EECS 495 Homework 0

D. Harmon Pollock

October 2014

Assignment: Particle Filter with Data Set 1

1)[UKF, PF] Design a simulated controller to estimate the position (2D location and heading) changes of a wheeled mobile robot in response to commanded speed (translational and rotational) controls. Clearly define the inputs, outputs and parameters of this model, and report the math. Explain your reasoning, as needed.

My simulated controller uses the following:

Inputs:

- A prior state estimate vector:

$$[x_{t-1}, y_{t-1}, r_{t-1}]$$

- Translational Speed Command, v
- Rotational Speed Command, ω
- Time Duration, d

Outputs:

- A Current State Estimate Vector:

$$[x_t, y_t, r_t]$$

This simulation uses 3 equations to model the changes in the state estimates:

$$x_t = x_{t-1} + \int_{t=0}^{d} v\cos(r_{t-1} + \omega t)dt$$

$$y_t = y_{t-1} + \int_{t=0}^{d} v\sin(r_{t-1} + \omega t)dt$$

$$r_t = r_{t-1} + \int_{t=0}^{d} \omega dt$$

I also tried the following equations as found in Probabilistic Robotics which turned out to be equivalent.

$$x = \begin{cases} x_{t-1} + v\cos(r_{t-1} + \omega\Delta t)\Delta t & \text{if } \omega = 0 \\ x_{t-1} - \frac{v}{\omega}\sin(r_{t-1}) + \frac{v}{\omega}\sin(r_{t-1} + \omega\Delta t) & \text{if } \omega \neq 0 \end{cases}$$

$$y = \begin{cases} y_{t-1} + v\sin(r_{t-1} + \omega\Delta t)\Delta t & \text{if } \omega = 0 \\ y_{t-1} + \frac{v}{\omega} * \cos(r_{t-1}) - \frac{v}{\omega}\cos(r_{t-1} + \omega\Delta t) & \text{if } \omega \neq 0 \end{cases}$$

$$r_t = r_{t-1} + \omega\Delta t$$

This simulated controller code can be found in the function: *simulatedControllerEstimate* in *models.py*

---

2) Verify your simulated controller on the following sequence of commands. Report the values of any parameters. (Below, v is translational speed, ω is rotational speed, and t is the duration of the commands.) Report the resulting plot.

[v=0.5m/s,ω=0rad/s, t=1s]

[v=0 m/s, $\omega=\frac{-1}{2*\pi}$ rad/s, t=1s]

[v=0.5m/s,ω=0rad/s, t=1s]

[v=0m/s,$\omega=\frac{1}{2*\pi}$ rad/s, t=1s]

[v=0.5m/s,ω=0rad/s, t=1s]

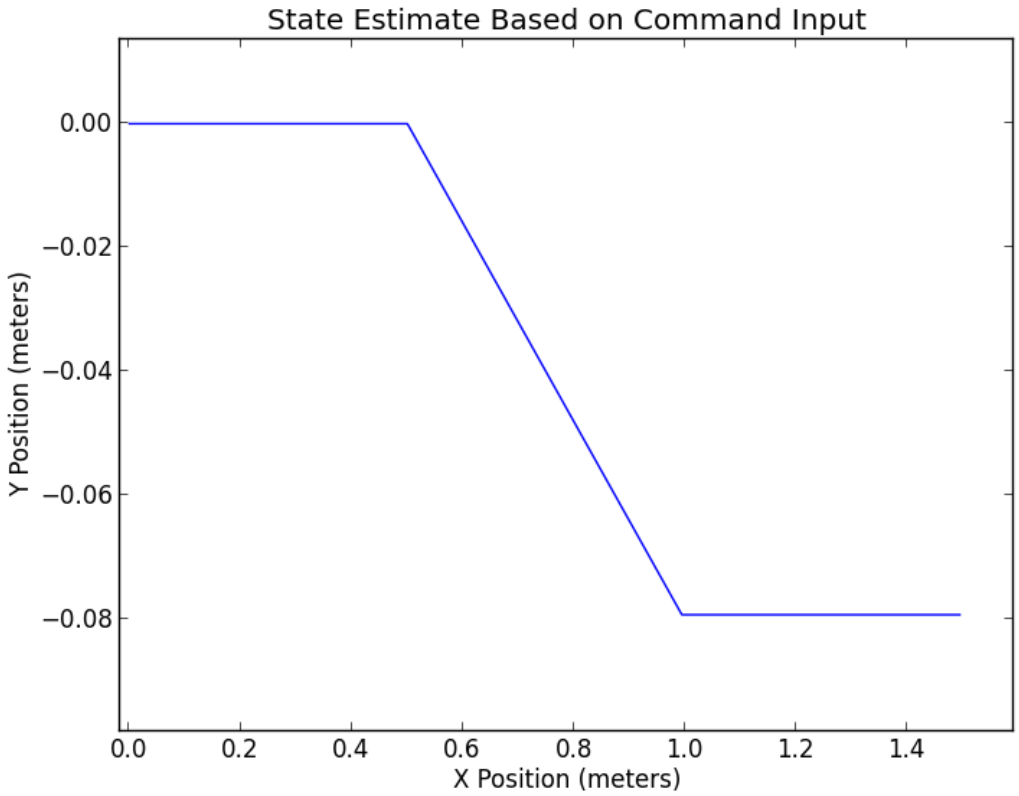

Figure 1: Resulting plot of the listed five inputs

Assumption: The starting location and heading of the robot are assumed to be [x=0, y=0, heading = 0]

---

3) Test your simulated controller on the robot dataset. Issue the controls commands (in Odometry.dat) to your controller, and compare this deadreckoned path (i.e. controls propagation only) to the ground truth path. Report and discuss the resulting plot.

