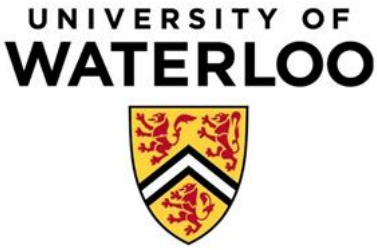


# MTE 544 Assignment

**Prof. M Biglarbegian**

**MTE 544 – Autonomous Mobile Robots**

**Due: March 11th, 2020**

By:

Reza Rajan – 20599340

# Table of Contents

# List of Figures

## Problem 1

This problem requires the derivation of a motion model for a three-wheeled robot. The control input, u, is given as $[v, \omega_\delta]^T$, where is $v$ is the velocity of a point, C, and $\omega_\delta$ is the rate of change of steering angle. Let $\theta$ and $\delta$ denote the heading and steering angles of the robot, respectively.

This is represented as follows:

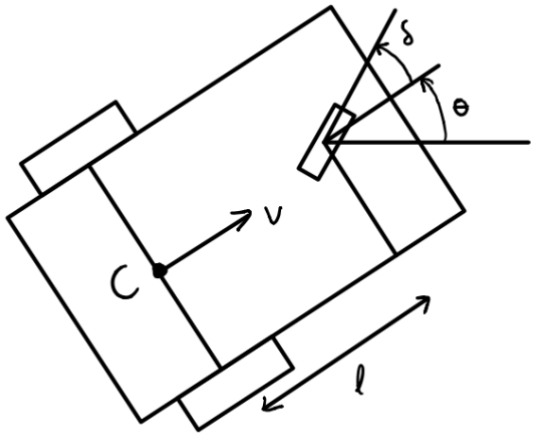


*Figure 1 - Diagram of the Robot in Problem 1*

The following presents the derivation of the motion model:

Assumptions:

1. The back wheels are not a differential drive;
2. The robot is turns only due to the steering, i.e. any frictional effects are negligible.

It is noted that the front wheels control the steering and thus, the heading angles, while the rear wheels control the translational velocity. Furthermore, the motion model considers a timestep approach, where the timestep between each pose calculation is denoted as T.

Back Wheels:

$$\dot{x_r} = \frac{x_{r,t} - x_{r,t-1}}{T} = vcos\theta$$
$$\dot{y_r} = \frac{y_{r,t} - y_{r,t-1}}{T} = vsin\theta$$

Re-writing:

$$x_{r,t} = x_{r,t-1} + vcos\theta_{t-1} \times T$$
$$y_{r,t} = y_{r,t-1} + vsin\theta_{t-1} \times T$$

To understand the dynamics of the steering and heading angles, the front wheels are analysed:

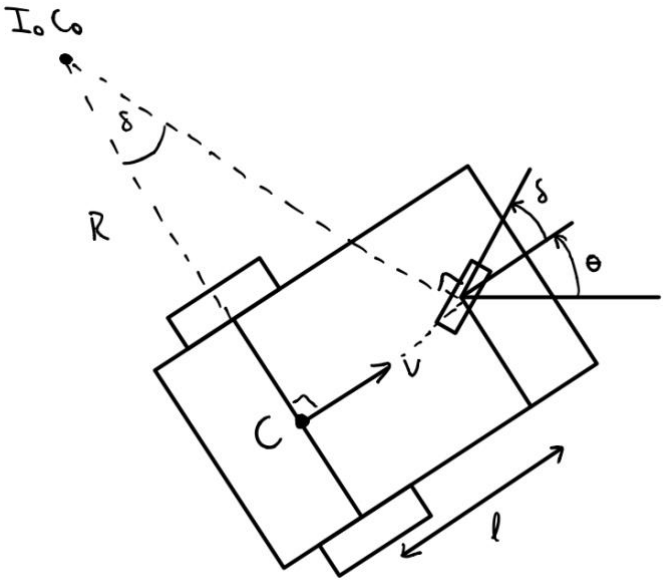


*Figure 2 - Robot Motion Model Diagram for Problem 1*

$$R = \frac{l}{\tan(\delta)}$$

$$\dot{\theta} = \frac{v}{\mathrm{R}}$$

Substituting:

$$\dot{\theta} = \frac{\mathrm{vtan}(\delta)}{l} = \frac{\theta_{r,t} - \theta_{r,t-1}}{T}$$

$$\theta_{r,t} = \theta_{r,t-1} + \frac{\mathrm{vtan}(\delta_{t-1})}{l} \times T$$

Also,

$$\dot{\delta} = \omega_{\delta} = \frac{\delta_{r,t} - \delta_{r,t-1}}{T}$$

$$\delta_{r,t} = \delta_{r,t-1} + \omega_{\delta} \times T$$

Using the following naming convention, the state matrix for the complete motion model is obtained:

$$x_{1,t} = x_{r,t}$$
$$x_{2,t} = y_{r,t}$$
$$x_{3,t} = \delta_{r,t}$$
$$x_{4,t} = \theta_{r,t}$$

## Problem 2

An EKF is designed for the robot in Problem 1 with the following Jacobians:

$$G = \begin{bmatrix} 1 & 0 & 0 & -dt \cdot v \cdot \sin(\theta) \\ 0 & 1 & 0 & dt \cdot v \cdot \cos(\theta) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{dt \cdot v \cdot (\sec \delta)^2}{l} & 1 \end{bmatrix}$$

$$H = \begin{bmatrix} \frac{x^2}{\sqrt{x^2+y^2}} & \frac{y^2}{\sqrt{x^2+y^2}} & 0 & 0 \\ \frac{x}{x^2+y^2} & \frac{y}{x^2+y^2} & 0 & 0 \end{bmatrix}$$

An outline of the EKF algorithm implemented in the code is available in Appendix A – Relevant Code for Part 2. The following are trajectory plots using assigned values for the state-and-process covariance matrices:

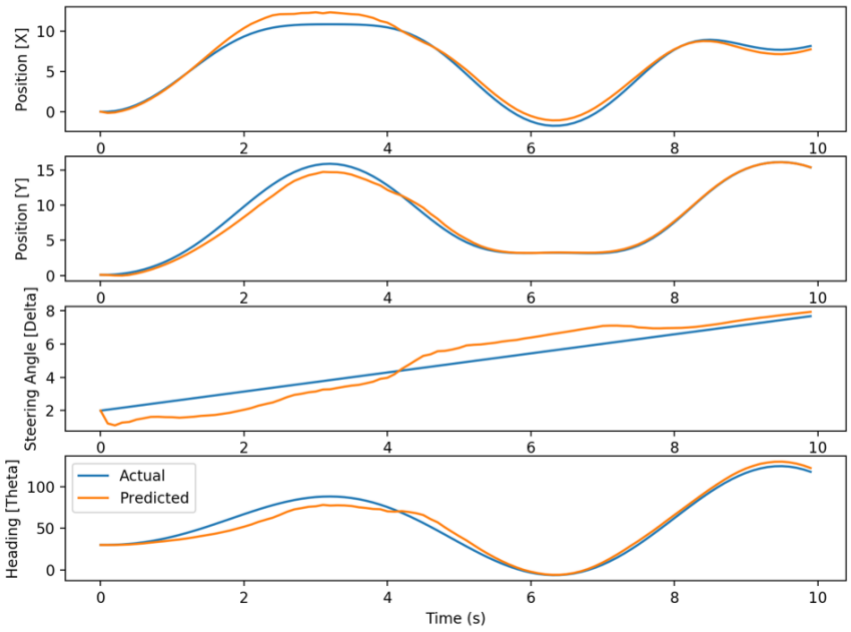


*Figure 3 - Parameter Plot of EKF for Problem 2*

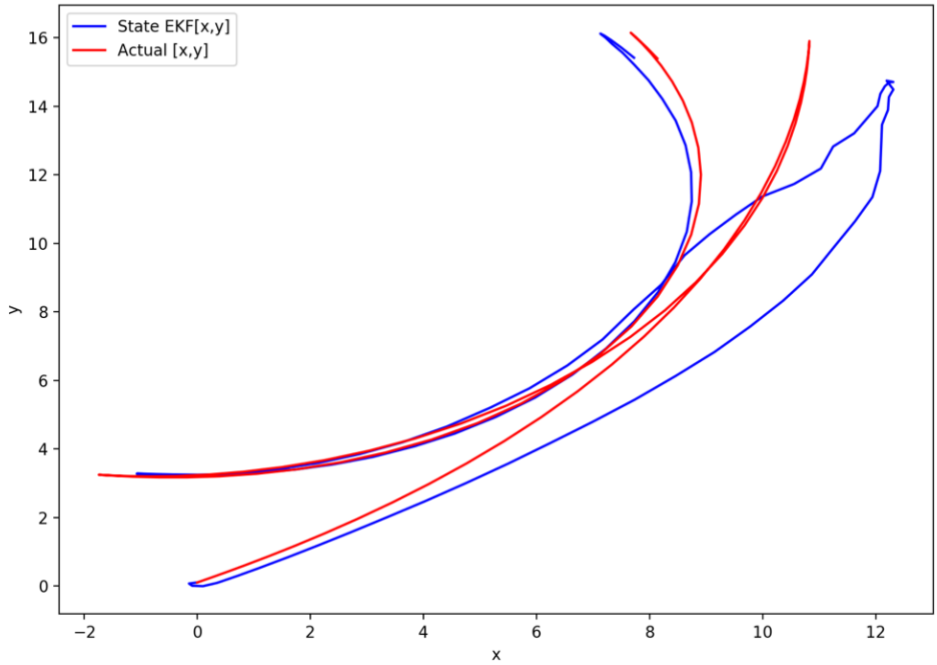


*Figure 4 - Trajectory Plot of EKF for Problem 2*

## Parameter Testing

To observe the behaviour of the EKF the state and process covariance matrices are manipulated. The tests are performed by fixing the process covariance matrix, while adjusting the state covariance matrix. Then, the process covariance matrix is adjusted once the most optimal state covariance matrix is obtained. Note that this assumes these two matrices behave independently, when in reality they may influence each other, and thus such a method of testing would be ineffective.

