Solution 1. (a) For generating the dataset, I put in the value of $a = -1$ in the dynamical system given to us. The initial state x_0 is drawn from a Gaussian sample with mean 1 and variance 2 using the *np.random.normal* function. The next state is then given as,

$$x_{k+1} = ax_k + \epsilon_k \tag{1}$$

The noise ϵ_k is again sampled from Gaussian distribution (mean 0 and variance 1) using the *np.random.normal* function. The observation that is to be stored in our dataset is given by,

$$y_k = \sqrt{x_k^2 + 1} + \nu_k \tag{2}$$

where the noise ν_k is again sampled from Gaussian distribution (mean 0 and variance 1/2) using the *np.random.normal* function. I ran this for 100 runs to collect 100 observations.

(b) We define a new variable, Z combining the state X and unknown system parameter a such that,

$$Z_k = \begin{bmatrix} x_{k+1} \\ a_k \end{bmatrix} = \begin{bmatrix} a_k x_k + \epsilon_k \\ a_k \end{bmatrix} \tag{3}$$

The matrix A is defined as,

$$A = \begin{bmatrix} \frac{\partial Z_k}{\partial x_k} & \frac{\partial Z_k}{\partial a_k} \end{bmatrix} = \begin{bmatrix} a_k & x_k \\ 0 & 1 \end{bmatrix} \tag{4}$$

$$A(\mu_{k|k}) = \begin{bmatrix} \mu_{k|k}(a) & \mu_{k|k}(x) \\ 0 & 1 \end{bmatrix} \tag{5}$$

Let us assume that the mean of this unknown system parameter a is -10 (can be assumed to be anything). The initial mean $\mu_0(x) = 1$ and $\mu_0(a) = -10$. The initial covariance matrix is a diagonal matrix defined by diagonal elements 2 (variance of observing x) and 1 (variance of observing a which is assumed).

The mean and covariance after the dyanamics propagation step is given by,

$$\mu_{k+1|k} = \begin{bmatrix} \mu_{k|k}(a)\mu_{k|k}(x) \\ \mu_{k|k}(a) \end{bmatrix} = \begin{bmatrix} \mu_{k+1|k}(x) \\ \mu_{k+1|k}(a) \end{bmatrix} \tag{6}$$

$$\Sigma_{k+1|k} = A\Sigma_{k|k}A^T + R' \tag{7}$$

Here R' defines the irreducible noise matrix, which is defined as a diagonal matrix with values 1 (noise in estimating x) and 0.1 (noise in estimating a which is assumed).

For incorporating the observation, $g(x_k) = \sqrt{x_k^2 + 1}$. We define the matrix C as,

$$C = \begin{bmatrix} \frac{\partial g}{\partial x_k} & \frac{\partial g}{\partial a_k} \end{bmatrix} = \begin{bmatrix} \frac{x_k}{\sqrt{x_k^2+1}} & 0 \end{bmatrix} \tag{8}$$

$$C(\mu_{k+1|k}) = \begin{bmatrix} \frac{\mu_{k+1|k}(x)}{\sqrt{\mu_{k+1|k}^2(x)+1}} & 0 \end{bmatrix} \tag{9}$$

The kalman gain is defined as,

$$K = \Sigma_{k+1|k} C^T (C \Sigma_{k+1|k} C^T + Q)^{-1} \tag{10}$$

We next incorporate the observation as,

$$\mu_{k+1|k+1} = \mu_{k+1|k} + K(y_{k+1} - g(\mu_{k+1|k}) \tag{11}$$

$$\mu_{k+1|k+1} = \mu_{k+1|k} + K\left(y_{k+1} - \sqrt{\mu^2_{k+1|k}(x) + 1}\right) = \begin{bmatrix} \mu_{k+1|k+1}(x) \\ \mu_{k+1|k+1}(a) \end{bmatrix} \tag{12}$$

$$\Sigma_{k+1|k+1} = (I - KC)\Sigma_{k+1|k} \tag{13}$$

The last entry of $\mu_{k+1|k+1}$ gives us $\mu_{k+1} = E[a_{k+1}|y_1, ..., y_{k+1}]$ as per handout. The last entry (at 1,1) of covariance matrix $\Sigma_{k+1|k+1}$ gives $\sigma^2_{k+1} = var(a_{k+1}|y_1, ..., y_{k+1})$.

(c) Result: Yes my estimated values $\mu_k \pm \sigma_k$ does match the ground-truth value $a = -1$. Yes, the error does reduce as we add more and more observations and after about 20 observation, it converges to it true value. Note that the red shaded region which shows the standard deviations reduces to the irreducible noise which we defined to be 0.1 in our case after convergence. The plot is shown in Figure 1.