Assumption: The starting location and heading of the robot are assumed to be (x=0.98038490, y=-4.99232180, heading = 1.44849633),

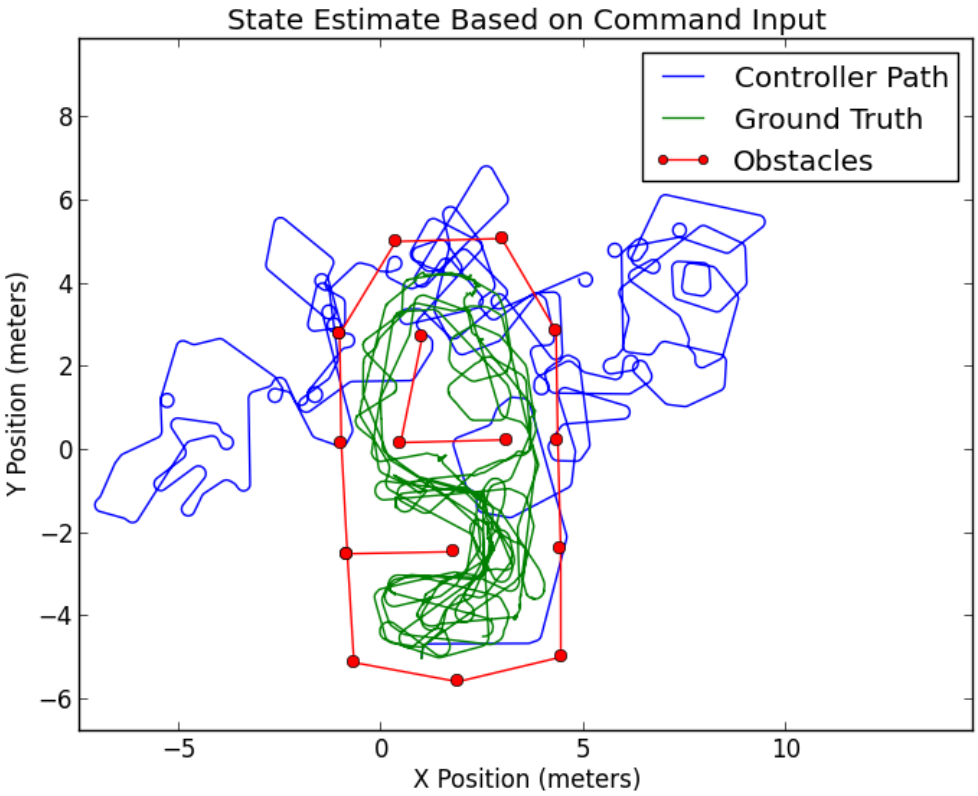

Figure 2: Resulting plot of the listed five inputs

the initial point provided by the ground truth, to make an more accurate comparison.

The green, ground truth path shows very clearly the actual path of the robot navigating around the obstacles in a very ciclic fashion. On the other hand the estimated controller path is much more erratic. The controller path loops on its self very often indicating that it is executing the ciclic behavior seen in the ground truth path but there are external factors such as wheel slippage that it must over compensate for.

---

4) Describe, concisely, the operation of your assigned filtering algorithm. Include any key insights, assumptions, requirements. Use equations in your explanation.

The assigned filter for this assignment is a Particle Filter. State estimation is done by tracking a number of proposed states (particles). Each proposed state or particle individually updated for each control step through the simulated controller shown in question one. Then at each measurement step the particle is assigned a weight/importance factor based on how closely the provided measurement matches the expected measurement. From here the particles are resampled proportionally to their importance factor distribution before repeating the algorithm on the new resampled particles.

---

5) [KF, IF] Design a motion model (starting from your simulated controller) that is appropriate for your assigned filter. Report all maths and reasoning; include images for illustration when appropriate.

Since the particle filter does not require any linearization or special treatment of the state equations, the simulated controller can be directly used as the motion model in the particle filter.

---

6) [KF, IF] Design a measurement model that is appropriate for your assigned filter. Report all maths and reasoning; include images for illustration when appropriate.

The measurement model is closely tied to application format. In this case the measurements are based on the arenas landmarks, providing a distance and bearing to that given landmark location. In the particle filter the measurement update comes in the form of

the updating the importance factor of the given particle, and that is calculated based off the difference of the excepted and given measurements. So the first part of the measurement model, determining the expected measurement looks like.

$$\begin{pmatrix} distance_{expected} \\ bearing_{expected} \end{pmatrix} = \begin{pmatrix} \sqrt{(x_landmark - x_t)^2 + (y_landmark - y_t)^2} \\ actan2(\frac{y_{landmark} - y_t}{x_{landmark} - x_t}) - r_t \end{pmatrix}$$

With these expected measurements we can use them to compute the second part of the measurement update, the importance factor. For this problem, we are assuming the error of the measurement, $Z_{measure}$, has some noise with gaussian distrobution with a covariance R:

$$IF = \frac{1}{\sqrt{2\pi R}} * e^{(\frac{(Z_{expected} - Z_{measure})^2}{2R})}$$

Note on implementation: A major hangup when trying to get my p.f. to converge was based on an error in this step. The measured bearing on an object is measured from $-\pi to \pi$, as is the result of arctan2. In most cases the $bearing_{expected}$ value is find and the p.f. converges well, but there are a few scenarios when the bounds of the $bearing_{expected}$ exceed the $-\pi to \pi$ bounds since the arctan2 value has the estimated heading, r_t subtracted it from it. It is important to make sure this whole formula result is bounded within in the range from $-\pi to \pi$ as well as the difference calculated between the expected and the measured. Figure 3show the difference in convergence.

Both these steps are implemented in *models.py* as their respective functions, *expectedMeasurement* and *getImportanceFactor*.

---

7) [UKF, PF] Implement the full filter.