### Test 1: Process Covariance Fixed

Fixing the process covariance matrix as:

$$Q = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

1. Setting the state covariance matrix to the identity matrix:

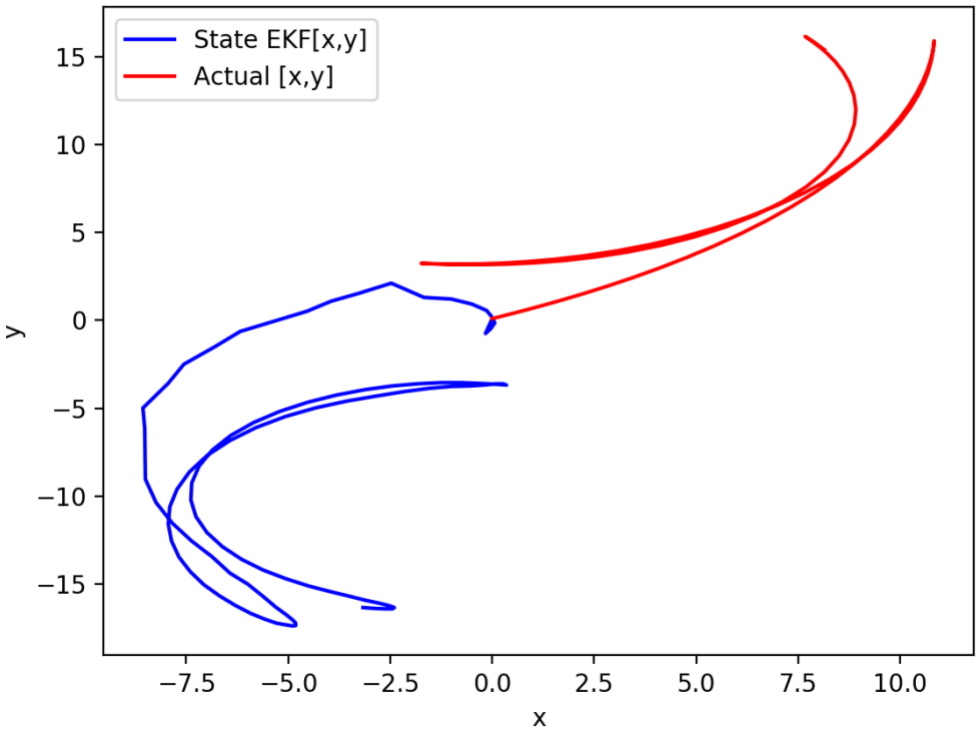


*Figure 5 - Problem 2 Test 1.1*

Clearly, with the identity matrix, the EKF produces a very disconnected prediction of state.

2. Setting the state covariance matrix to the zero matrix:

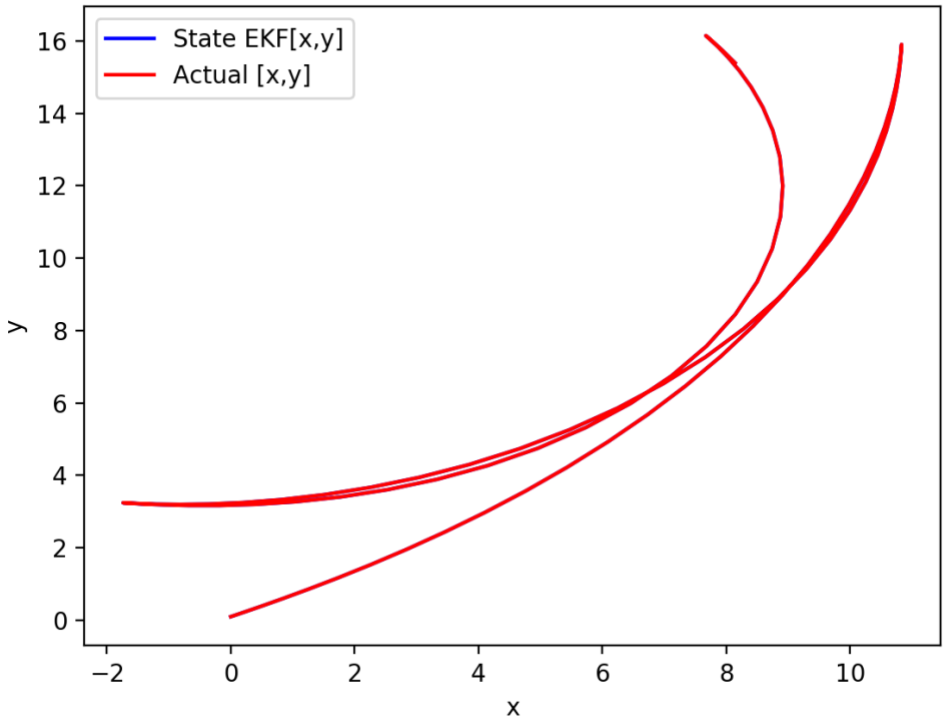


*Figure 6 - Problem 2 Test 1.2*

Clearly, this results in the most accurate EKF state prediction. However, it is unrealistic since in a real system some level of covariance will exist.

3. Setting the state covariance matrix to 0.001:

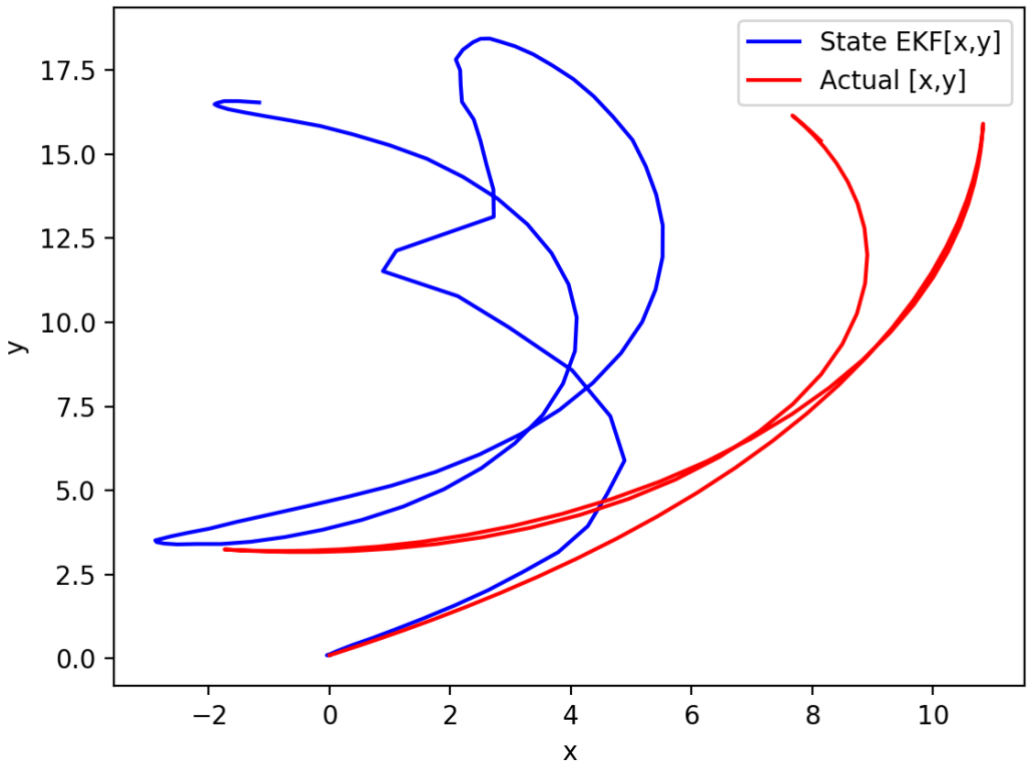


*Figure 7 - Problem 2 Test 1.3*

Though small, this clearly indicates that the state covariance matrix is very sensitive, producing offset predictions.

