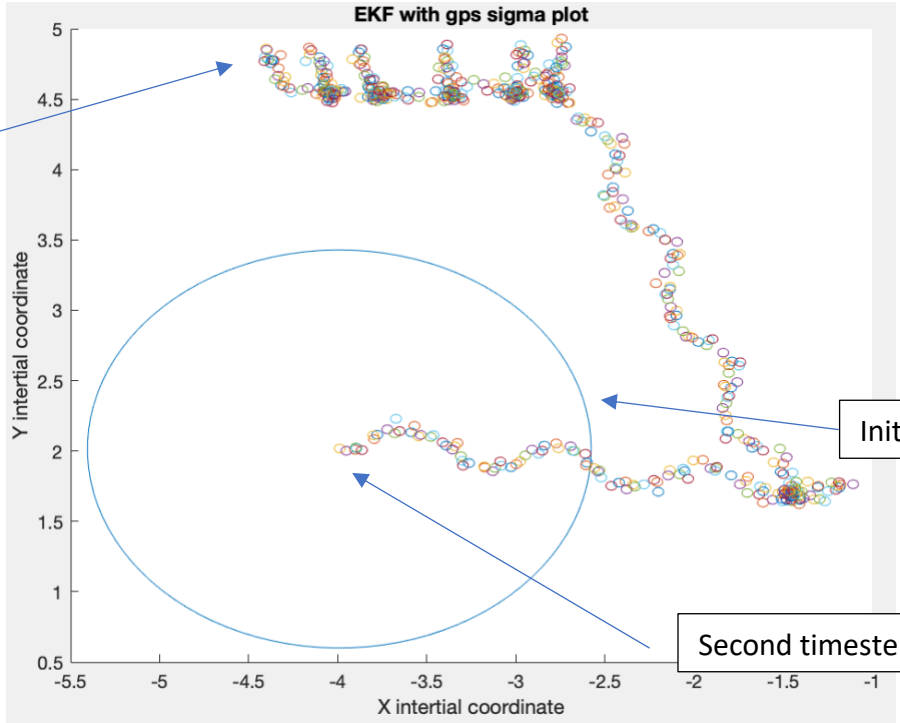
Simulation

3. The R has a 3x3 dimension as it is related to the robot pose. The Q dimension is based on the sensor measurement. For GPS, it will be 3x3 due to its relation to robot pose. For Depth it will be 9x9, as each depth measurement consist of 9 measurements.
4. Below are the answers:
    a. The GPS noise is related to sensor noise, which has the variance value of 0.001. So, the standard deviation is sqrt(0.001) = 0.03162. The GPS sensor noise distribution is (0,0.03162) with the format of (mean, std).
    b. For the odometry noise, since we are given a process noise variance R of 0.01, the mean is set to 0 and the std is set to 0.1. For the rsdepth noise, since we are given a process noise variance Q of 0.001, the mean is set to 0 and the std is set to 0.03162.
    c. No answer needed

d. Plot with initial belief of true initial pose and sigma of [2,0,0;0,2,0;0,0,0.1]



Note that the dead reckoning trajectory has the exact location to truth pose, and its green line is under the red line. We can see that the calculated EKF mu is pretty similar to the truth pose trajectory, which is great!



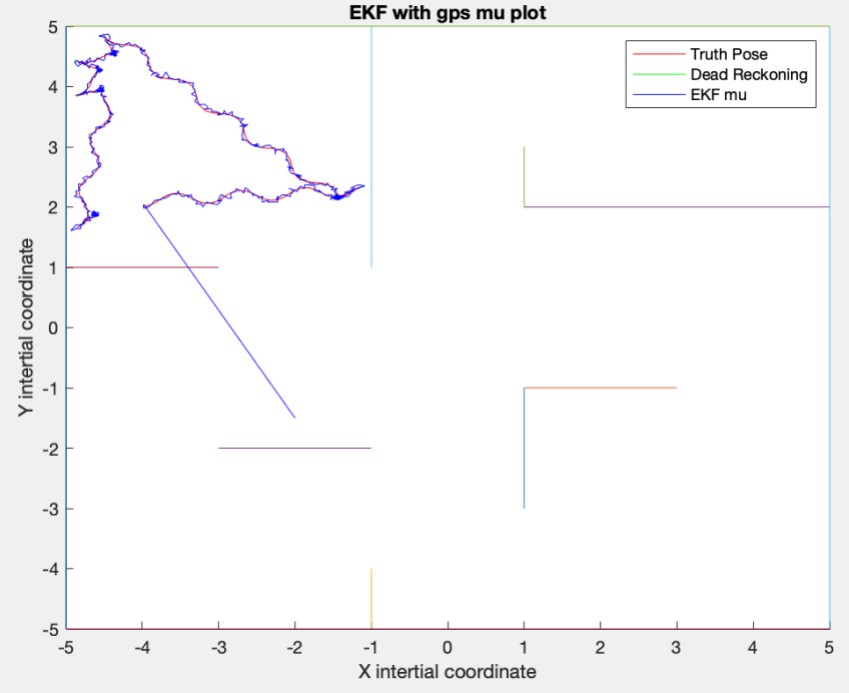
Final prediction

Initial prediction

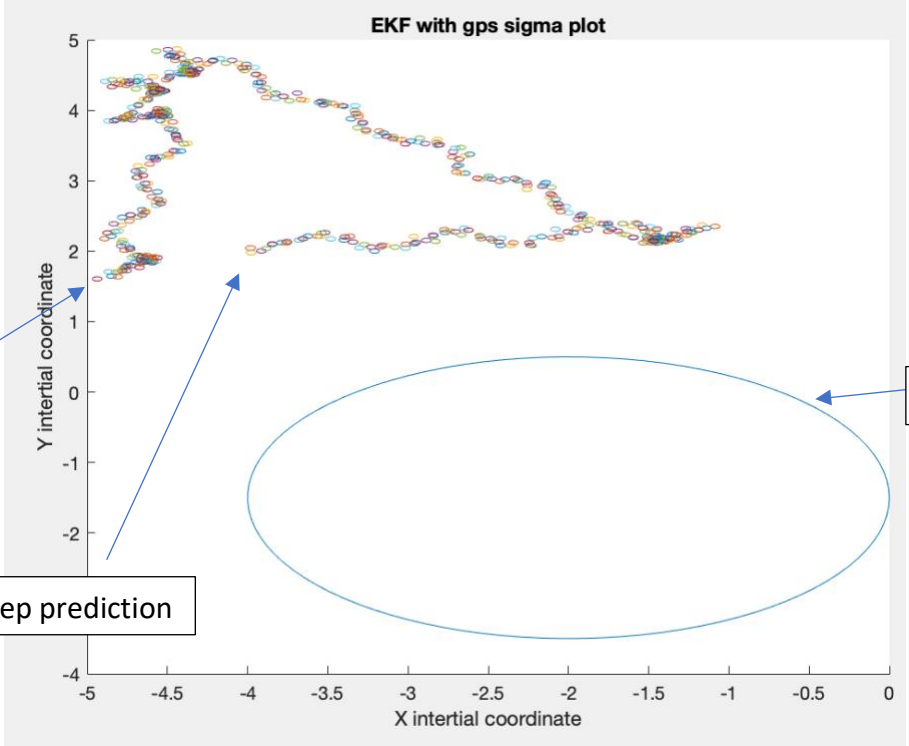
Second timestep prediction

A legend is not included as it will only confuse the reader. We can see that the initial prediction has a large variance, but then quickly shrinks through the timesteps. The variance seemed to be consistently small through the timesteps.