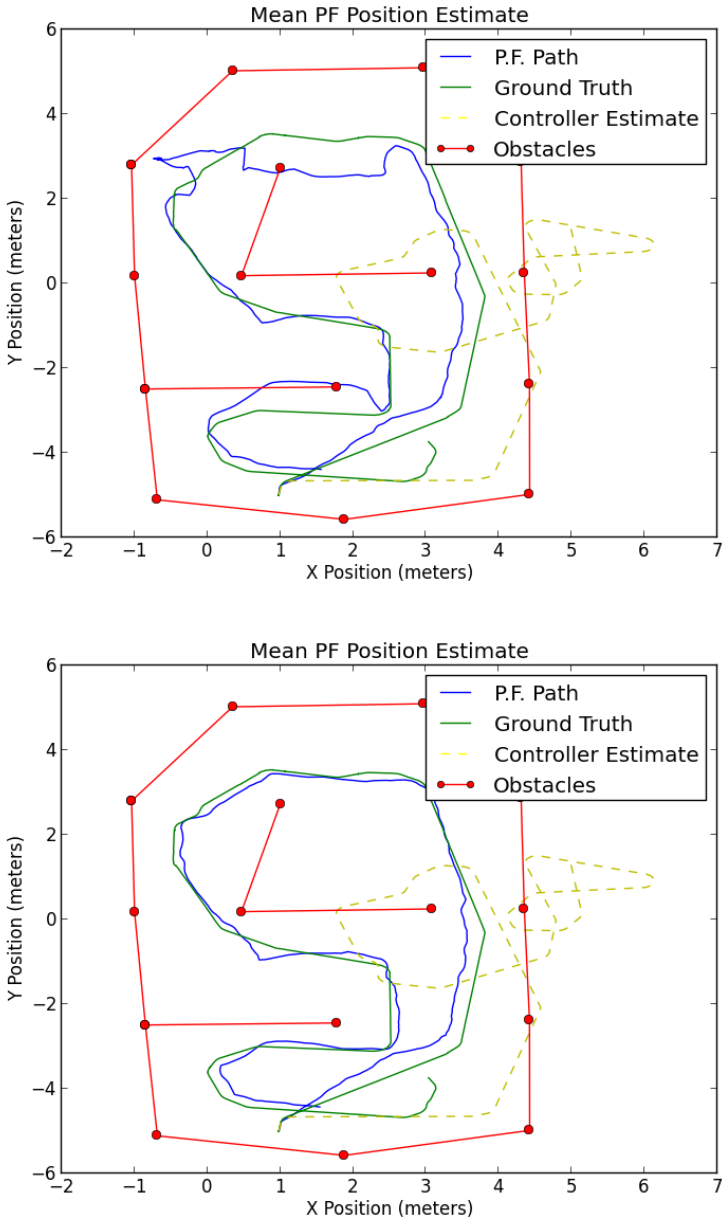

Figure 3: The top graph shows the p.f. with no angle bounding in the measurement step, clearly the measurements are pushing the p.f. off track but it able to re-converge. In the bottom figure the deviations caused by the angles not being bound are eliminated. Note only sampled for the first 2000 command steps for clarity.

A particle filter state class is implemented in *ParticleFilter* in *simulations.py*. The class maintains the state of particles, and cheap step of the algorithm must be called independently. This was done in a piece wise fashion for ease of analyzing each step and greater level of control when experimenting with various parameters. For example the update step, for recalculating a control command and updating the importance factor is a separate function from the resampling step.

Some implementation notes:

- Initial conditions – Unless otherwise noted the initial state of the robot is (x=0.98038490, y=-4.99232180, heading = 1.44849633) for easy comparison to the ground state – See Q10 for random initial conditions.
- Mean state extraction (note see Q10 for more details and variations)
- State mean smoothing – Unless otherwise noted the extracted state is averaged over the past ten values. This may cause a slight lag in the robots performance over the ground truth but it offers a much smoother path, traceable path.
- Particle Number m = 500
- Measurement noise is gaussian, unless otherwise noted, the a covariance R = 0.1 (note see Q10 for more details)
- Resampling randomization – see Q10 for variations.

See code comments for more details.

---

8) [*] Compare the performance of your motion model (i.e. your filter without the measurement update step) and your simulated controller on the sequence of commands from step 2, and also on the robot dataset (as in step 3). Report the results, explain any differences.