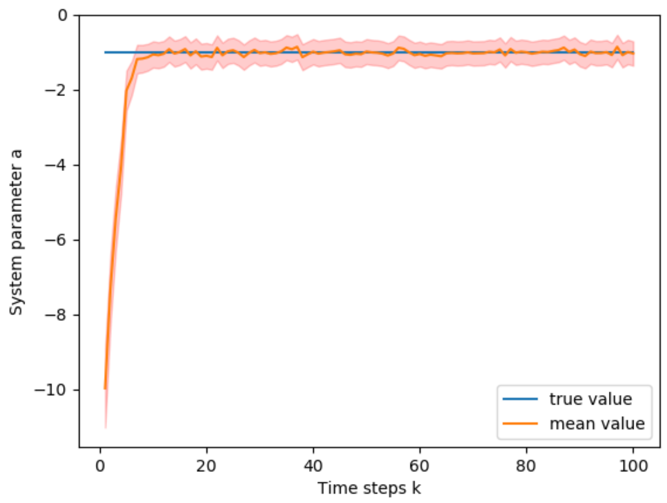


FIGURE 1. Blue line depicts the true value $a = -1$. The orange line depicts the mean μ_k and the red shaded region shows the positive and negative standard deviation $\mu_k \pm \sigma_k$.

Solution 2. (b) Calibrating the sensors:

We assume that only force acting is the gravitational force in the z direction. a_x and a_y should give a reading of zero as the quadrotor stays still in the beginning which means, $a_x = 0$ and $a_y = 0$ for the first 100 time steps. Taking an average of these 100 raw readings of a_x and a_y values gives 511 and 501. Instead of zero values, the readings are biased. Therefore we have biases, $\beta_{ax} = 511$ and $\beta_{ay} = 510$. The figure below (Figure 2) shows a plot of raw values of accelerometer.

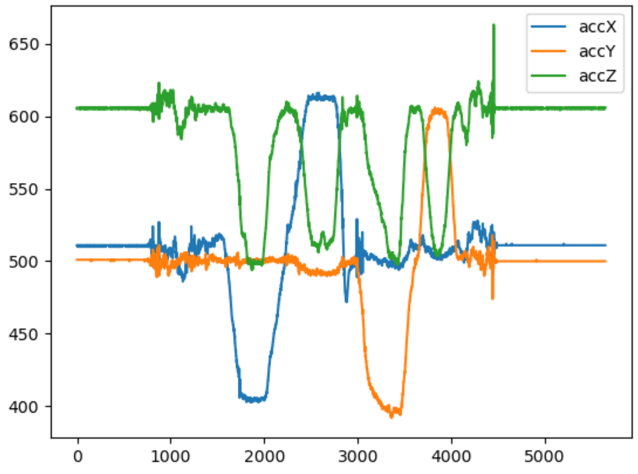


FIGURE 2. Raw values of accelerometer

The roll, pitch and yaw values from Vicon data (obtained by converting the rotation matrices from Vicon data to Euler angles) are plotted (Figure 3). An additional red line is also plotted at value $\pi/2$. In order to obtain the roll angle from the accelerometer, we use the formula,

$$tan\phi = \frac{a_y}{a_z} \tag{14}$$

When $a_z = 0$, the roll value ϕ equals to $pi/2$. Figure 3 shows that the roll value equals $\phi = \pi/2$ within time steps 3406 and 3458 (computed through code and verified through graph). Taking an average of the raw reading of a_z obtained from the accelerometer within these time steps gives a value of 503. Therefore the bias $\beta_{az} = 503$.

Next to compute the senstivity of the accelerometer, we use the equation,

$$value = (raw - \beta)\frac{3300}{1023\alpha} \tag{15}$$

As we discussed, the quadrotor is still in the begining and the only force acting is the gravitation force. Therefore, in the intial time steps, the value of $a_z = 9.81$. Using the raw values from the