e. Plot with initial pose belief of [-2,-1.5,pi.2]and sigma of [4,0,0;0,4,0;0,0,0.02]



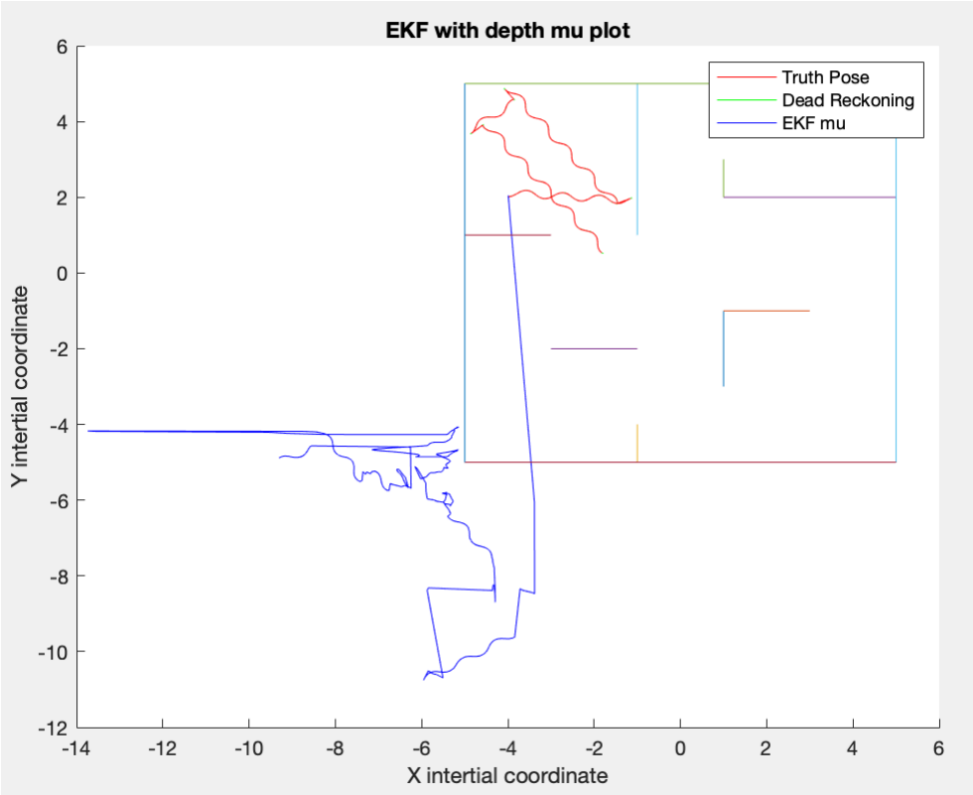
In this plot, we can see the jump of the EKF mu trajectory from the initial belief around (-2,-1.5) to (-4,2). The EKF mu then has a similar trajectory to truth pose.



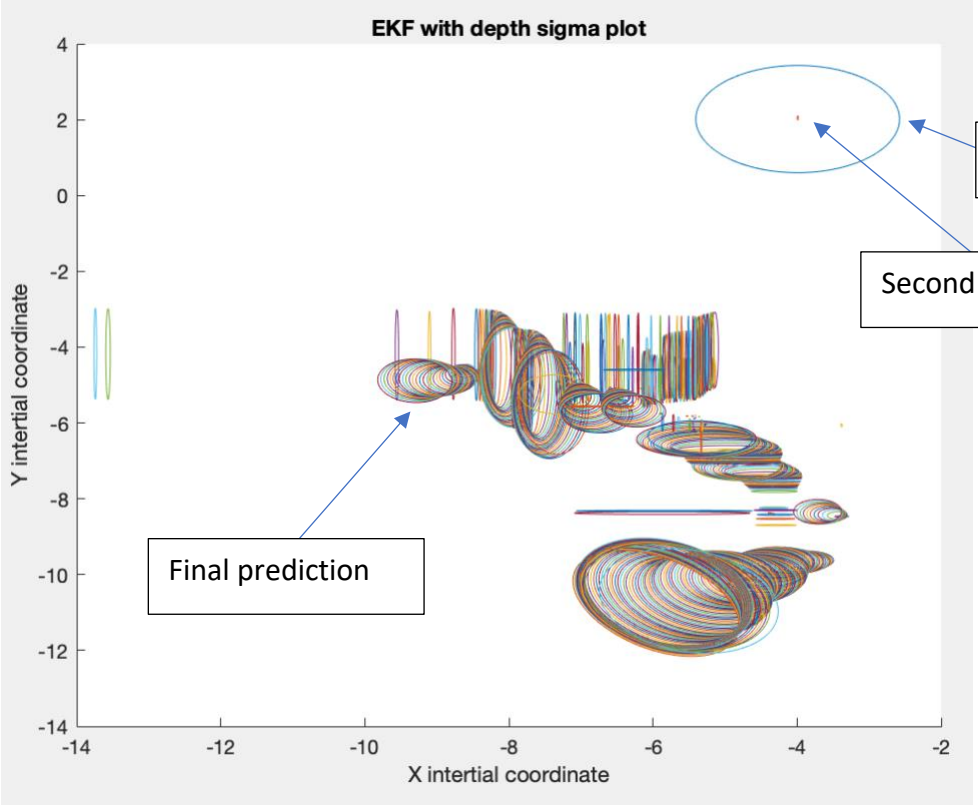
A legend is not included as it will only confuse the reader. We can see how the mean of the initial and second prediction is different. The variance of the initial is also way larger than the other predictions after it.

f. The difference between (d) and (e) is only the initial belief of the pose and its variance. Based on the result above, the initial belief does not really affect the final measurement. In both cases, the predicted robot pose from using EKF is very similar to the truth pose of the robot. This is actually one of the benefit of using EKF, where the initial belief does not really have to be accurate to the real pose, because the update step will fix it based on sensor measurement. Looking at the second plot in f about the variance, we can see that there is a sudden jump of location from the initial location, to one time interval after that. This shows how quickly the algorithm fix the pose prediction in the update step.