4. Setting the state covariance matrix to 0.00001:

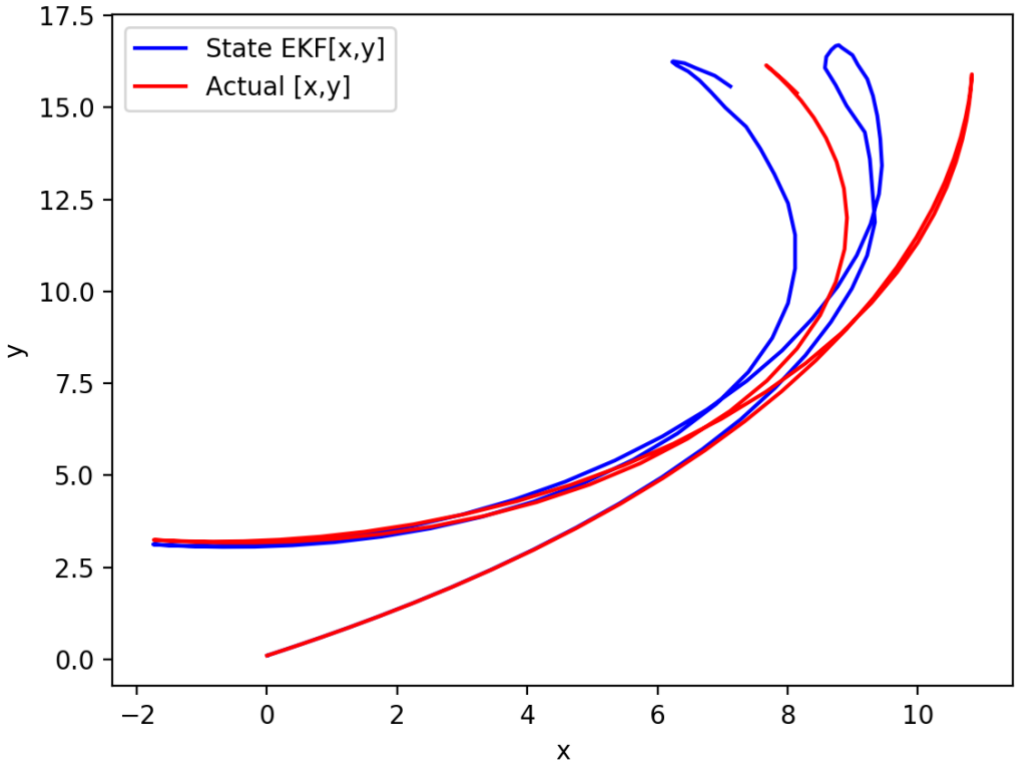


*Figure 8 - Problem 2 Test 1.4*

This produces a more accurate EKF prediction but may still be unrealistic since the covariance value may not represent that of a real system.

### Test 2: State Covariance Fixed

Now, fixing the state covariance matrix, with all values at 0.00001, the process covariance matrix is adjusted:

1. Setting Q to the identity matrix:

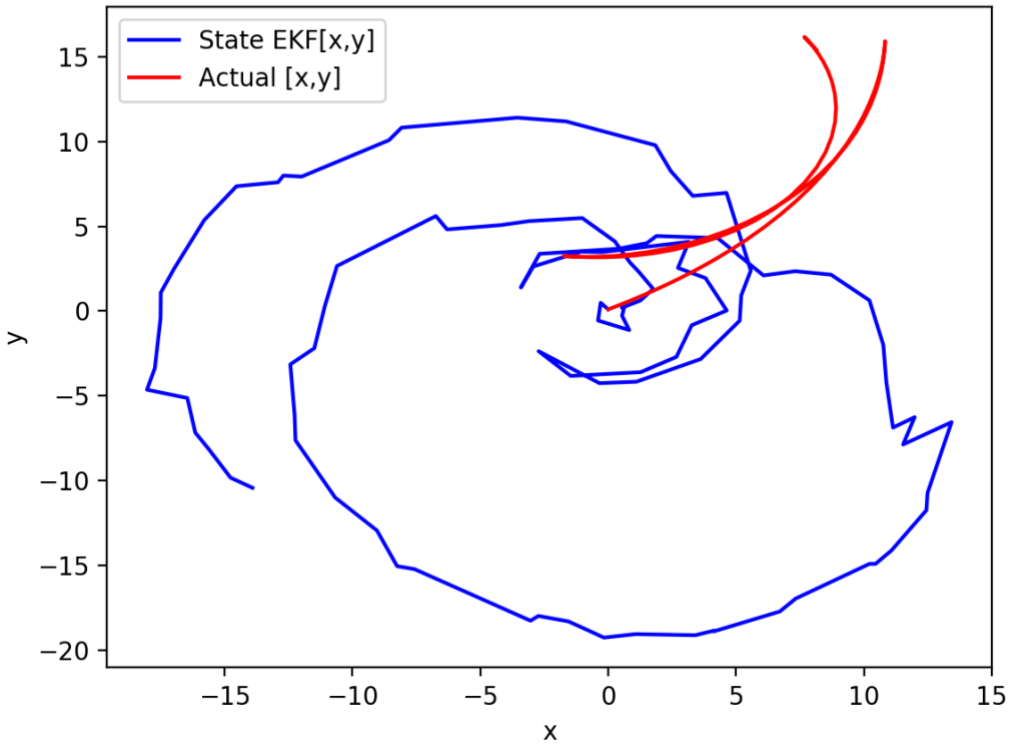


*Figure 9 - Problem 2 Test 2.1*

Clearly, these results indicate a very inaccurate EKF prediction, as expected from an identity covariance matrix.

2. Setting Q to 0.0001:

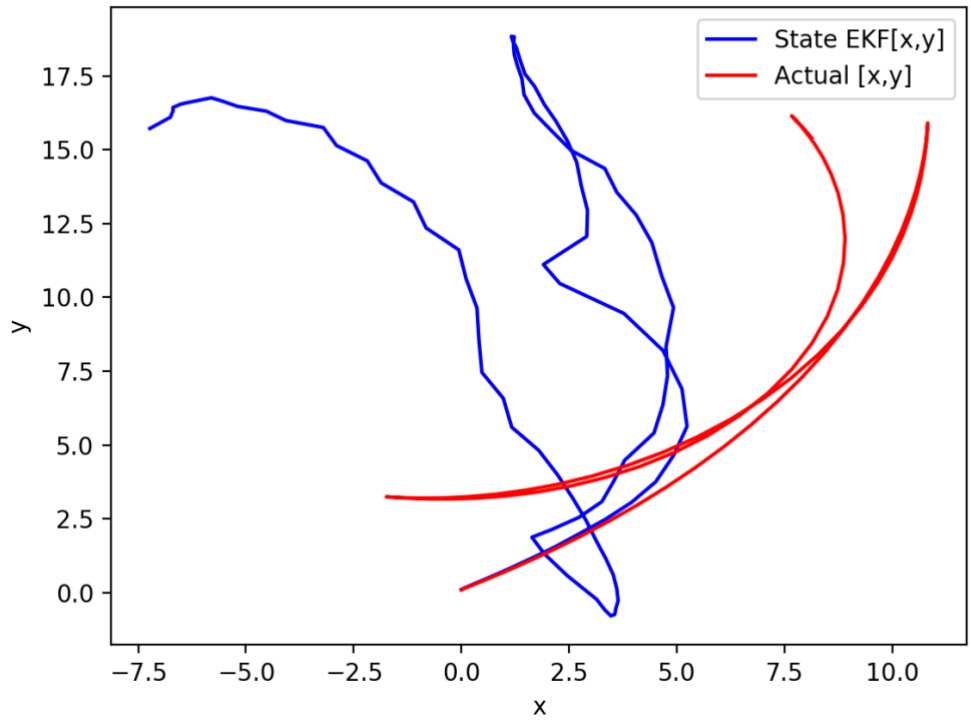


*Figure 10 - Problem 2 Test 2.2*

The results of decreasing the values of the process covariance matrix seem to result in the convergence of the EKF predictions to the actual values.

3. After multiple tests following a similar pattern: adjusting values in favour of the results which produce the most convergence of predictions, the following is tested:

$$Q = \begin{bmatrix} 0.000001 & 0 & 0 & 0 \\ 0 & 0.000001 & 0 & 0 \\ 0 & 0 & 0.00000001 & 0 \\ 0 & 0 & 0 & 0.00000001 \end{bmatrix}$$

This results in the following:

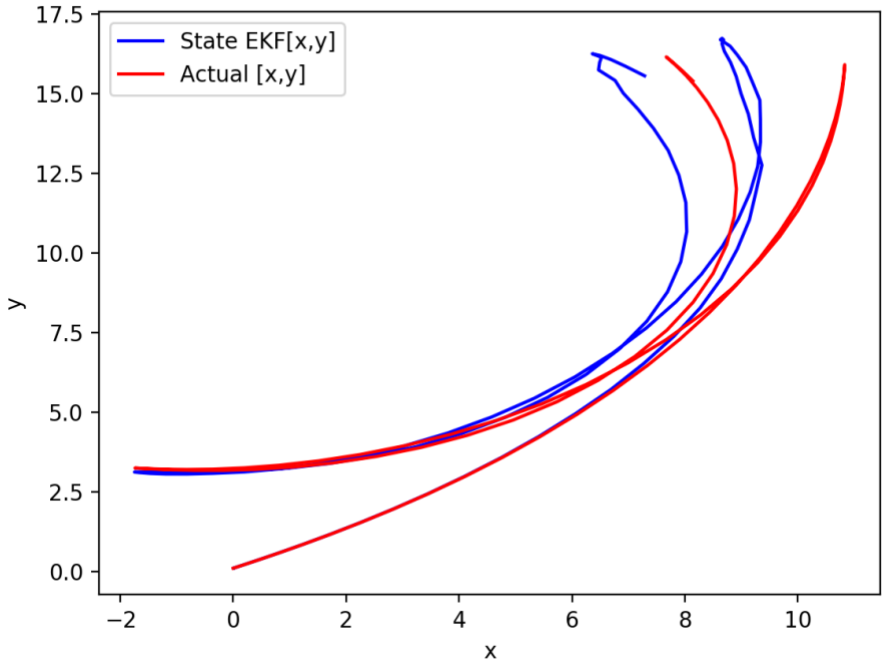


*Figure 11 - Problem 2 Test 2.3*

## Summary

As seen in the above tests, it is noted that as the covariance matrices approach the zero matrix, the more accurate the results. This would make sense since it would mean the variables do not interfere with each other, and there is no apparent noise. It is also apparent that very small adjustments in the covariance parameters can result in either significant improvement, or degradation of the EKF predictions. As such, it is important to tune parameters to meet realistic expectations, while ensuring accuracy of the predictions.