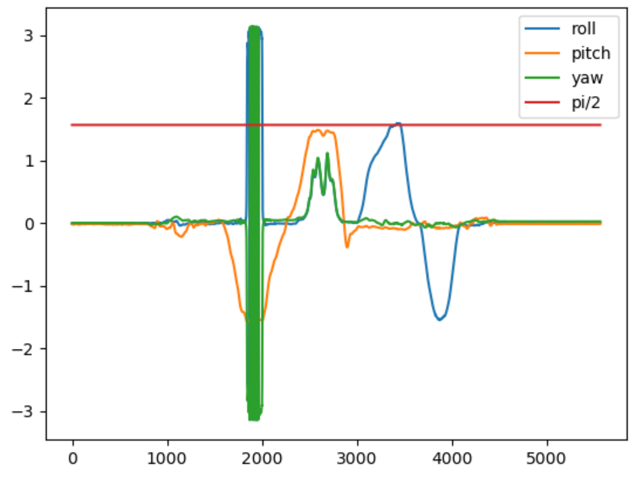


FIGURE 3. Roll Pitch Yaw values from Vicon data

accelerometer of z and bias of z (=503), the sensitivity is computed as,

$$\alpha = \frac{raw - \beta}{value} * \frac{3300}{1023} * 9.81 \tag{16}$$

$$\alpha = \frac{raw_{accz} - 503}{9.81} * \frac{3300}{1023} * 9.81 \tag{17}$$

This gives us the sensitivity value $\alpha = 330mV/g$.

The code for computing the bias and sensitivity of the accelerometer is given below.

```
bias_x = np.round(np.average(accel[0,0:100]))
bias_y = np.round(np.average(accel[1,0:100]))

quat = Quaternion()
for i in range(3000,3500):
        rot = vicon['rots'][:,:,i].reshape(3,3)
        q = quat.from_rotm(rot)
        euler_angles = quat.euler_angles()
        if(euler_angles[0] >= np.pi/2):
                z.append(accel[2,i])

bias_z = np.round(np.average(z))

raw_z = accel[2,0:100]
```

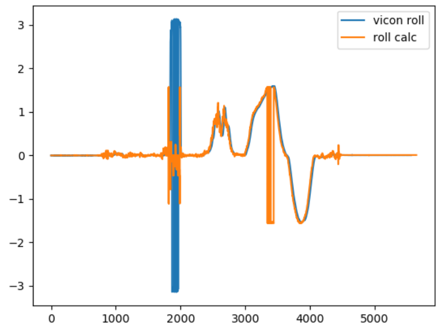
```
alpha = np.round(np.average((raw_z - bias_z) * 3300/1023))
```

In order to check the values, I computed the roll and pitch angles from the accelerometer readings (a_x and a_y readings are flipped in sign and equation 15 is used to compute actual values from raw readings) using the formula,

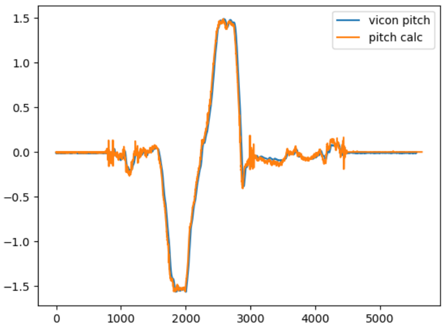
$$tan\phi = \frac{a_y}{a_z} \tag{18}$$

$$tan\theta = \frac{-a_x}{\sqrt{a_y^2 + a_z^2}} \tag{19}$$

The plots comparing the calculated roll and pitch values with the ground truth vicon data is shown below. These validate my bias and sensitivity values.



(A) Roll calculated and Roll vicon (ground truth)



(B) Pitch calculated and Pitch vicon (ground truth)

Computing the bias for gyroscope is pretty straightforward. In the beginning, the quadrotor is still, hence there is no angular velocity. So, the raw initial readings of the gyroscope are nothing but biases. $\beta_{gx} = 374$, $\beta_{gy} = 375$, and $\beta_{gz} = 370$. The figure below (Figure 5) shows a plot of raw values of gyroscope.

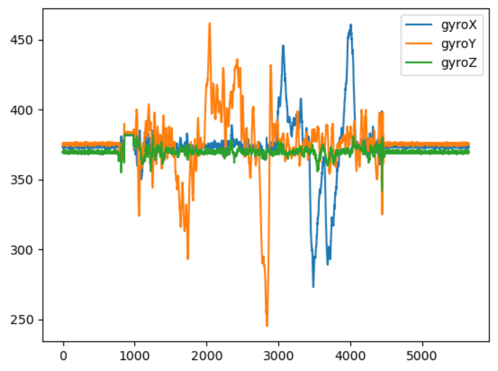
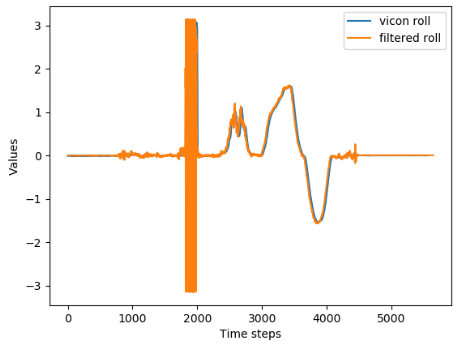
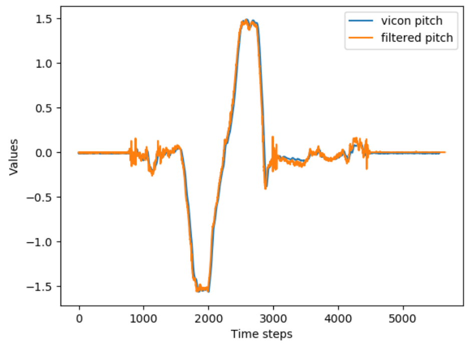