5. (a) Plot for EKF with depth data



In the plot above, the dead reckoning trajectory is right below the truth pose trajectory, as they have the same value. We can see that the EKF mu trajectory above is way off from the truth pose trajectory. This is caused by the EKF update stage that is trying to update the pose prediction based on the real and expected measurement. What happened is that due to noise, the sensor measurement and the calculated depthPredict() function detected different walls to calculate the depth. In this scenario, at initial time, the robot is at (-4,2) and facing the 0 degree direction. The depth sensor at an offset of -27deg is still detecting the wall at (-1,1). However, due to noise, the depthPredict() is calculating to the wall that is further on the map at (1,-1). The huge difference in measurement causes a huge impact in pose prediction. This error is carried over throughout the run and kept getting propagated. As a result, the entire predicted trajectory information is bad.

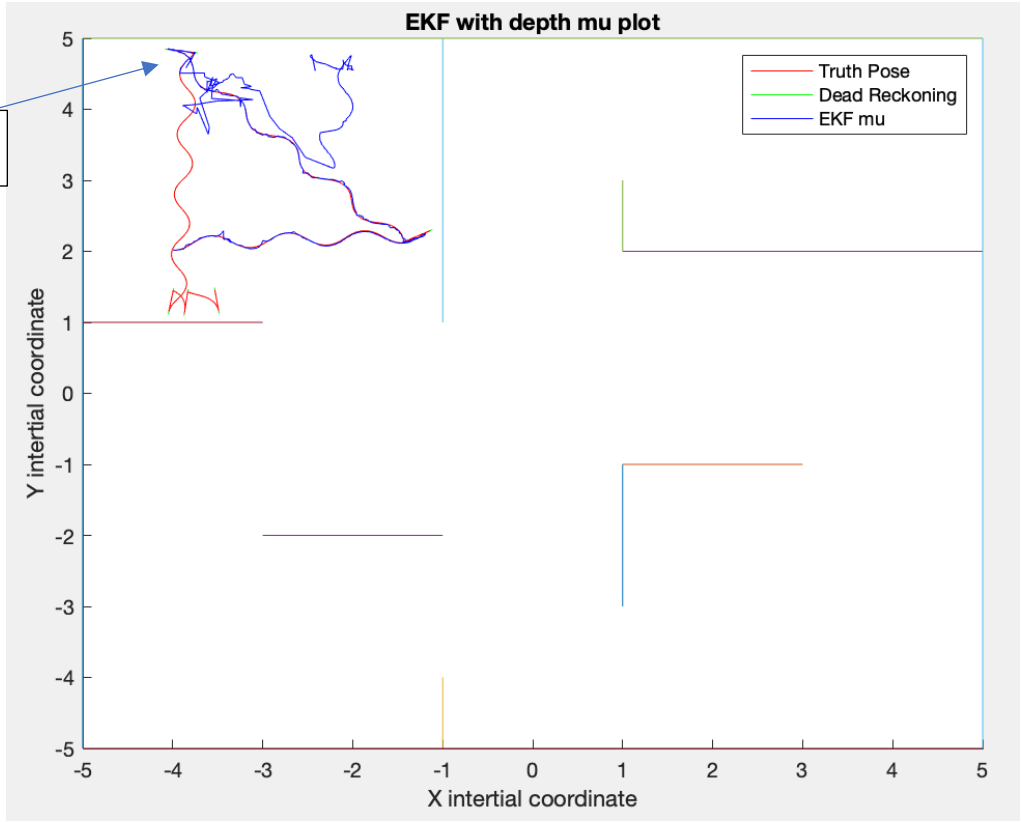


Note: legend is not included to prevent confusion, labels are used instead.
The sigma plot is also plotted. We can see that the calculated mean is jumping all over the place, and with pretty random sized variance. There is no meaningful information that can be learned, aside from that this is a bad plot. This shows one of the limitation of the EKF when used in a map with discontinuous walls. Due to noise and imperfect robot pose prediction, it will be having trouble when comparing measurements and updating the predicted pose, as the measurements may be detecting different walls.

In order to fix this problem, I added a few lines of code that will try to ignore the comparison of specific angle measurements when it is detecting two different walls. This idea is still pretty rough and imperfect, but this is done by having a threshold for the ratio of the difference between the two measurements with the expected(calculated) measurement. This condition is only applied to when the robot is not close to the wall, as the depth measurement becomes more sensitive when it is small.
The condition is as below, where "z" is the sensor measurement and "expected" is the expected calculated measurement.

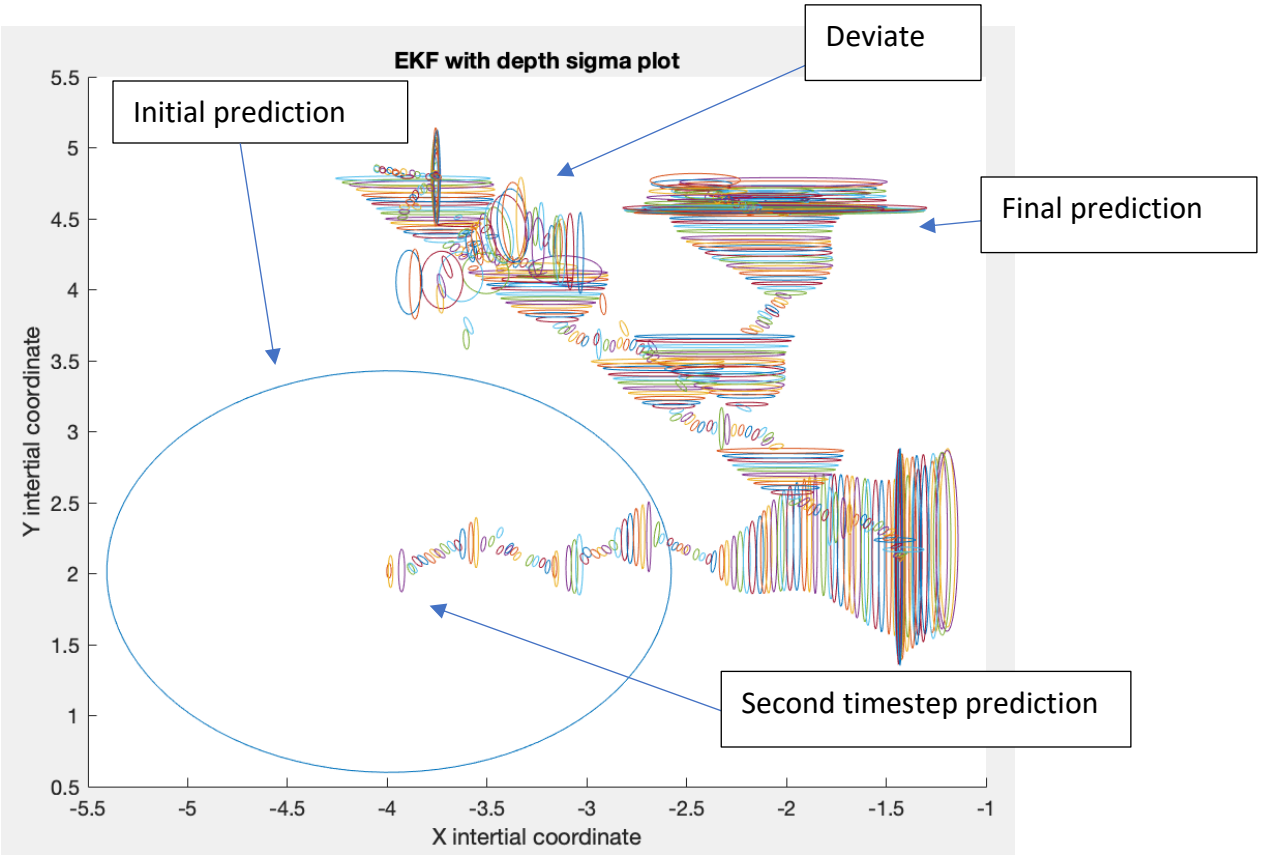
```
if (z(i) >= 0.3 && abs((z(i)-expected(i))/expected(i)) < 0.2) || z(i) < 0.3
```