# Problem 3

## Part a

This problem requires the derivation of a motion model for a two-wheeled, differential drive robot considering:

i. How much the centre of the robot will travel and
ii. The incremental change in yaw of the robot

The derivation is as follows:

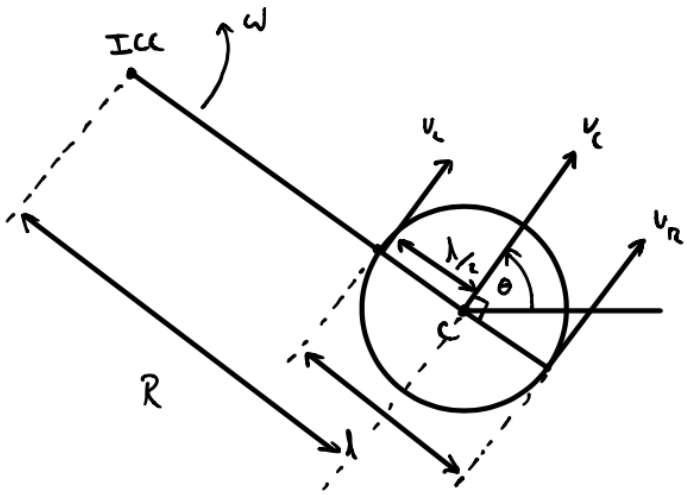


*Figure 12 - Robot Motion Model Diagram for Problem 3*

Considering the notations given in Figure 12, $v_c$ is the velocity at the centre of the robot, C. Likewise, $v_l$ and $v_r$ denote the velocities of the left and right wheels, respectively. The baseline of the robot is defined as $l$, and its heading angle is defined as $\theta$. ICC represents the centre of rotation of the robot, with R being the length from the centre of the robot to ICC.

Noting the following:

i. Each wheel moves with a different angular velocity and
ii. The wheels are rigidly attached to the robot body

The conservation of angular momentum may now be applied:

$$v_r = \left(R + \frac{l}{2}\right) \times \omega \ldots (1)$$

$$v_l = \left(R - \frac{l}{2}\right) \times \omega \ldots (2)$$

Solving simultaneously yields:

$$R = \frac{l}{2}\left(\frac{v_r + v_l}{v_r - v_l}\right) \ldots (3)$$

$$\omega = \frac{v_r - v_l}{l} \ldots (4)$$

Also,

$$v_c = R\omega$$

Substituting in (1) and (2) yields:

$$v_r = \left(v_c + \frac{\omega l}{2}\right) \ldots (5)$$

$$v_l = \left(v_c - \frac{\omega l}{2}\right) \ldots (6)$$

$$\therefore \ \Delta D_{c,t} = \ R\omega \cdot T = \frac{v_r + v_l}{2} \cdot T \ldots (7)$$

Similarly, for yaw:

$$\therefore \ \dot{\theta_{c,t}} = \ \omega = \frac{\theta_t - \theta_{t-1}}{T} \ = \frac{v_r - v_l}{l}$$

$$\therefore \ \Delta\theta_{c,t} = \ \omega \cdot T = \left(\frac{v_r - v_l}{2}\right) \cdot T \ldots (8)$$

## Part b

For this part, two profiles are assumed for the left and right wheels of the robot, and the robot's pose and trajectory is calculated and plotted using the equations in Part a. The plot is generated with a timestep of 0.1 seconds, and a control for the right and left wheels, respectively, is $[\sin(dt * iteration) \quad 2\cos(dt * iteration)]$, where dt is the timestep. The following are the results:

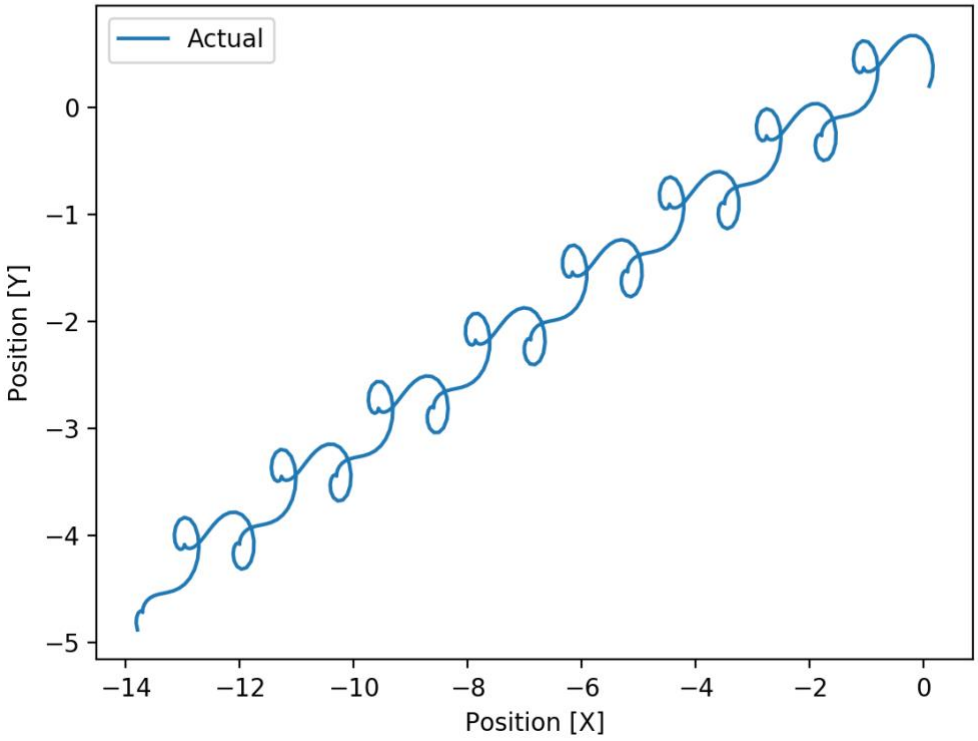


*Figure 13 - Plot of Differential Drive Robot Trajectory*

## Part c

Using the same control as in Part b, the wheel encoders are assumed to have an error.

With a 2% error on both wheels, the following is obtained:

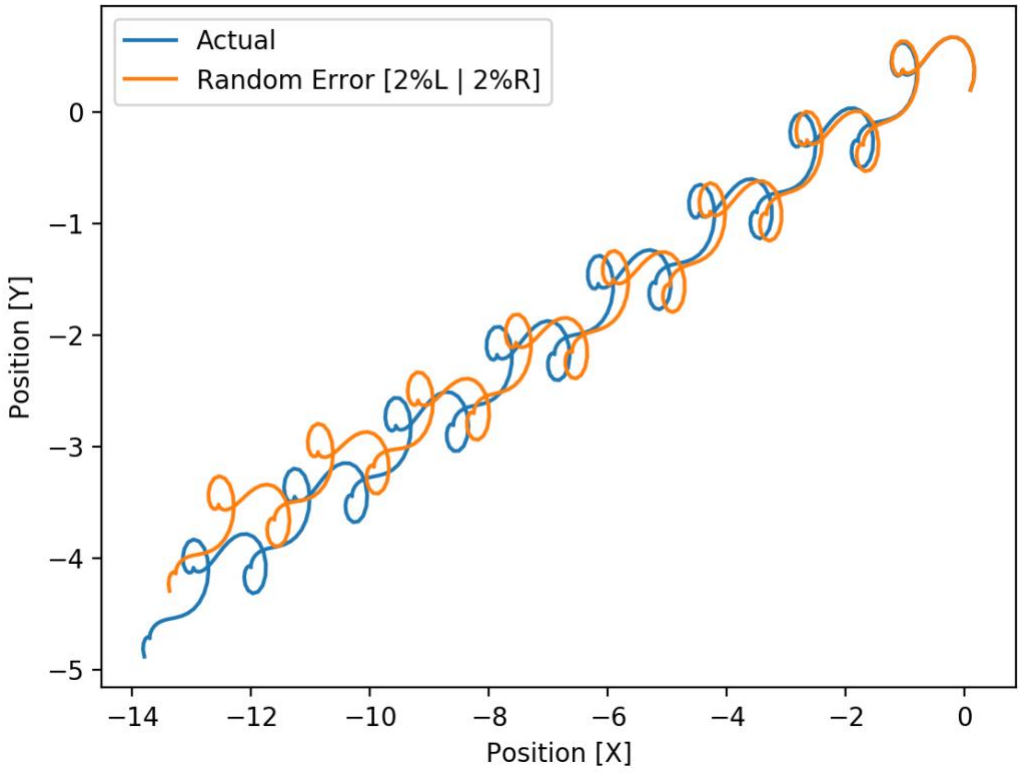


*Figure 14 - Plot of Differential Drive Robot Trajectory with 2% Encoder Error*

This indicates that with an error, the position readings gradually deviate from the actual position (robot starts at the right-top of Figure 14). This makes the robot appear misaligned from its original path. Furthermore, the associated RMSE (root mean squared error) is 0.2274