FIGURE 5. Raw values of Gyroscope

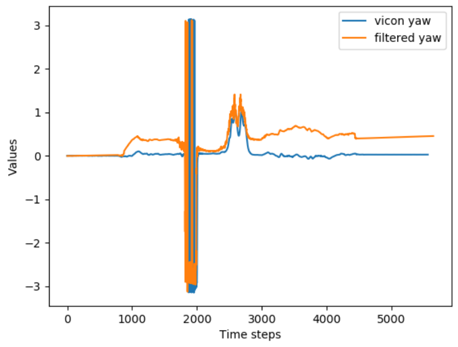
(e) Analysis and debugging:
Results on dataset 1 - The resulting UKF filtered roll, pitch and yaw are plotted with the ground truth Vicon data in the figure below (Figure 6). It shows that the filter is working very well and can precisely estimate roll, pitch and yaw values from accelerometer and gyroscope readings.



(A) Roll filtered and Roll vicon (ground truth)



(B) Pitch filtered and Pitch vicon (ground truth)

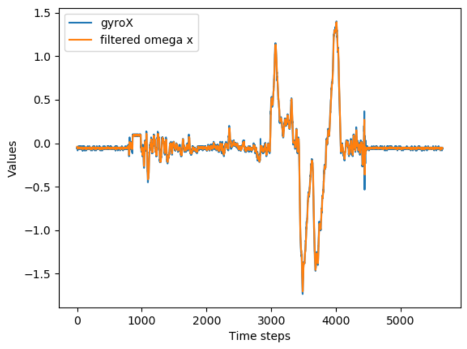


(C) Yaw filtered and Yaw vicon (ground truth)

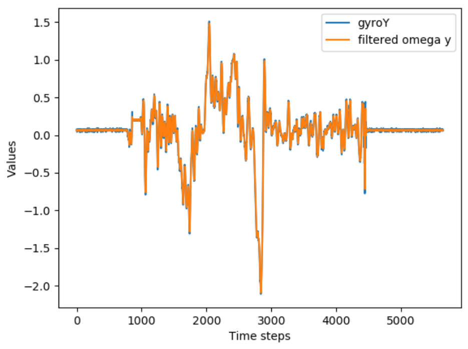
FIGURE 6. Quaternion filtered with Quaternion from Vicon data

The figure below (Figure 7) shows the observations gyroscope readings (in rad/sec) plotted with the resulting UKF filtered estimates of angular velocity.

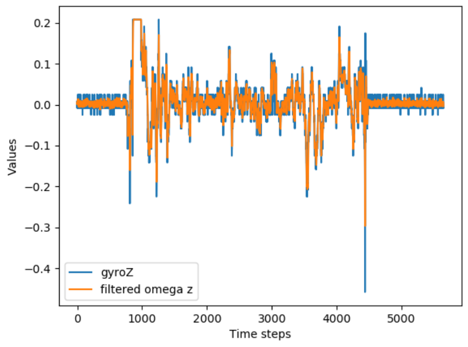
The next figures (Figure 8 and Figure 9) show the mean and standard deviation (shaded red region) of filtered quaternions and filtered angular velocities.