Shown below is the plot with the condition:



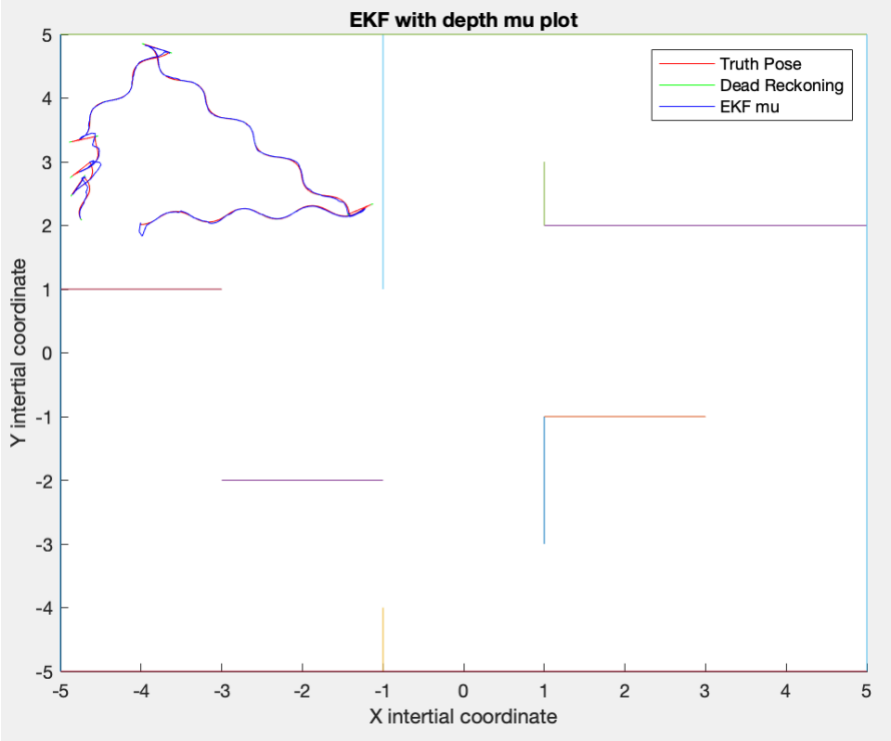
Again, the dead reckoning plot is right under the truth pose trajectory. We can see that as the robot starts from (-4,2), the EKF prediction is pretty accurate for about two-thirds of the run. However, at the pose pointed with the arrow, we can see that the prediction started to deviate. It is likely that this is again caused discontinuous wall problem and sensor noise.

We can see from this plot that the added condition helps improve the errror in calculation caused by the discontinuous wall, however it is not perfect. It will be extremely difficult to optimize this EKF program that can solve this discontinuous wall problem. Perhaps EKF may not be the best choice.

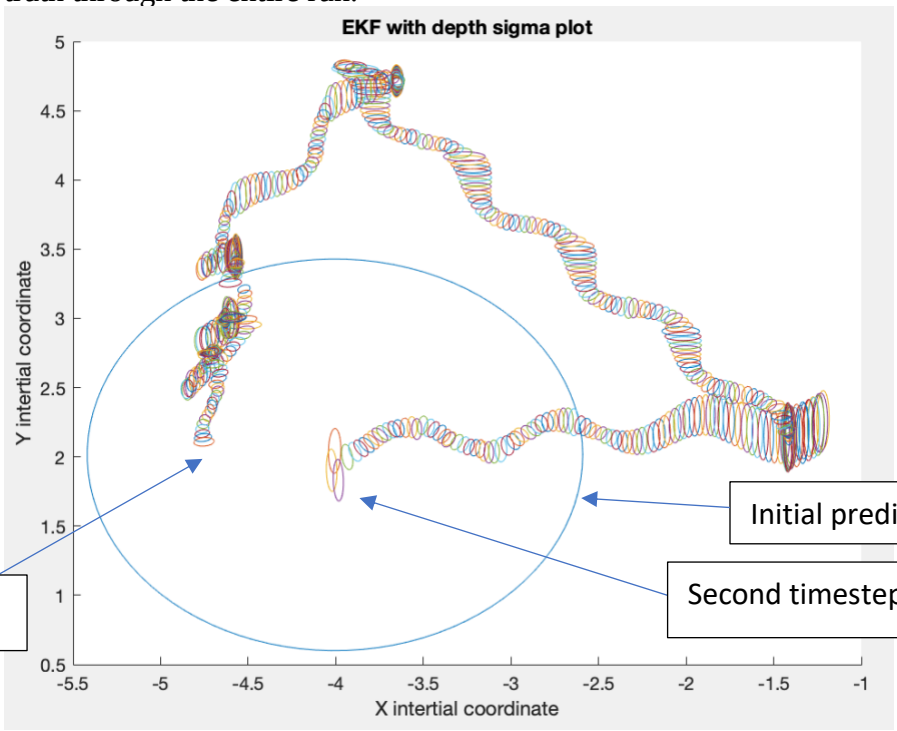


From the variance plot above, we again see how the variance of initial belief is the largest. Then throughout the timestep as the prediction and update happens in the EKF, the variance grew and shrinked throughout the trajectory. Unlike the previous variance plot, we can sort of see how each of the variance is connected, which describe the predicted trajectory of the robot. However, there is a point where the prediction deviates from the actual robot pose, as labeled above.