With a 3% error on the left wheel, and a 2% error on the right, the following is obtained:

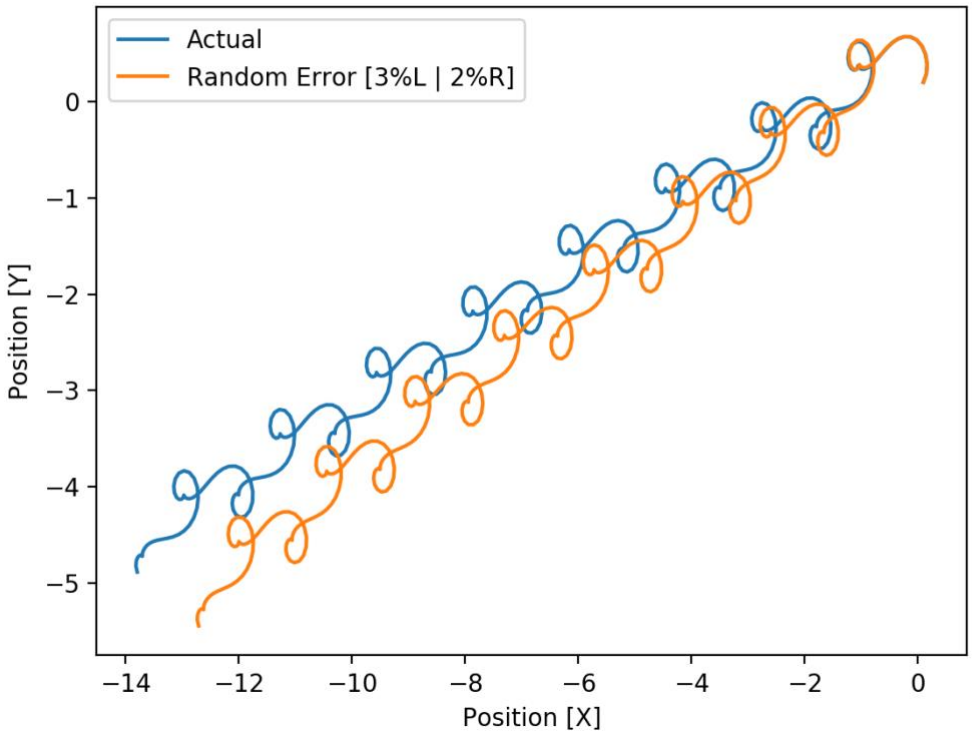


*Figure 15 - Plot of Differential Drive Robot Trajectory with 3% [L] and 2% [R] Encoder Error*

Figure 15 indicates that similar to the results from Figure 14, the position estimates gradually deviate from the actual position. Particularly, when the encoder errors are different for both wheels it has the effect of making the robot appear as though it is shifting, i.e. both misaligned and offset from its original path. In this case, the associated RMSE is 0.3923.

# Problem 4

This part considers an EKF applied to a two-wheel differential drive robot.

## Parameter Selection

The parameter selection process is similar to that outlined in Problem 2: Parameter Testing. It follows an iterative process, fixing one covariance matrix and tuning the other, until a sufficiently optimal solution is found.

## Measurement Models

### Case 1

The measurement model in this case is of the form: $[x \quad y \quad \theta]^T + \delta_t$, where $\delta_t$ is the measurement noise. A measurement covariance matrix of $diag(0.5, 0.5, 1)$ is used. Initial conditions are set with the robot starting at (0.1, 0.2), at an angle of 45 degrees.

Using the motion model in Problem 3Problem 2 and the EKF algorithm from Problem 2, the algorithm is modified with the following Jacobian matrices:

$$G = \begin{bmatrix} 1 & 0 & -delta\_d \cdot \sin\left(\theta + \frac{\Delta\theta}{2}\right) \\ 0 & 1 & delta\_d \cdot \cos\left(\theta + \frac{\Delta\theta}{2}\right) \\ 0 & 0 & 1 \end{bmatrix}$$

$$H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Using a randomized control signal, the following is obtained:

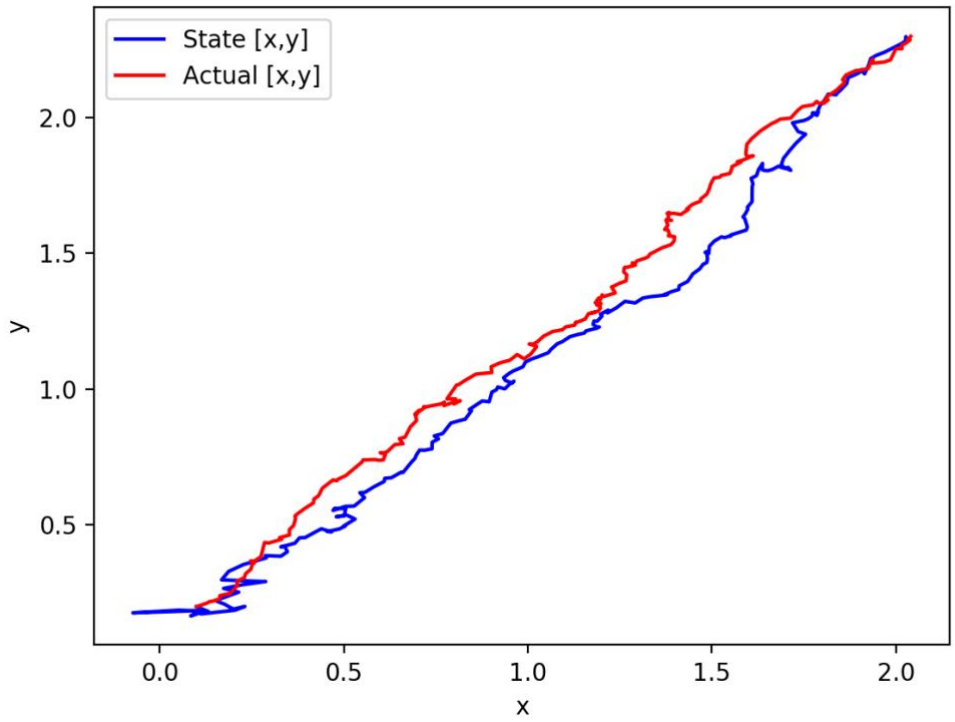


*Figure 16 - EKF for a Random Control Signals for Problem 4, Measurement Case 1*

Figure 16 shows a relatively close prediction by the EKF, with relatively similar trajectories, but at times large offsets.

With a sinusoidal control signal, the following is obtained:

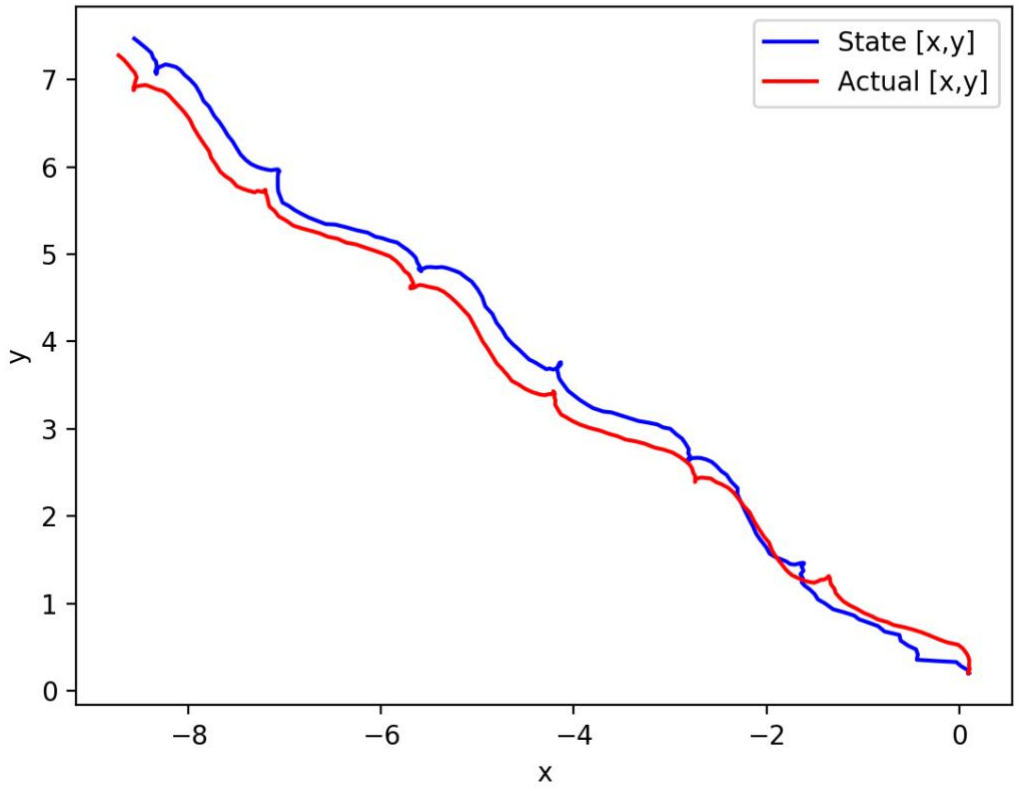


*Figure 17 - EKF for a Sinusoidal Control Signals for Problem 4, Measurement Case 1*

Figure 17 shows a relatively close prediction by the EKF, with relatively similar trajectories, but with an offset.