(A) Angular velocity filtered X and Gyroscope reading X

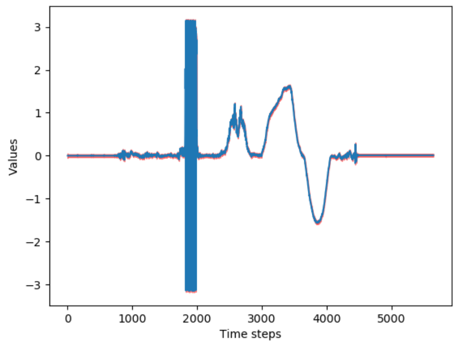


(B) Angular velocity filtered Y and Gyroscope reading Y

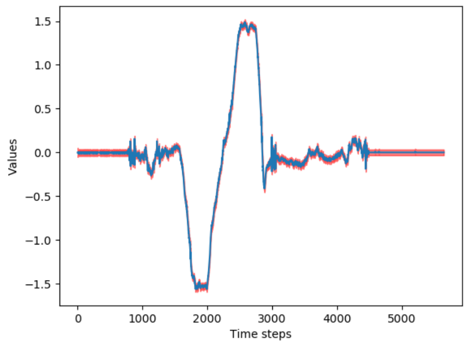


(C) Angular velocity filtered Z and Gyroscope reading Z

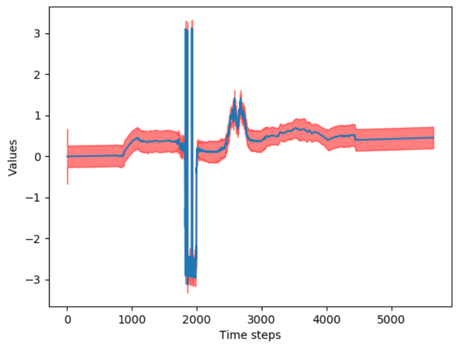
FIGURE 7. Filtered Angular velocities with Gyroscope readings



(A) mean and standard deviation of filtered Roll

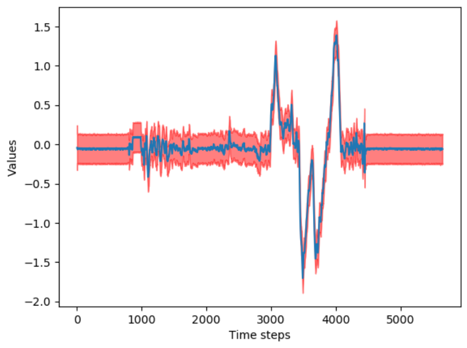


(B) mean and standard deviation of filtered Pitch

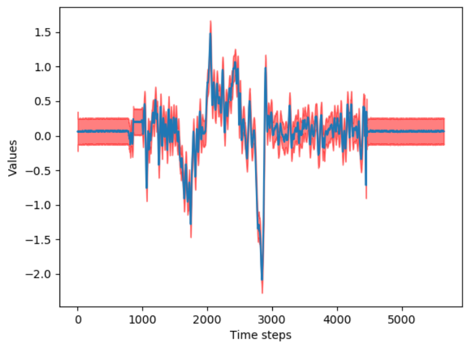


(C) mean and standard deviation of filtered Yaw

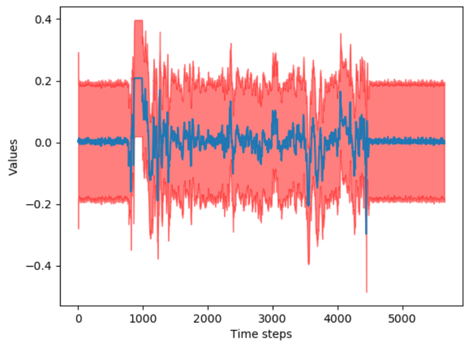
FIGURE 8. Mean and Standard deviation of filtered Quaternions



(A) mean and standard deviation of filtered angular vel X



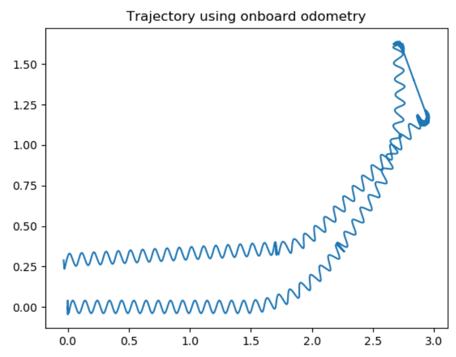
(B) mean and standard deviation of filtered angular vel Y



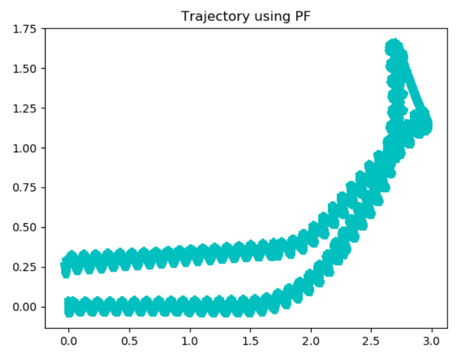
(C) mean and standard deviation of filtered angular vel Z

FIGURE 9. Mean and Standard deviation of filtered Angular velocities

Solution 3. (c) For implementing the dynamics step, the $get\_control$ function and $dynamics\_step$ functions were written. Using the pose (x,y, and yaw θ) at time steps t and $(t-1)$, I calculate the control that the robot took at $(t-1)$ in a particular pose to reach the next pose at step t (i.e. compute Δx, Δy, and $\Delta\theta$). We assume that this is the same control that the robot will take in the dynamics step at time step t to go to $(t-1)$. For each time step, this control is performed on each particle to obtain the updated location of the particles. A multivariate noise (Q) is also added to each particle. The plots of odometry trajectory and particle trajectory for all datasets are shown below.