1. Now, the process noise is decreased from 0.01 to 0.001. The sensor noise is increased



From the plot above, we can immediately see how the EKF prediction is very close to the truth through the entire run.



We can see the trajectory of the robot through the prediction. The variance of each pose prediction throughout the run is also much lower compared to earlier plot.

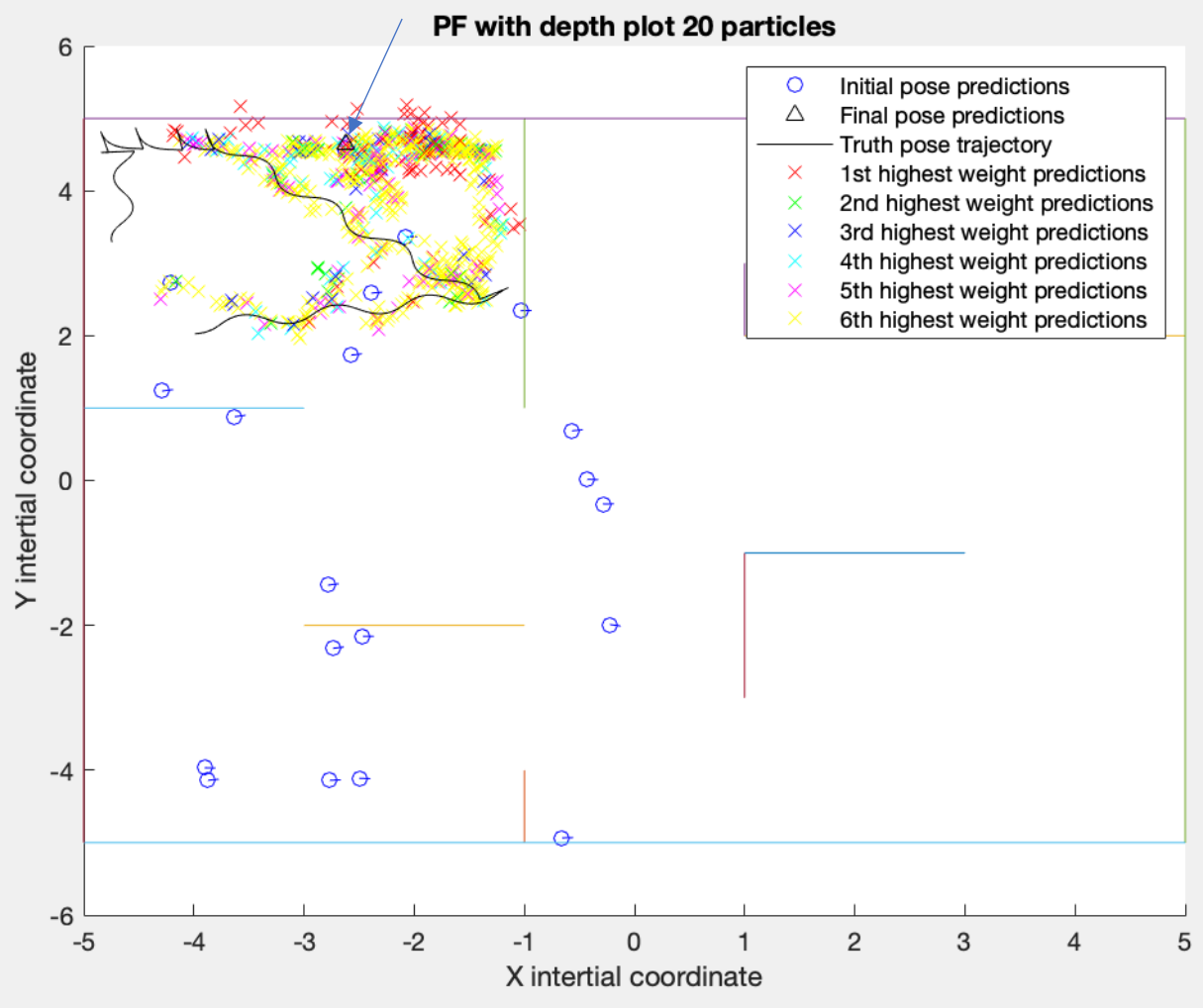
(c) With this change in noise, we can see major difference between the first EKF plot and second EKF depth plot. There are 2 things that changes: an increase in process noise and a decrease in sensor noise. From comparing the mu plot, we can see a huge improvement in pose prediction accuracy where with a lower process noise and higher sensor noise. From the sigma variance plot, we can see that with a lower process noise and higher sensor noise, the mean value at each of the timestep seemed to flow nicely from one to another, without any sudden huge deviation in mean value between timesteps. The variance of the prediction at each timestep is also kept low throughout the run. Based on this analysis, we can conclude that the process noise covariance would greatly affect the robot estimate. We would have to try to keep it low in order to have a good estimate. On the other hand, the measurement noise covariance seemed to have a minimum impact on the robot estimate, although preferably we would want it at a low value too.

(d) No, the filter did not behave like I expected. I expected the filter to perform as well as the GPS measurement. However, it turns out that noise and pose prediction inaccuracies may cause major pose prediction error related to wall discontinuities. Once this error happens, the filter also has no chance in fixing this error. It clearly shows one of the weakness of EKF when used for depth prediction. However, I am also surprised how process noise has a major impact to the prediction, as demonstrated in the experiment above.

(e) As seen during the experiment above, the major problem is on how this scenario is very sensitive to noise and inaccuracies. One of the case is that we may be comparing the depth of different walls when comparing the expected depth calculation with the actual depth. The initial pose or the predicted pose may be slightly different due to some noise, which would then impact the directions in which the depth sensor is facing. When the difference between the depth measurements are small, the EKF algorithm will be “fixing” the pose prediction of the robot during the update step, to be more similar to the true pose. However, there is a huge difference in depth measurement that is actually caused by measuring different walls, the EKF will take it as a huge “innovation” and will dramatically change the pose prediction. This will instead cause a huge error in pose prediction. The worse problem is that when the pose prediction is this far off, there is no way for the EKF algorithm to fix it. So, I think the main problem is EKF would be too sensitive to noise when used for depth measurement (in a map with discontinuous walls), and that once a huge error happen in the predicted pose there is no way to fix it.