The results of Figure 16 and Figure 17 indicate that despite the different control signals, the EKF produces good state predictions. It should be noted, however, that the performance may still be improved by adjusting the state and process covariance matrices even further.

### Case 2

Using a different measurement model of the form: $\left[\sqrt{x^2+y^2} \quad tan\theta\right]^T + \delta_t$, with a measurement covariance matrix of: $diag(0.5, 1)$, the following Jacobians are obtained:

$$G = \begin{bmatrix} 1 & 0 & -delta\_d \cdot \sin\left(\theta + \frac{\Delta\theta}{2}\right) \\ 0 & 1 & delta\_d \cdot \cos\left(\theta + \frac{\Delta\theta}{2}\right) \\ 0 & 0 & 1 \end{bmatrix}$$

$$H = \begin{bmatrix} \frac{x^2}{\sqrt{x^2+y^2}} & \frac{y^2}{\sqrt{x^2+y^2}} & 0 & 0 \\ \frac{x}{x^2+y^2} & \frac{y}{x^2+y^2} & 0 & 0 \end{bmatrix}$$

With the same initial conditions as in Case 1 and using a randomized control signal, the following is obtained:

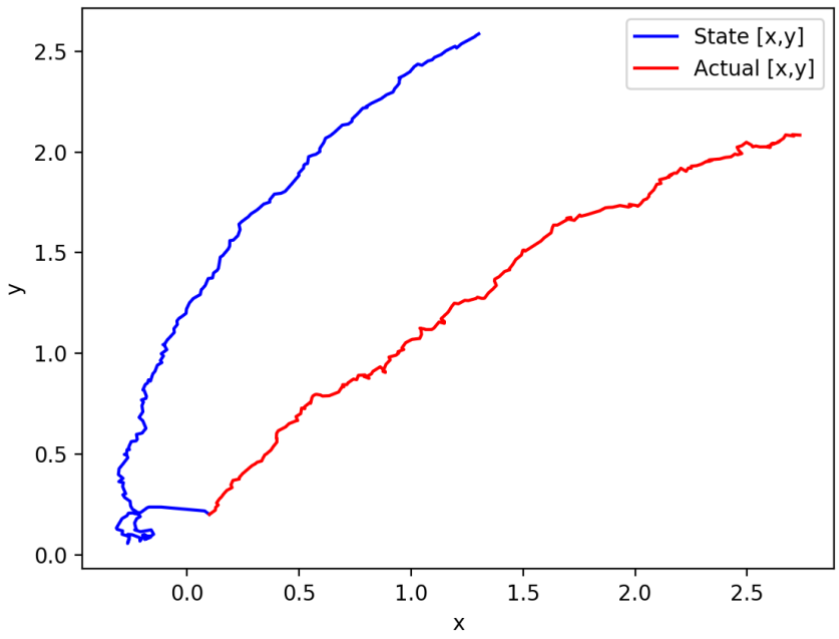


*Figure 18 - EKF for a Random Control Signals for Problem 4, Measurement Case 2*

Figure 18 indicates a drastic difference in the EKF prediction versus the actual position. Furthermore, this is clearly distinct result compared to Figure 16, which produces a relatively accurate prediction. Potential reasons for this differing result include unsuitable covariance matrices, since the covariance matrices have been left the same as for Case 1. Therefore, for a randomized control signal, the EKF parameters need tuning when different measurement models are used.

Again, with the same initial conditions as in Case 1 and using a sinusoidal control signal, the following is obtained:

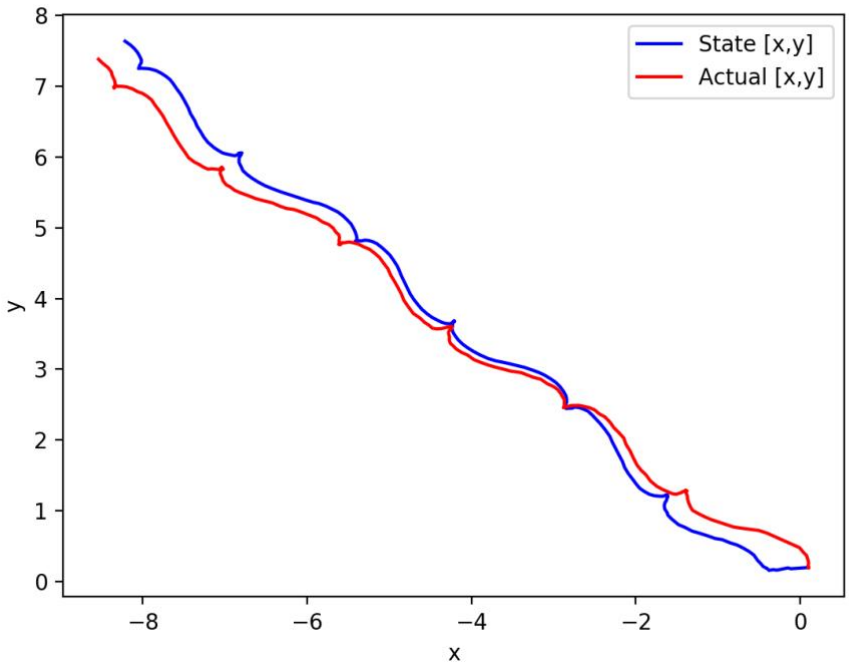


*Figure 19 - EKF for a Sinusoidal Control Signals for Problem 4, Measurement Case 2*

Figure 19 indicates that with a sinusoidal input, the EKF produces relatively accurate state predictions. These results are similar to those of Figure 17, and indicates that even if the covariance matrices are not changed, then the EKF produces relatively accurate results when a sinusoidal control signal is provided.

## Parameter Testing:

Process and state covariance matrices are adjusted to observe their effects on the system. This section will only consider a randomized control signal, since the effects were the most noticeable between the predictions with different measurement models, as previously noted:

1. Setting the following parameters:

$$Q = zeros((3,3))$$
$$P = identity(3)$$

a) With the measurement model in Case 1:

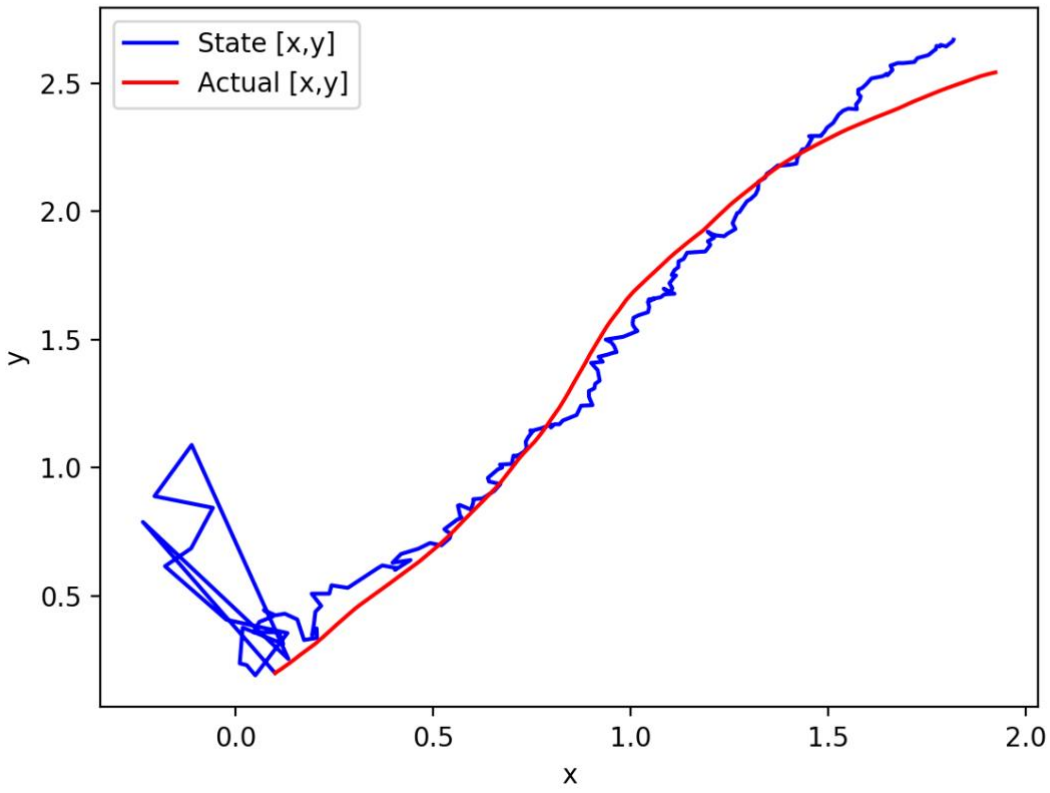


*Figure 20 – Parameter Testing 1a*

b) With the measurement model in Case 2:

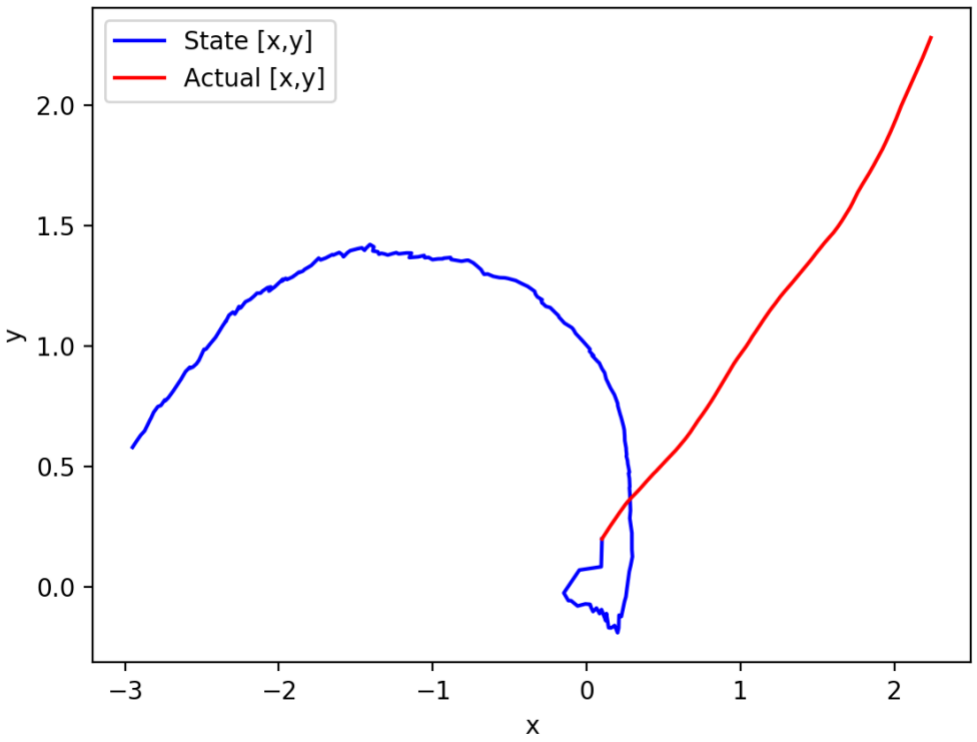


*Figure 21 - Parameter Testing 1b*

Between the two graphs - Figure 20 and Figure 21 – it is clear that this pair of covariance matrices best suits the first case. However, the predictions appear noisy, especially at the start. Therefore, the parameters should be further tuned. It is also interesting to note the differences in behaviours – whereas the first case better suits the graph, the sinusoidal control signal in the second case does not react well with the covariance matrices.

2. Setting the following parameters:

$$Q = zeros((3,3))$$

$$P = \begin{bmatrix} 0.001 & 0.001 & 0.01 \\ 0.001 & 0.001 & 0.00001 \\ 0.01 & 0.001 & 0.0001 \end{bmatrix}$$

a) With the measurement model in Case 1:

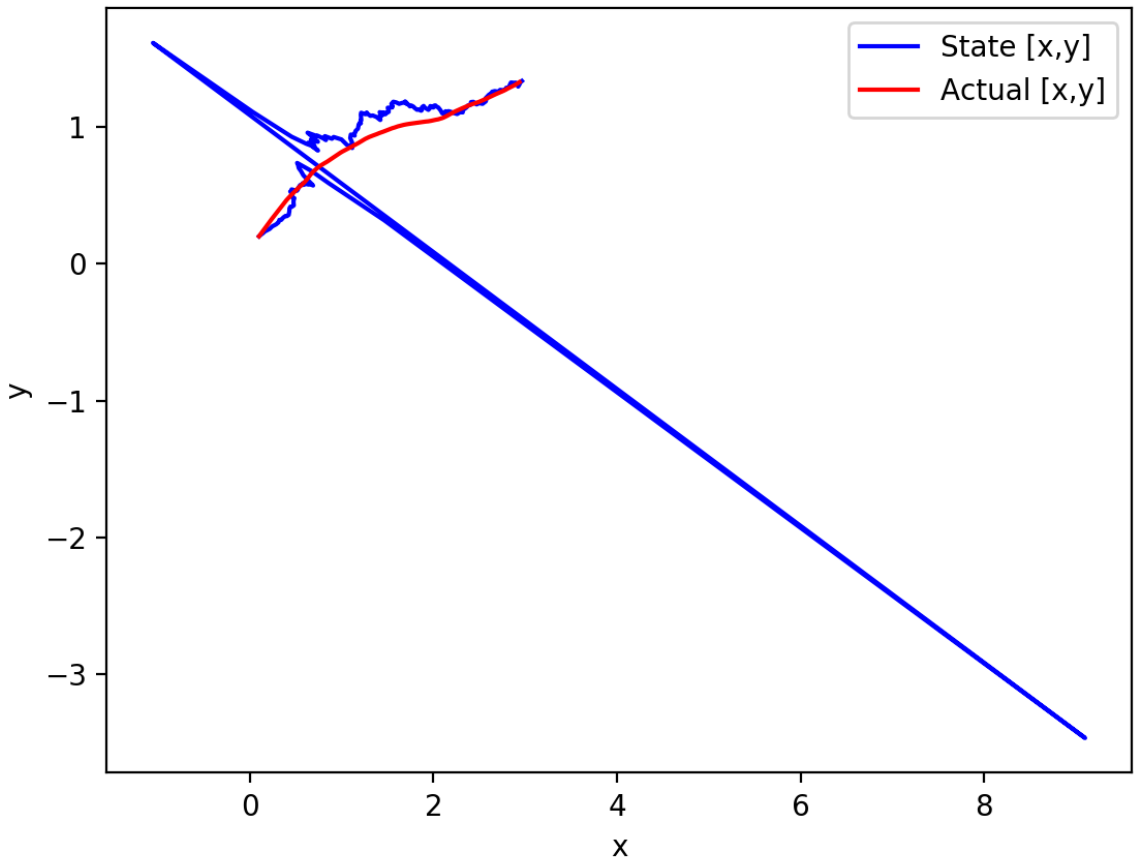


*Figure 22 - Parameter Testing 2a*

b) With the measurement model in Case 2:

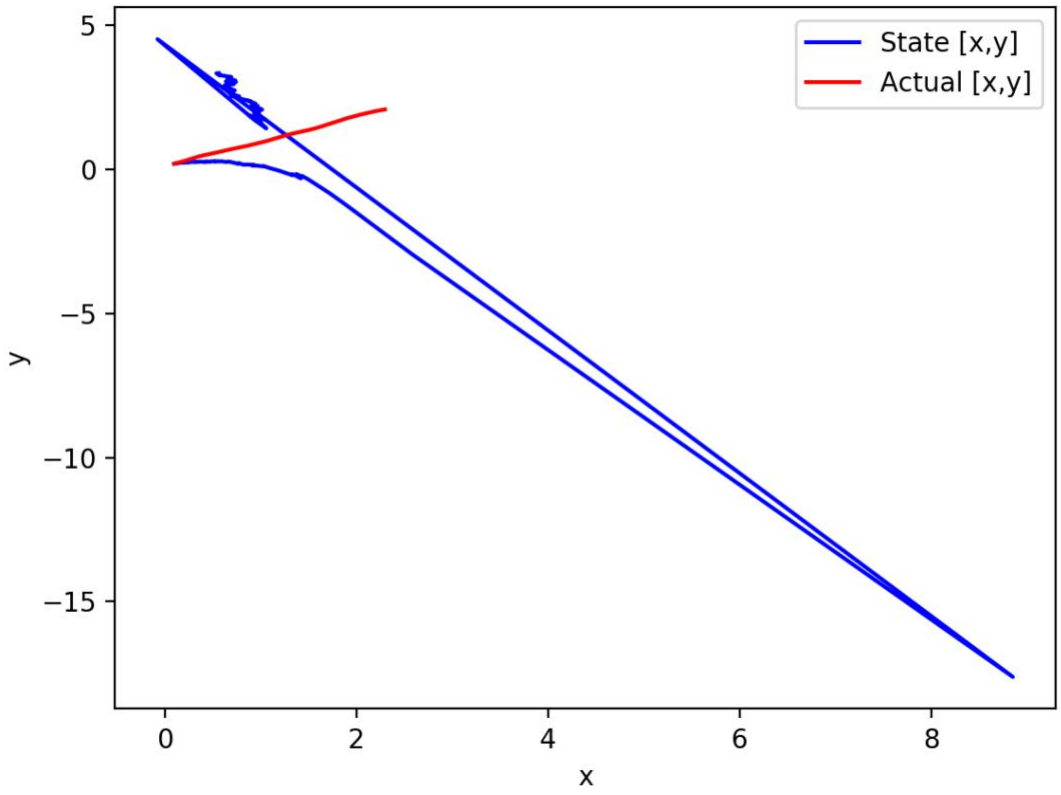


*Figure 23 - Parameter Testing 2b*

For both Figure 22 and Figure 23, it is clear that the covariance matrices do not suit either case. The EKF predictions appear vastly disconnected to the actual positions. Therefore, further tuning is required.