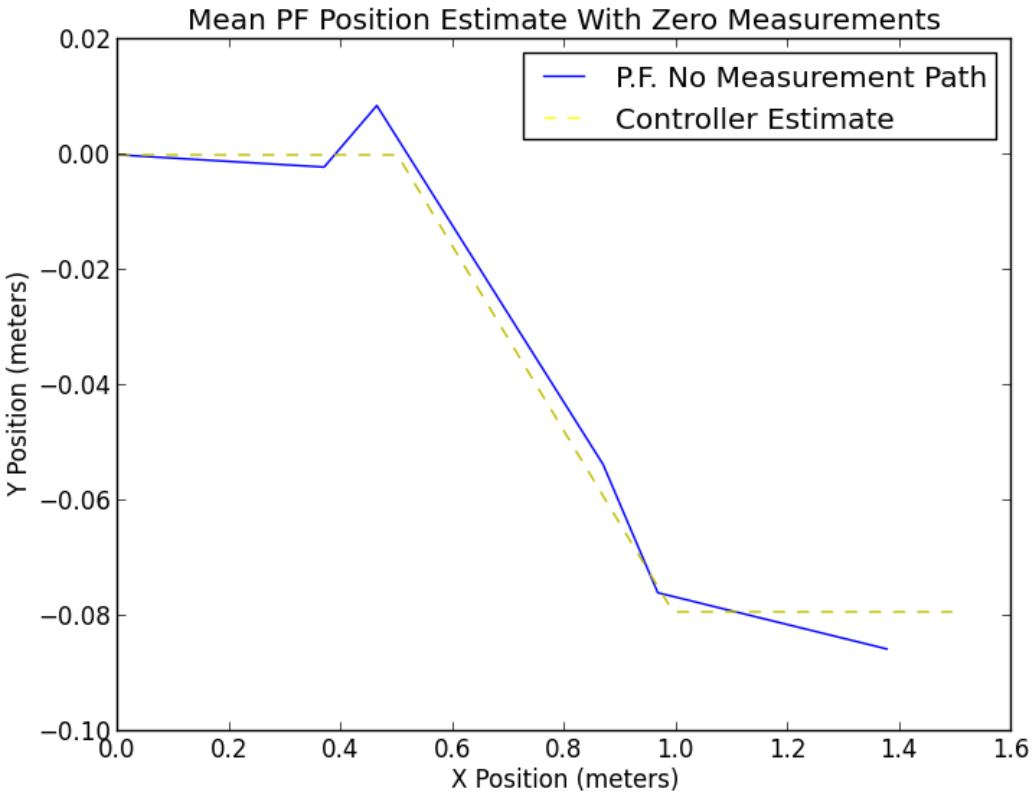

Figure 4: The command vector provided in Question 2, with the p.f. motion model applied to it without a measurement update step.

As described in question five, the motion model is borrowed directly from the simulated controller model, so it is expected that they look very similar.

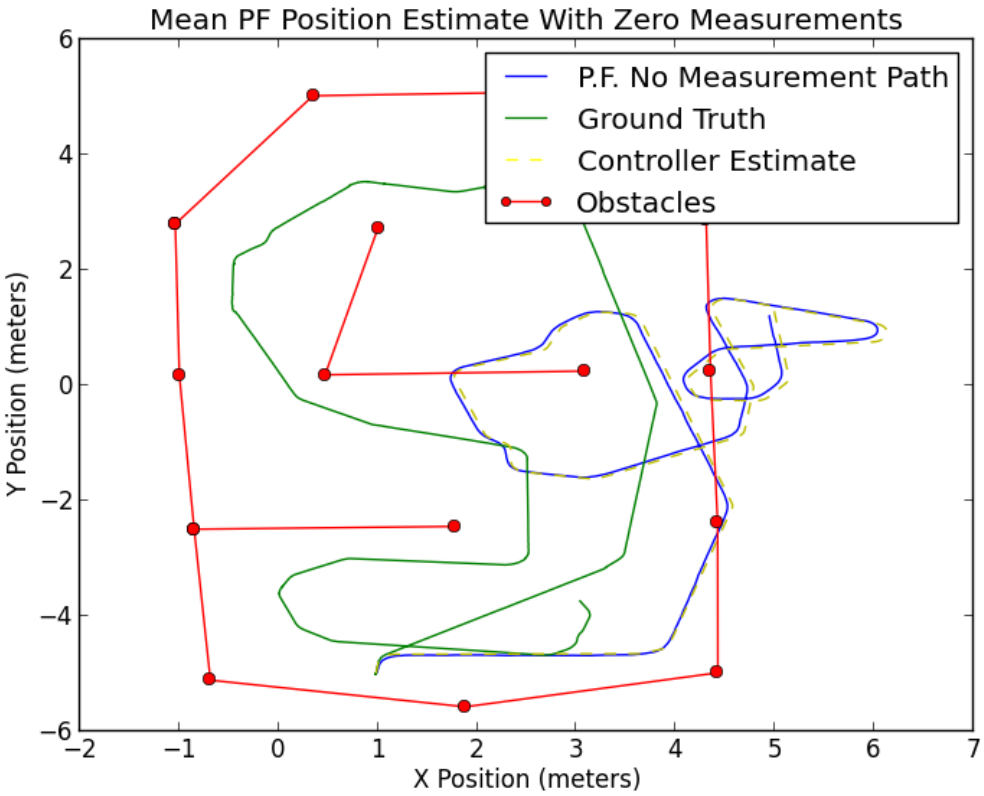

Figure 5: A close up view of view of the first 2000 commands and some additional ground truth information.

In Figure 4 we can see the that the measurement model has a bit of noise on this small scale example most likely due to the gaussian structure of the initial conditions, which are centered around (0,0,0) with standard deviations of (0.001, 0.001, 0.005) respectively.