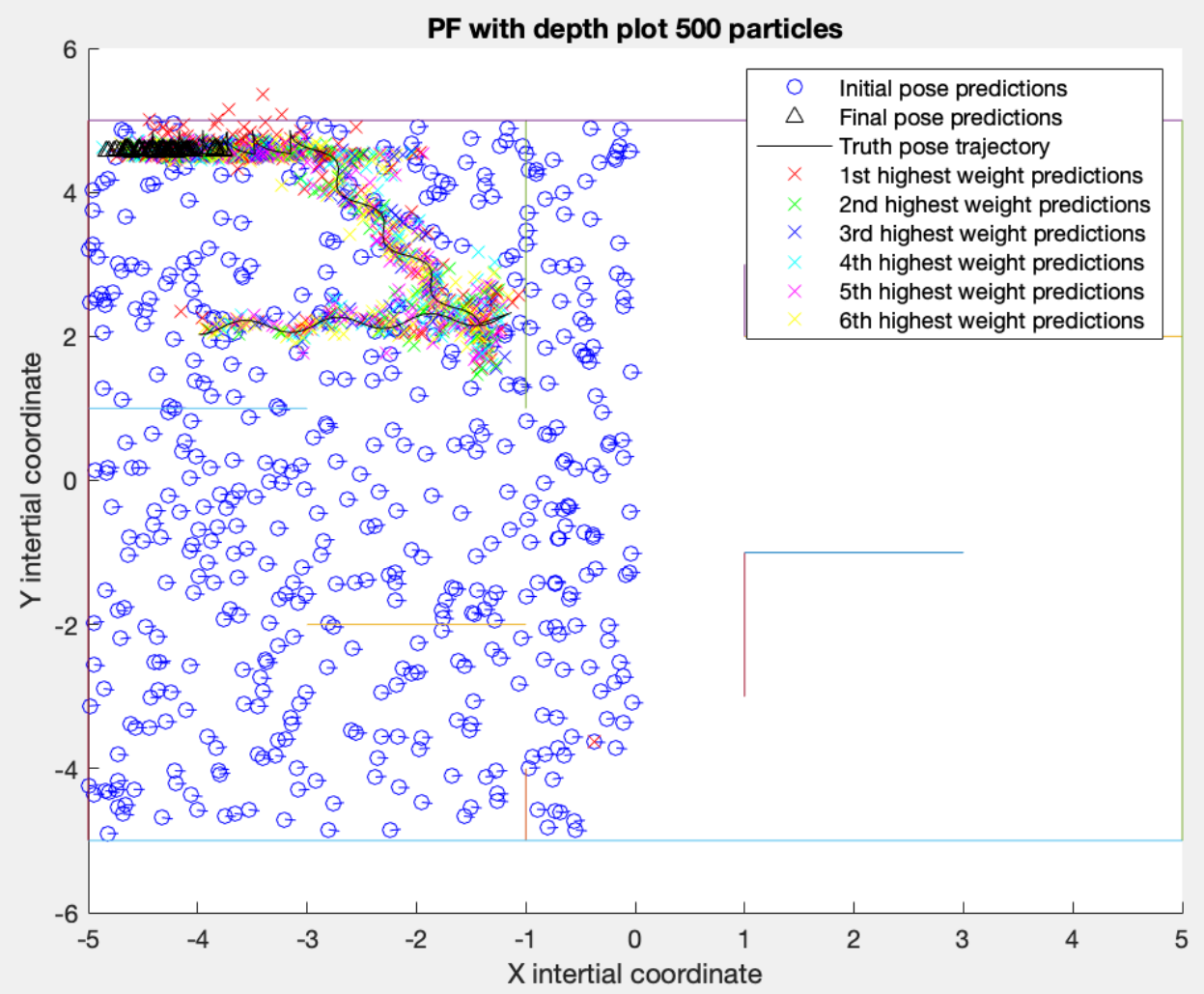
(f) The error that we will be seeing with the physical robot will be the ones demonstrated above. I think that the process noise on the physical robot will be pretty big. This will cause some amount of errors in the preliminary prediction of pose (mu_bar) in the prediction step of EKF. With this error in pose, the expected depth measurement made will also be quite different from the real measurement. Since the lab has discontinuous walls, with section of walls throughout the map, there is a huge chance that the actual and expected measurement will be comparing depth calculations of different walls. As a result, we will have similar errors to what was demonstrated above. We may end up getting a bad pose prediction trajectory. Some of the errors may be attenuated with additional condition that I have added in my code, however errors will still be seen.