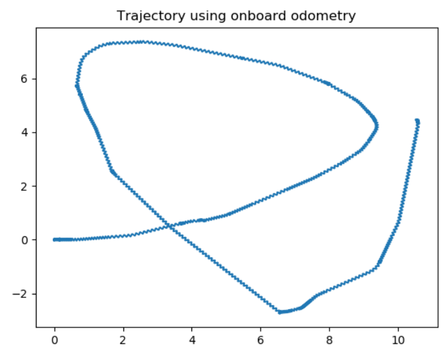


(A) Odometry trajectory

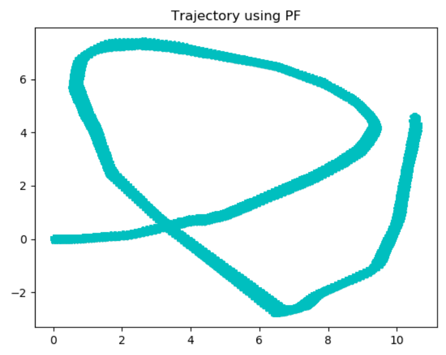


(B) Estimated trajectory

FIGURE 10. Dataset 0



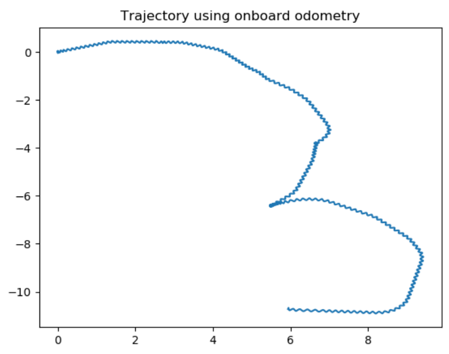
(A) Odometry trajectory



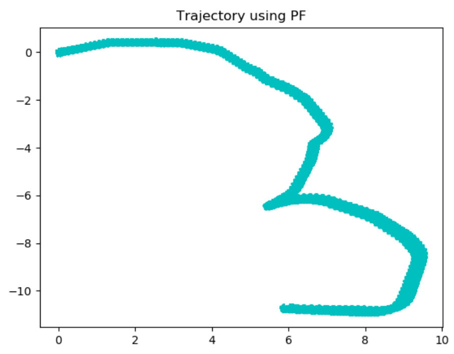
(B) Estimated trajectory

FIGURE 11. Dataset 1

(d) For implementing the observation step, the neck and head angles are computed at each time steps which are then used to project lidar scan data into the world frame. This lidar data is converted to map indices for each particle to calculate which cells are obstacles according to this particle for this scan. I used this information (summing the current estimate of the binarized map) to compute the observation log-probability which is used to update the particle weights. The particle with the largest

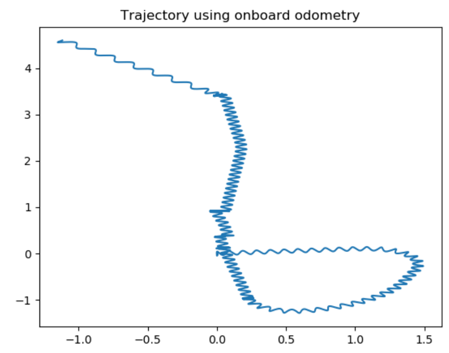


(A) Odometry trajectory

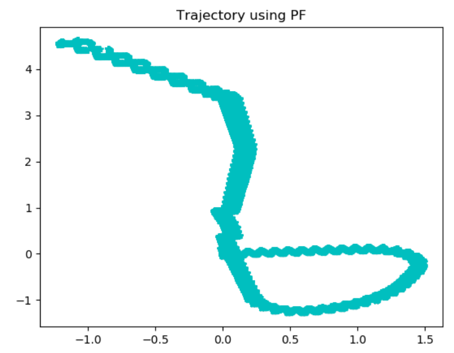


(B) Estimated trajectory

FIGURE 12. Dataset 2



(A) Odometry trajectory



(B) Estimated trajectory

FIGURE 13. Dataset 3

weight is found and the lidar data corresponding to this particle is computed and projected into the world frame. The map indices for this lidar data are the cells that are obstacles. So, I added log odds of occupied cells to them. Next, a linear space is computed from map indices computed from the minimum reading of the lidar data to the obstacle map indices to obtain the cells that are unoccupied. The unoccupied log odds is added to these free cells. The number of observations received for each cell is also updated (which will be used to identify unexplored cells). Lastly, resampling is carried out using the algorithm specified in the notes.