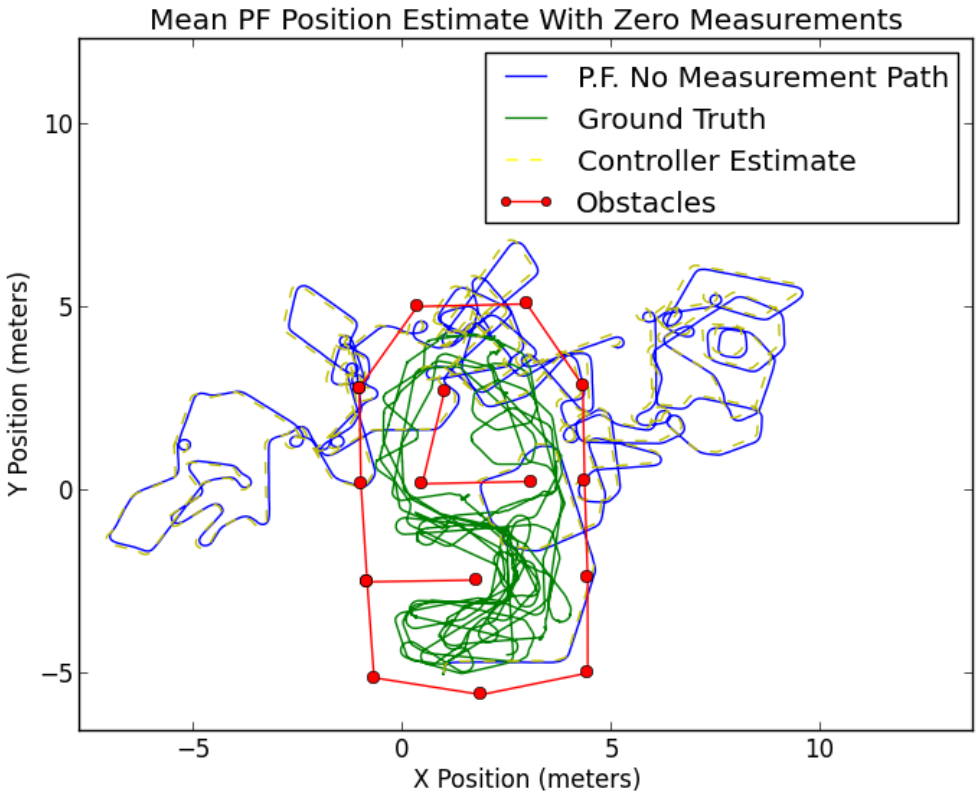

Figure 6: A full view of all the commands executed

By looking at Figure 5 we can see that that the particle filter without measurements almost mirrors the output form the simulated controller. In this close up view though it is possible to see a small deviation from the simulated. This is caused by the variance in resampling as well as the state extraction process, which returns the average of the last ten mean state extractions. The whole result can be seen in Figure 6

---

9) [*] Compare the full filter performance to the robot dataset results reported in step 8. Provide explanations for any differences (what performs well? what performs poorly? under what con-

ditions? and why?) ground your explanations in the algorithm maths.

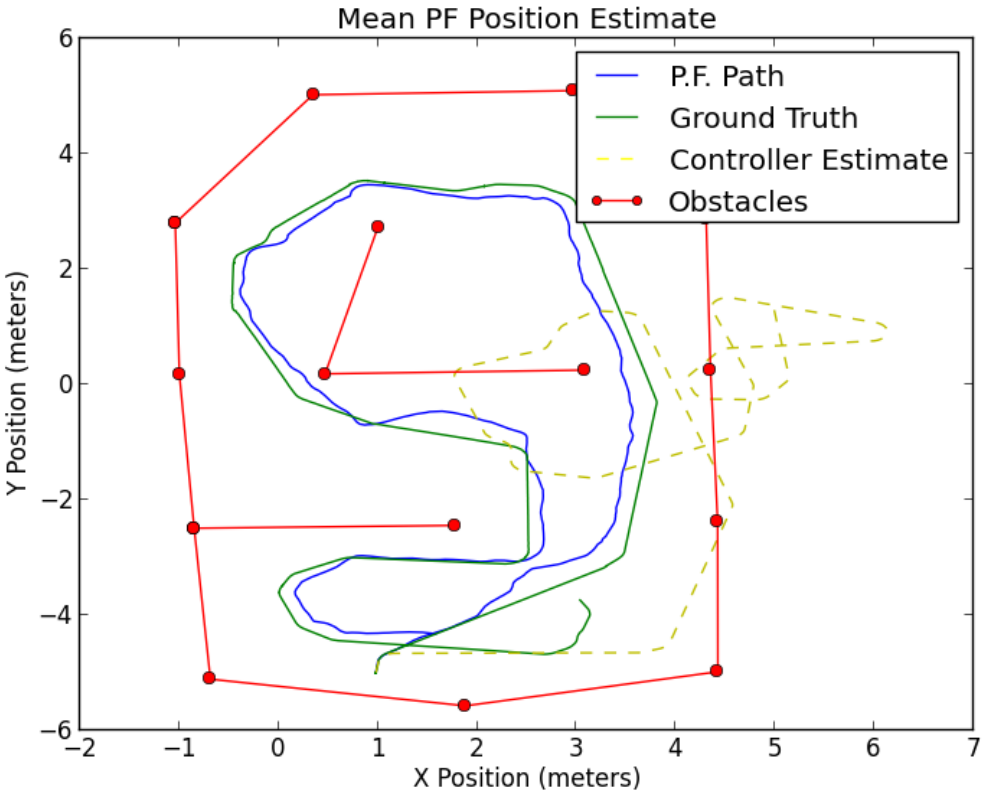

Figure 7: A view of the first 2000 commands ran through the particle filter, including measurements.

Now that the measurement has been incorporated in the particle filter, there is a stark improvement in performance in relation to ground truth over the non-measurement performance as seen in Q8. Looking at Figure 7 we can clearly see a small sample of the first 2000 commands. By visual inspection, the p.f. does a good job of converging on the ground truth when the command inputs

are constant for a period of time.