6. Plot with 20 particles:

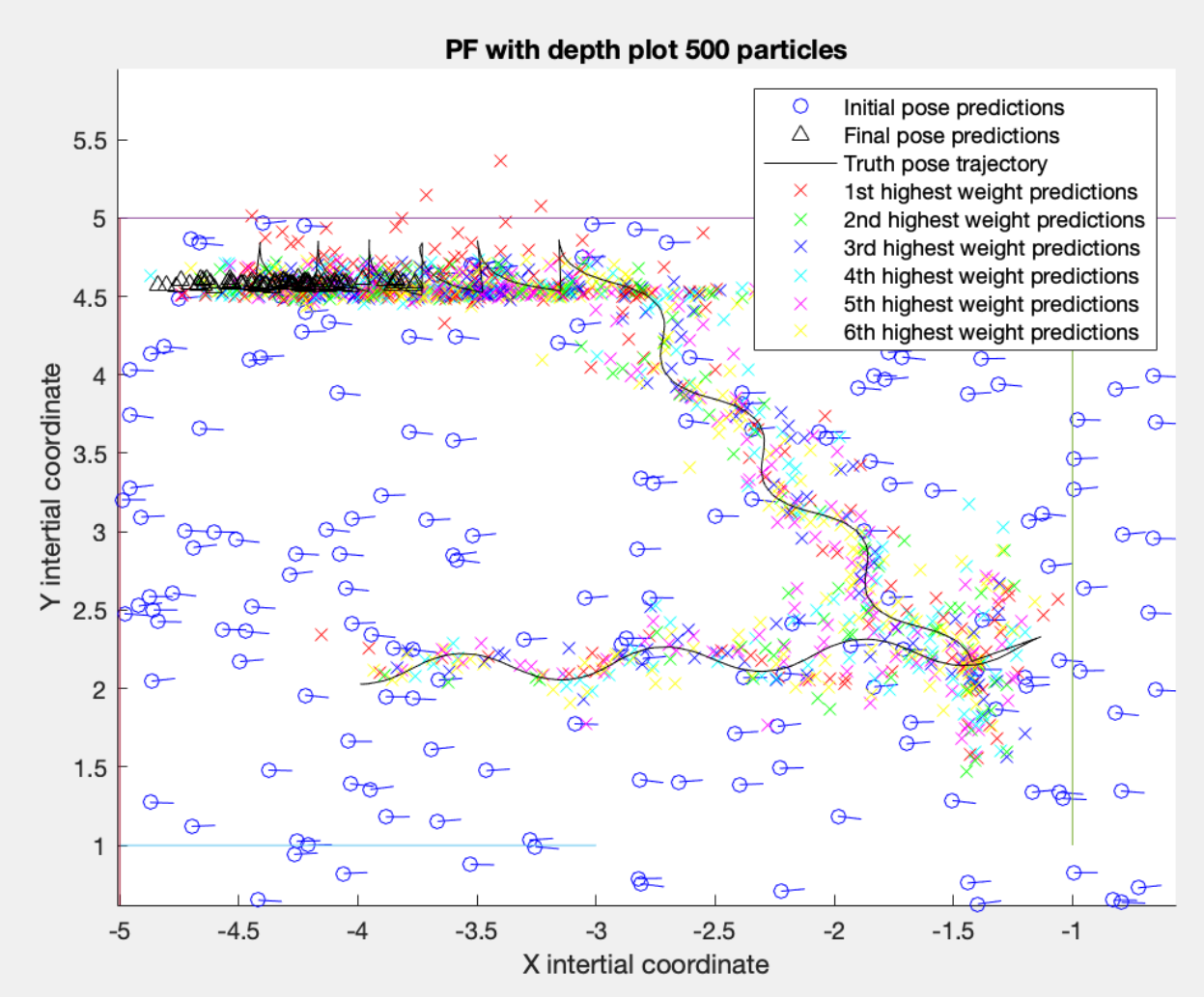


The true trajectory is shown with the black line. We can see that the top weighted predictions shown with 'x' are mostly far from the truth pose trajectory. The final pose (pointed with the blue arrow) is also far from the end of the true trajectory. This is because the 20 initial beliefs are pretty far from the true initial pose (-4,-2).

Below is the plot for 500 particles:



Due to the amount of plots, it is rather difficult to see. One obvious thing is the 500 initial pose predictions that we now have. Refer to the next figure of the zoomed in plot.

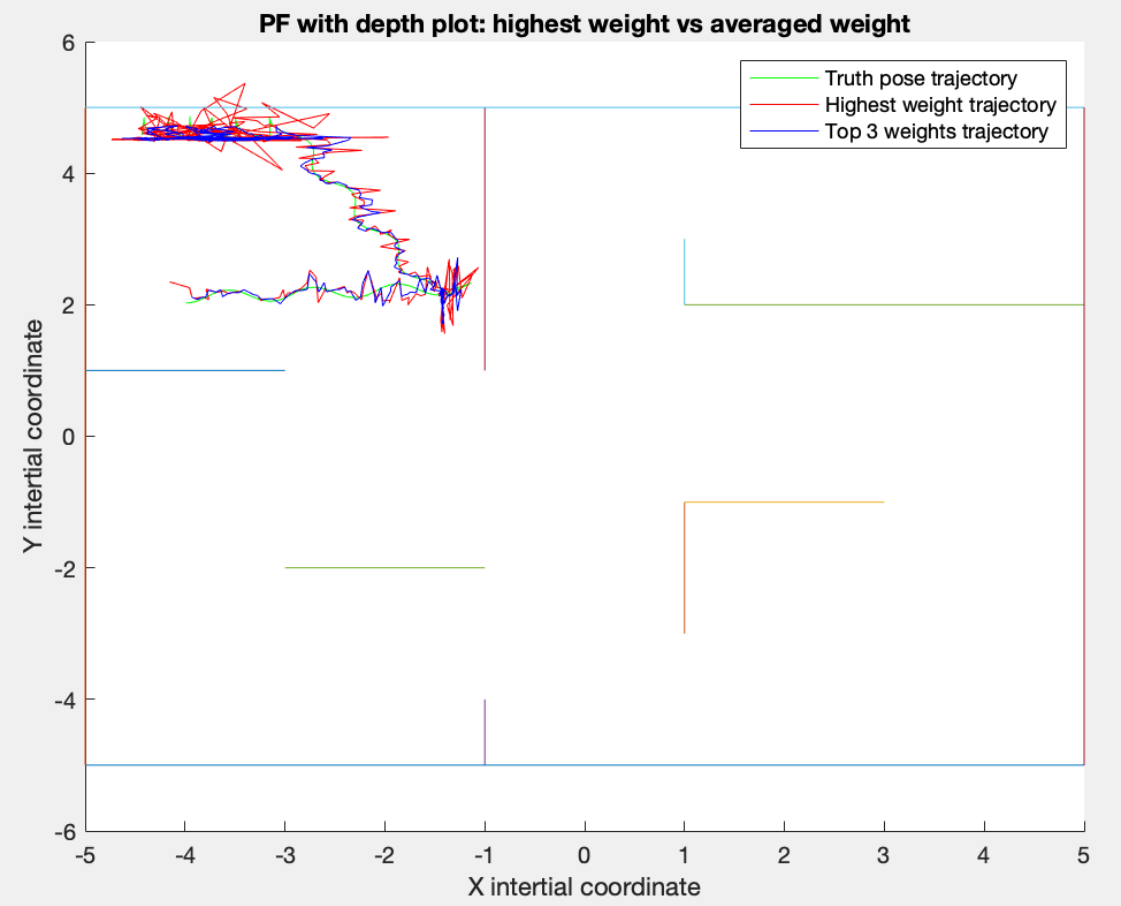


In this plot with much more number of particles, we can see that the top weighted predictions shown in 'x' are mostly pretty close to the true pose trajectory shown with the black line. Due to the amount of plots, the end of the trajectory is difficult to see. However, we can see that the final pose prediction and the end of the true pose are both on the top left region of the map, consistent with each other. The reason that we now have much better predictions is that we have more particles. With more particles, there is a higher chance too that at least one of our initial pose belief will be close to the truth. The downside is that with more particles, it requires more computation power.