The output of the *run_observation_step* function which prints the particles and their weights is shown below for all the 4 datasets.

Dataset 0 -

```
python3 main.py --mode observation --idx 0
INFO:root:> Reading data
```

```
INFO:root:> Particles
: [[-4.64670052e-10]
 [ 4.05781452e-04]
 [ 0.00000000e+00]]
INFO:root:> Weights: [1.]
INFO:root:

INFO:root:> Particles
: [[2.  0.2 3. ]
 [2.  0.4 5. ]
 [2.7 0.1 4. ]]
INFO:root:> Weights: [1.05306174e-20 1.00000000e+00 1.05306174e-20]
```

Dataset 1 -

```
python3 main.py --mode observation --idx 1
INFO:root:> Reading data
INFO:root:> Particles
: [[0]
 [0]
 [0]]
INFO:root:> Weights: [1.]
INFO:root:

INFO:root:> Particles
: [[2.  0.2 3. ]
 [2.  0.4 5. ]
 [2.7 0.1 4. ]]
INFO:root:> Weights: [0.00667641 0.99086747 0.00245611]
```

Dataset 2 -

```
python3 main.py --mode observation --idx 2
INFO:root:> Reading data
INFO:root:> Particles
: [[0]
 [0]
 [0]]
INFO:root:> Weights: [1.]
INFO:root:
```

```
INFO:root:> Particles
: [[2.  0.2 3. ]
 [2.  0.4 5. ]
 [2.7 0.1 4. ]]
INFO:root:> Weights: [0.1553624 0.4223188 0.4223188]
```

Dataset 3-

```
python3 main.py --mode observation --idx 3
INFO:root:> Reading data
INFO:root:> Particles
: [[0]
 [0]
 [0]]
INFO:root:> Weights: [1.]
INFO:root:

INFO:root:> Particles
: [[2.  0.2 3. ]
 [2.  0.4 5. ]
 [2.7 0.1 4. ]]
INFO:root:> Weights: [2.26031919e-06 9.99995479e-01 2.26031919e-06]
```

(e) The full SLAM algorithm performs one dynamics step and observation step at each iteration. First time t0 is found around which we have both LiDAR data and joint data. Then the occupancy grid is initialized using one particle and calling the $observation\_step$ function. The SLAM algorithm is then started with 100 particles and a map is built from the data. The occupied indices are the one where the binarized logs odds has a value of 1. These cells are marked black. The unexplored indices (where the number of observations is zero) are marked grey. All the other indices are marked white. The pose of the robot at each time step is converted to map indices and is plotted in blue. The location of the particle in the particle filter with the largest weight at each time-step is converted to map indices and plotted in red. Plot of the occupancy map, the odometry trajectory (blue), and the estimated trajectory (red) from the particle filter for all the datasets is shown below.