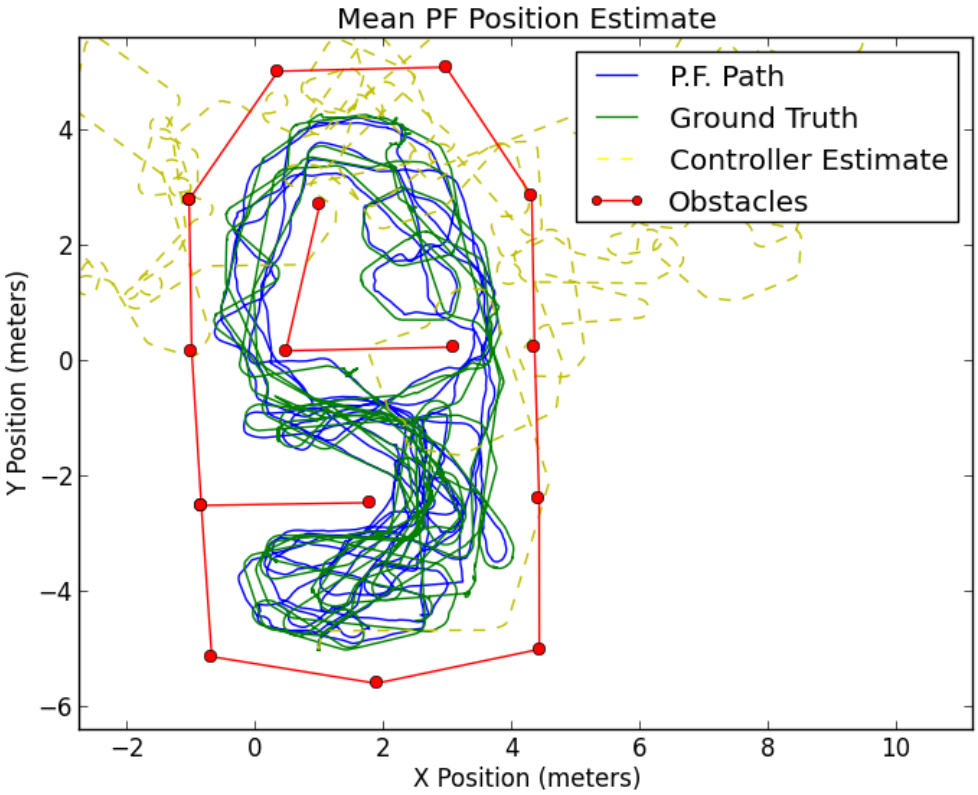

Figure 8: A full view of all the commands ran through the particle filter, including measurements.

Here, looking at Figure 8, the p.f. seems to match the ground truth fairly closely. There is a small persistent lag displacement throughout the p.f. that is most likely caused by the smoothing affect, where the previous 9 p.f. estimations are averaged into

the current estimation, making the state more resilient to large changes.

---

10) [*] Examine the effect (trends? limits?) of uncertainty in the algorithm (i.e. changing the noise parameters). Examine its performance at different parts of the execution (e.g. when missing an expected measurement reading). Report plots and/or tables, and provide explanations.

For Q8 and Q9, the p.f used a smoothed mean state extraction, where I simply took the average of the state vectors (x, y, r) across all the particles to determine the current state. This was effective due to my resampling characteristics of generating a lot of particles in the proximity of my expected location, giving me a biased particle cloud. I also implemented a histogram based extraction method whose performance can be seen in Figure 9.

Figure 9 shows that the histogram extraction method has a similar output as the mean extraction method and maintains the general shape and consistencies as the mean smooth. It will likely prove a more robust method of state extraction, particularly when there is a less biased particle cloud.

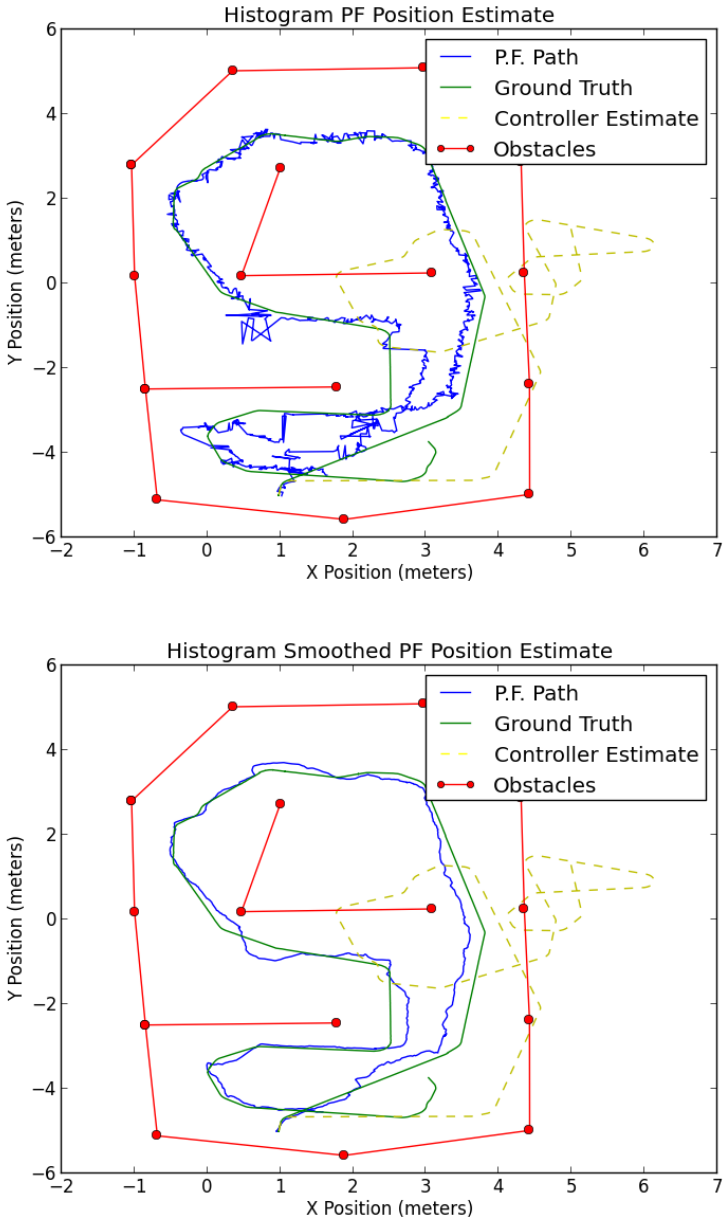

Figure 9: The p.f using the histogram state extraction method, where the top figure uses return the maximum histogram bin state for each p.f step, where the bottom figure is the average of that value over the last 10 steps for a smoother path. Note only sampled for the first 2000 command steps for clarity