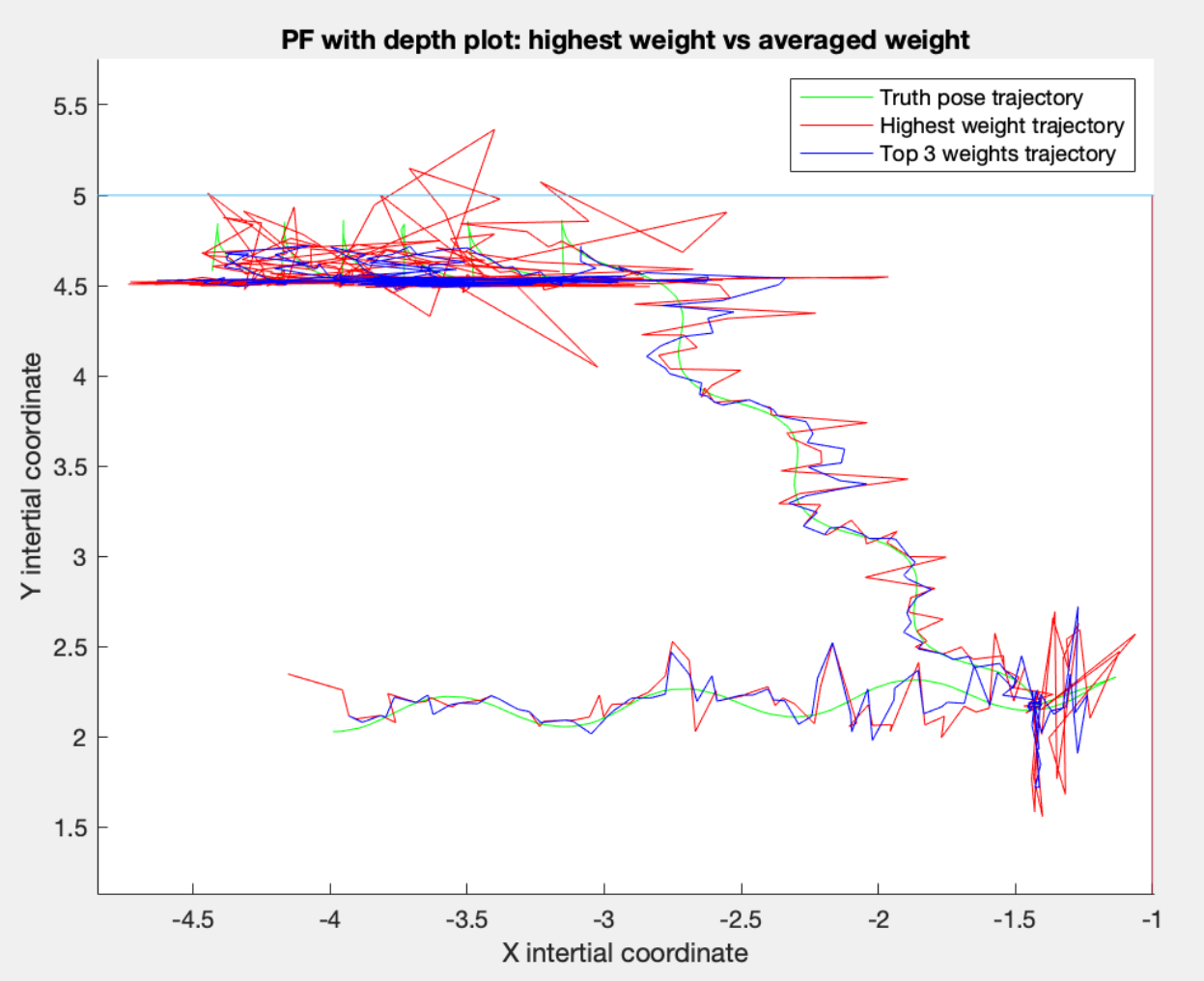
f) Based on the two plots above, the filter behave like I expected. I expected that with more number of particles, the prediction throughout the timesteps will be close to the true trajectory. The final beliefs are also very close to the final true pose. I also expected that when the number of particles are low, we will not get a reliable result. However, I did not expect that using 20 particles would provide a result that is this far off.

g) With more number of initial particles, there is a higher chance that the initial pose of the particles would be close to the actual pose of the robot. As a result, throughout the timesteps and during the final time, the predictions will be pretty close to the truth. The particles should converge to the right regions of the robot's true pose. During each timestep, we may have multiple possible high-weighted pose to choose from, which would allow us to perform averaging or other processes to get better results. Overall, more particles allows to have a more accurate prediction of the robot pose. When the number of particles are small, it may be the case that all initial pose of the particles are nowhere close to the true pose. Thus, the final prediction may be far off, and may converge to the wrong regions. It may also be the case that at the end of the run, there is only a small number of unique pose available from the final particles set, and these unique pose are all wrong.

h) Another option that we have is that instead of choosing the pose with the highest weight, we could average the pose between 3 highest weighted pose. Below is a plot that compares the trajectory of the truth pose, the highest weighted prediction, and an average of 3 highest weighted:



A zoomed in plot is included below for visualization:



The trajectory starts at (-4,-2). We can ignore the top part of the plot, as the overlaps of the line based on the trajectory may be confusing. We are comparing on how similar the red line (highest weight) and blue line (average of top 3 weights) are to the green truth pose line. In most cases, the blue line is closer to the green line, when compared to the red line. This shows that averaging of several highest weights, in this case 3 weights, would help get a better prediction and closer to the truth.

Comparing Filters

Kalman Filter:
Assumptions: The system is linear, gaussian distribution of noise.
Appropriate when: Linear system, such as a stationary robot.
Advantages: -Quick computation.
- Pretty accurate result, given how simple it is to use.
Disadvantages: -Limited application, can only be used on a linear system, while real-life situation is almost always non-linear.

Extended Kalman Filter:
Assumptions: gaussian distribution of noise.
Appropriate when: The system is non-linear but is gaussian (noise), can be used for moving robot, but preferably in a map with continuous walls.
Advantages: -May be used in a non-linear system (unlike Kalman Filter).
-Initial belief may not affect prediction (only in some cases like the GPS, but not for depth measurement).
-Sensor noise has minimum impact on estimates.
Disadvantages: -Performance may depend greatly on process noise, large noise -> inaccurate.
-Involve linearization and approximation, which means it has some errors, and may give a very bad prediction when linearized at the wrong value

Particle Filter:
Assumptions: System is a Hidden Markov model, where there is observation/evidence presented at each time step in order to predict the future states.
Memoryless system, where future states are based only on current state and not the past.
Appropriate when: Can be used anytime, both linear and non-linear system, gaussian or non-gaussian noise. It works well for robot localization in maps with discontinuous walls (better than EKF). Would be appropriate if we do not know the initial position of robot, and so we can use multi-hypothesis by using a big number of particles across a large area as initial particles.
Advantages: -May provide very accurate prediction (when using a lot of particles).
-Freedom to control accuracy (given a strong computation power)
Disadvantages: - Particle deprivation may happen.
-Initial belief affects the final prediction or result. If initial belief is far from true pose, the estimate will be bad.
- Need a strong computation power in order to perform well due to a tradeoff between computation vs prediction accuracy.
- Non deterministic as it produce different outputs depending in random initial belief.