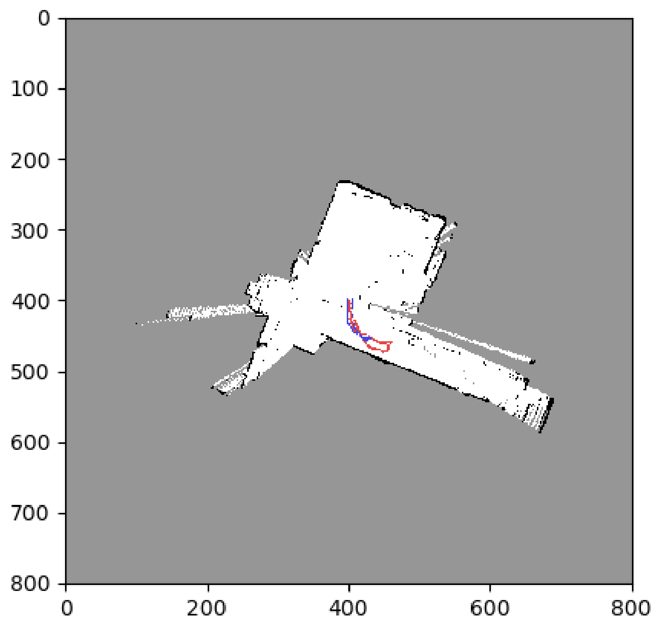


FIGURE 14. Dataset0

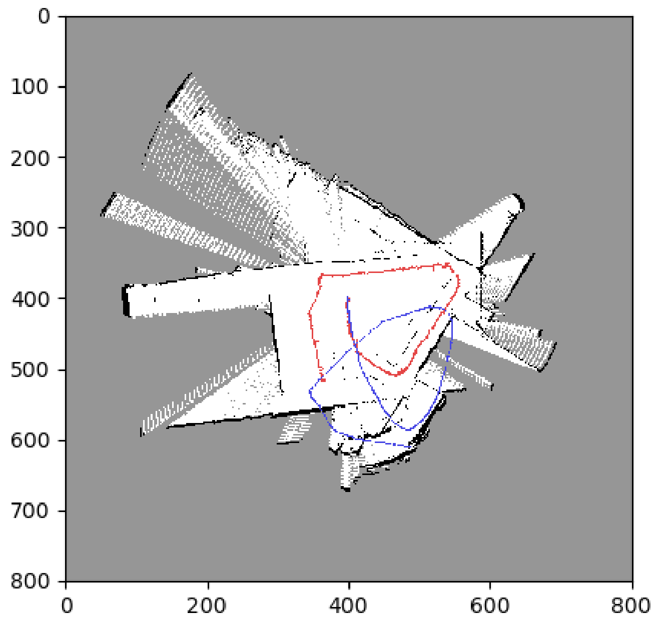


FIGURE 15. Dataset1

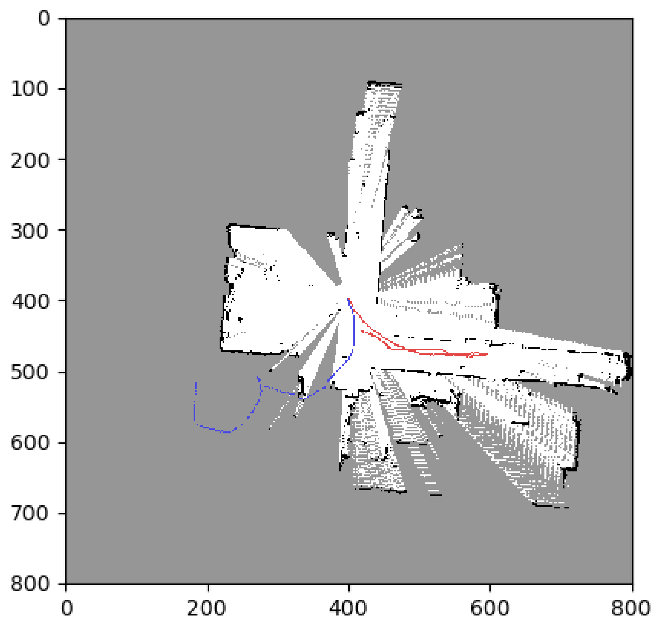


FIGURE 16. Dataset2

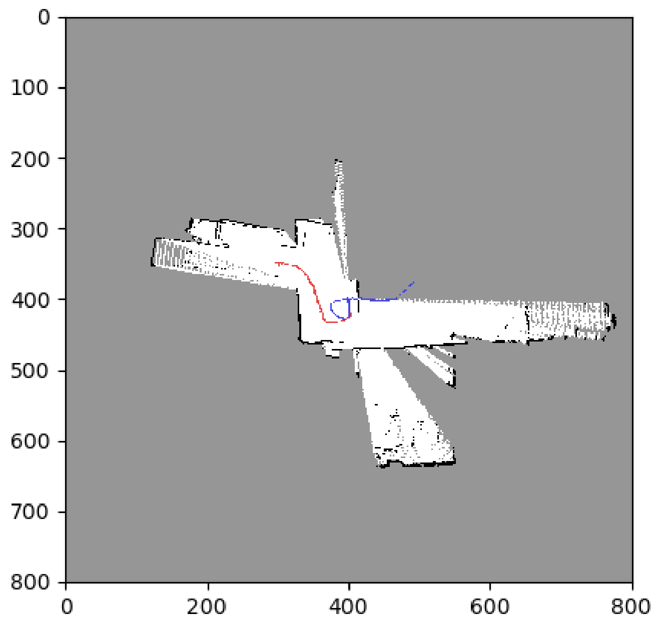


FIGURE 17. Dataset3