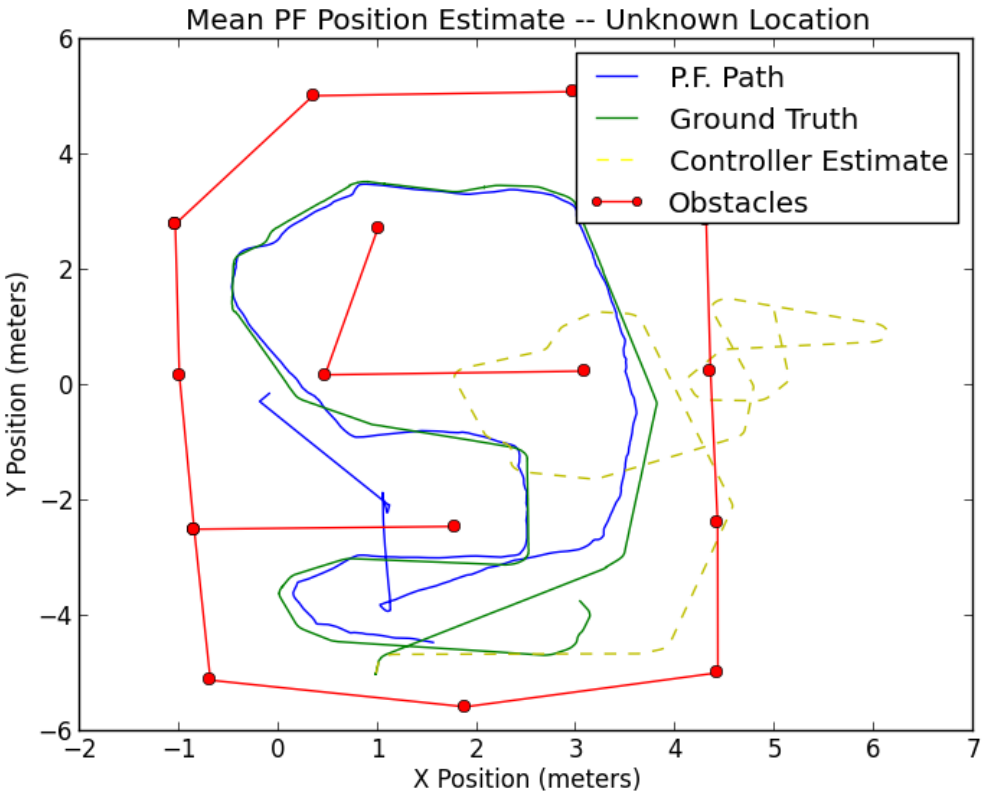

Figure 10: Here the initial conditions for the p.f. are basically unknown, with a particle cloud of random positions and uniform weights, using mean smoothed state extraction. Note only sampled for the first 2000 command steps for clarity

Another assumption that was made for Q8 and Q9 were the starting conditions. For the problems I assumed the the starting conditions were the robot ground truth position and created particles within a gaussian distribution around that location. This allowed for easy comparisons between ground truth, p.f. and controller performances, but it does not show the strength of the particle filters ability to converge on a estimate. Here in Figure 10 the robot initial point cloud is comprised of random locations and headings all with a uniform weight distribution. Here we can see the ro-

bustness of the p.f. even such uncertainty, as it converges on the ground truth within 500 or so command steps.

Looking at Figure 10 we see the robots location starting somewhere near (0,0), as the random locations values were generated from +10 to -10 and averaged towards the origin. From here the p.f. adjusts downwards to where the robot starts, and then as the robot starts to move it begins to adjust as well, in a similar but displaced direction before arriving at a more accurate location.

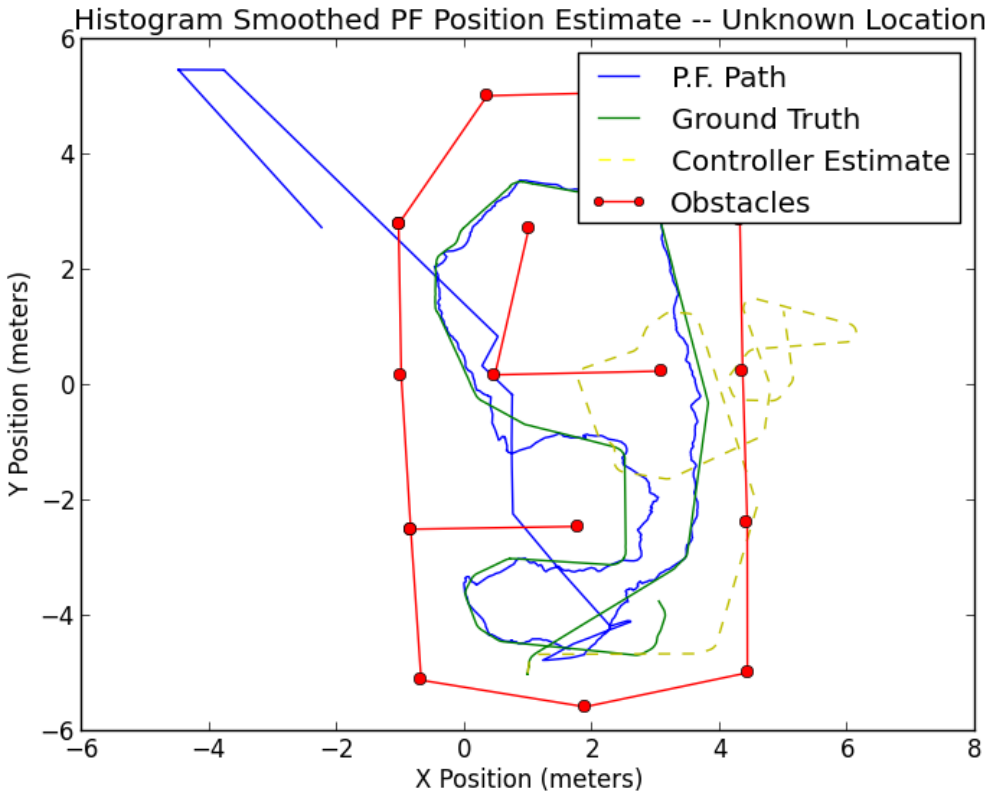

Figure 11: Again the initial conditions for the p.f. are randomly generated, but this time we are using histogram smoothed state extraction. Note only sampled for the first 2000 command steps for clarity

As mentioned earlier with the histogram model of state extraction might be more robot to a less certain particle cloud. In the case of a randomly generated starting cloud, it may prove more efficient in converging. This turns out to be the case in our scenario. Looking at Figure 11 we can see the p.f. taking more much drastic steps to converge on the ground truth location in a more direct and faster fashion since the histogram ignores much of the particle cloud when estimating its location, versus the mean method which must wait of the changes to propagate throughout the particle cloud in order to converge correctly.