3. Setting the following parameters:

$$Q = \begin{bmatrix} 0.0001 & 0 & 0 \\ 0 & 0.0001 & 0 \\ 0 & 0 & 0.0001 \end{bmatrix}$$

$$P = \begin{bmatrix} 0.1 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.0001 \\ 0.01 & 0.001 & 0.001 \end{bmatrix}$$

a) With the measurement model in Case 1:

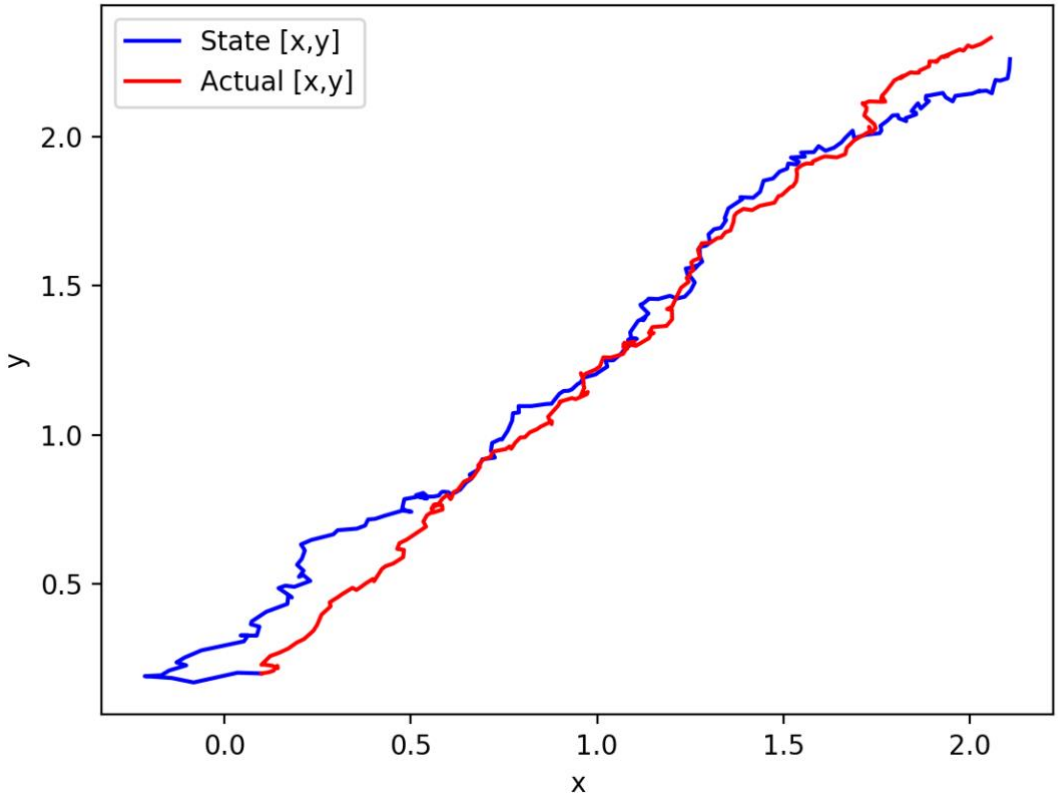


*Figure 24 - Parameter Testing 3a*

b) With the measurement model in Case 2:

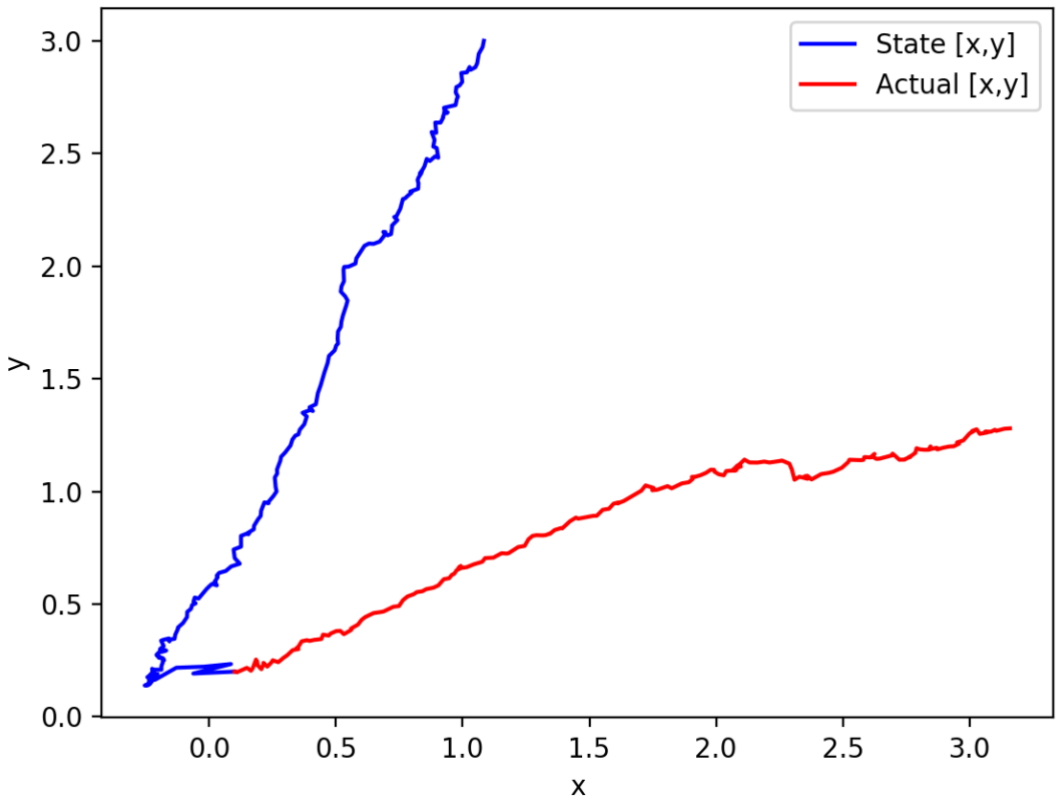


*Figure 25 - Parameter Testing 3b*

Between Figure 24 and Figure 25 it is clear that the pair of covariance matrices better suit the first case. In the first case, the EKF makes predictions relatively close to the actual state, however, in the second case – when a sinusoidal control signal is provided – the EKF tends to completely diverge from the actual position.

# Appendix A – Relevant Code for Part 2

The EKF algorithm is defined in a function, where all state and sensor values, as well as covariance matrices are passed. Within the function, the Jacobian arrays must be changed if the motion and/or measurement models are changed. The following is a snippet of the code:

```
#*********************************************************************
def ekf(xhat, u, z, Q, R, P, dt, l):
    xhat_k =  np.add(np.array([xhat]), np.array([[dt*u[0]*cos(xhat[3]), dt*u[0]*sin(xhat[3]), dt*u[1], dt*u[0]*tan(xhat[2])/l]]))
    G = np.array([[1, 0, 0, -dt*u[0]*sin(xhat[3])], [0, 1, 0, dt*u[0]*cos(xhat[3])], [0, 0, 1, 0], [0, 0, dt*u[0]*(1/(cos(xhat[2])**2))/l, 1]])

    # Linearized Measurement Model Jacobian Matrix
    H = np.array([[xhat[0]/sqrt(xhat[0]**2 + xhat[1]**2), xhat[1]/sqrt(xhat[0]**2 + xhat[1]**2), 0, 0], [-xhat[1]/(xhat[0]**2 + xhat[1]**2), xhat[0]/(xhat[0]
**2 + xhat[1]**2), 0, 0]])

    P_predict = np.linalg.multi_dot([G,P,G.T]) + Q

    K = np.dot(np.dot(P_predict, H.T),np.linalg.inv((np.linalg.multi_dot([H,P_predict,H.T]) + R)))

    # Transposing here is required since xhat was initialized as [1x4] instead of [4x1]
    xhat = np.add(xhat_k, np.dot(K, (np.subtract(np.array([z]).T, np.dot(H,xhat_k.T)))).T)
    P = np.dot((np.subtract(np.identity(4), np.dot(K,H))), P_predict)

    return xhat, P
```

*Figure 26 - EKF Algorithm in Code for Problem 2*

## Appendix B – Parameter Selection Process

The following are code snippets, with commented code, showing the iterations made during the parameter tuning process:

1. Fix the process noise covariance matrix to the most accurate result, the zero matrix:

```
# Process Noise Covariance Matrix
# Q = np.identity(3)
Q = np.zeros((3,3))
# Q = np.identity(3)*(0.01**2)
# Q = np.diag(np.array([0.01, 0.01, 0.0001]))*0.01
```

*Figure 27 - Parameter Selection Process Step 1*

2. Filter through values of the state covariance matrix, until the most accurate one is found:

```
# State Covariance Matrix
# P = np.identity(3)
# P = np.zeros((3,3))
# P = np.array([[0.001, 0.001, 0.01], [0.001, 0.001, 0.00001], [0.01, 0.001, 0.0001]])
P = np.array([[0.1, 0.01, 0.01], [0.01, 0.01, 0.0001], [0.01, 0.001, 0.001]])
```

*Figure 28 - Parameter Selection Process Step 2*

3. Fix the state covariance matrix at the most accurate solution, then filter through Q values until the most accurate solution is found:

```
# Process Noise Covariance Matrix
# Q = np.identity(3)
# Q = np.zeros((3,3))
# Q = np.identity(3)*(0.01**2)
Q = np.diag(np.array([0.01, 0.01, 0.0001]))*0.01
```

*Figure 29 - Parameter Selection Process Step 3*

4. Select the most accurate solution:

```
# Process Noise Covariance Matrix
# Q = np.identity(3)
# Q = np.zeros((3,3))
Q = np.identity(3)*(0.01**2)
# Q = np.diag(np.array([0.01, 0.01, 0.0001]))*0.01
```

*Figure 30 - Parameter Selection Process Step 4*