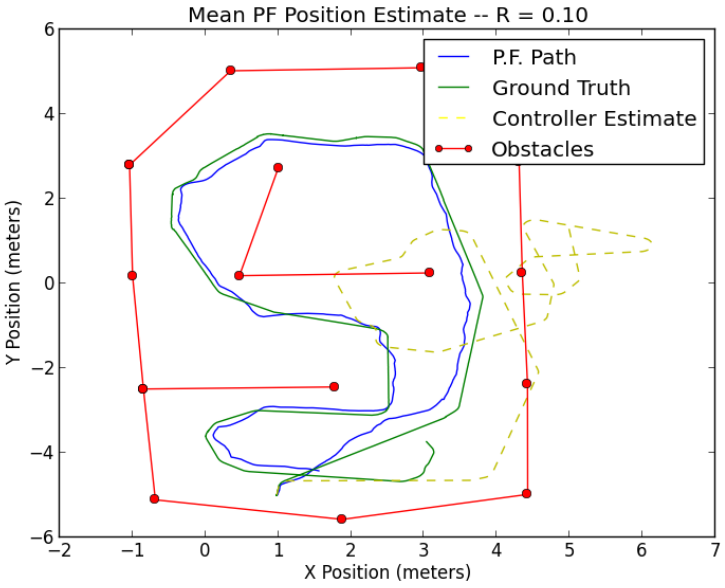

Figure 12: R = 0.1, the default case for this paper and for the following figure comparisons. Note only sampled for the first 2000 command steps for clarity

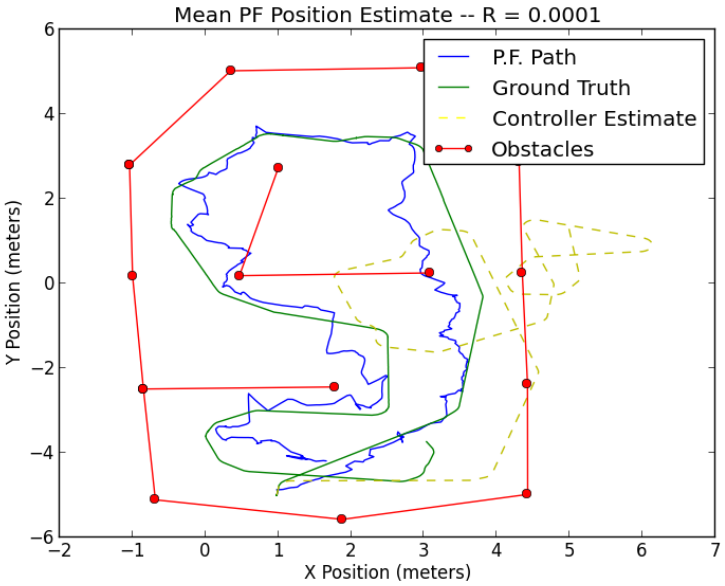

Figure 13: R = 0.0001, the default case for this paper and for the following figure comparisons. Note only sampled for the first 2000 command steps for clarity

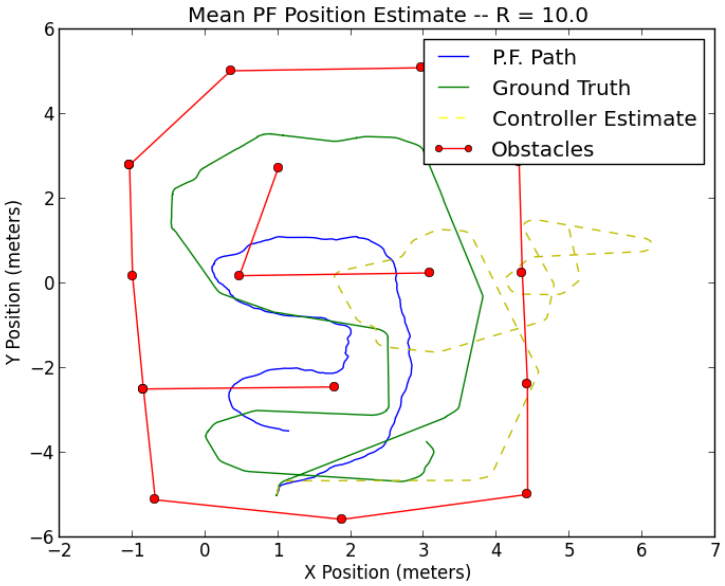

Figure 14: R = 10.0, the default case for this paper and for the following figure comparisons. Note only sampled for the first 2000 command steps for clarity

Another major assumption made earlier in Q7 was the covariance value R, used in measurement model, from Q6. This assumption was based off testing numerous R values. A couple extreme examples in Figure 13 and Figure 14 helped narrow down to a more function value of R seen in Figure 12

Resource:

- Probabilistic Robotics – Equation 5.9 – to help double check motion model

- Daves Student Tutorial on Particle Filter – helped with resampling technique
  http://studentdavestutorials.weebly.com/particle-filter-with-matlab-code.html