Aniket Sanjay Gujarathi
1007629979

Assignment 2

AER 1513
November 2, 2021

Problem 1

In the given sensor error plots, we can clearly notice that the distributions do not resemble a Gaussian. This is because in real-world, it is not possible to get noise resembling a perfect Gaussian distribution. As real-world is non-linear, the mean and the mode of the distribution will never be perfectly aligned, thus violating the property of the Gaussian distribution of $mean = mode = median$.
However, for the case of this assignment, the data has been particularly smoothened out, giving synchronized and continuous readings through linear interpolation, and avoiding any data dropouts due to the data arriving asynchronously. Further, to avoid processing raw laser rangefinder scans, the range/bearing is calculated using the groundtruth data obtained from a precise motion-capture system. This ensures that the data to be used is less noisy and can be fit approximately with a Gaussian. Although the mean of the distribution in figures 2.5, 2.6 are not exactly zero, they are close to zero. Also, the mode in the real data do not perfectly align with the mean, but are close.
In short, there are a wide variety of sources of noise, affected through completely random processes. Hence, it is impossible to account for every source of noise. However, as these noises are sums of a large number of independant factors, we can apply the central limit theorem which states that the sum of independant random variables can be well approximated by a Gaussian random variable. Hence, the assumption of a zero-mean Gaussian noise is reasonable.
The values for the variances are :

$$\mathbf{Q_k} = \begin{bmatrix} \sigma_v^2 & 0 \\ 0 & \sigma_\omega^2 \end{bmatrix}, \mathbf{R_k} = \begin{bmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_b^2 \end{bmatrix} \tag{1}$$

$$\tag{2}$$

where, $\sigma_v = v_{var}, \sigma_\omega = om_{var}, \sigma_r = r_{var}, \sigma_b = b_{var}$.

Problem 2

For the given motion and observation models the expressions for the Jacobians used in the EKF are as follows:

For the motion model:

$$\begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} = \begin{bmatrix} x_{k-1} \\ yk-1 \\ \theta_{k-1} \end{bmatrix} + T \begin{bmatrix} cos(\theta_{k-1} & 0 \\ sin(\theta_{k-1}) & 0 \\ 0 & 1) \end{bmatrix} (\begin{bmatrix} v_k \\ \omega_k \end{bmatrix} + \mathbf{w_k}) \tag{3}$$

The matrix equation can be split into seperate system of linear equations as follows:

$$f_1 = x_k = x_{k-1} + Tcos(\theta_{k-1})(v_k) + Tcos(\theta_{k-1})\mathbf{w_k} \tag{4}$$
$$f_2 = y_k = y_{k-1} + Tsin(\theta_{k-1})(v_k) + Tsin(\theta_{k-1})\mathbf{w_k} \tag{5}$$
$$f_3 = \theta_k = \theta_{k-1} + \omega_k + \mathbf{w_k} \tag{6}$$

The Jacobian elements with respect to the state variables are:

$$\frac{\partial f_1}{\partial x} = 1 \tag{7}$$
$$\frac{\partial f_1}{\partial y} = 0 \tag{8}$$
$$\frac{\partial f_1}{\partial \theta} = -Tsin(\theta)v_k|_{\hat{x}_{k-1},v_k,0} \tag{9}$$
$$\frac{\partial f_2}{\partial x} = 0 \tag{10}$$
$$\frac{\partial f_2}{\partial y} = 1 \tag{11}$$
$$\frac{\partial f_2}{\partial \theta} = Tcos(\theta)v_k|_{\hat{x}_{k-1},v_k,0} \tag{12}$$
$$\frac{\partial f_3}{\partial x} = 0 \tag{13}$$
$$\frac{\partial f_3}{\partial y} = 0 \tag{14}$$
$$\frac{\partial f_3}{\partial \theta} = 1 \tag{15}$$

Now the Jacobian $\mathbf{F'}$ is :

$$\mathbf{F'} = \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f1}{\partial y} & \frac{\partial f_1}{\partial \theta} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f2}{\partial y} & \frac{\partial f_2}{\partial \theta} \\ \frac{\partial f_3}{\partial x} & \frac{\partial f3}{\partial y} & \frac{\partial f_3}{\partial \theta} \end{bmatrix} \tag{16}$$

The Jacobian with respect to the process noise is :

$$\mathbf{w'_k} = \begin{bmatrix} Tcos(\theta_{k-1}) & 0 \\ Tsin(\theta_{k-1}) & 0 \\ 0 & T \end{bmatrix} \tag{17}$$

and $\mathbf{Q'_k} = \mathbf{w'_k Q_k w'^T_k}$.

For the observation model :

$$\begin{bmatrix} r_k^l \\ \phi_l^l \end{bmatrix} = \begin{bmatrix} \sqrt{dx^2 + dy^2} \\ atan2(dy, dx) - \theta_k \end{bmatrix} + \mathbf{n_k^l} \tag{18}$$

where;
$dx = x_l - x_k - dcos(\theta_k)$
$dy = y_l - y_k - dsin(\theta_k)$
the linear equations are:

$$f_1 = r_k^l = \sqrt{dx^2 + dy^2} + \mathbf{n_k^l} \tag{19}$$
$$f_2 = \phi_k^l = atan2(dy, dx) - \theta_k + \mathbf{n_k^l} \tag{20}$$

The Jacobian elements with respect to the state variables are:

$$\frac{\partial f_1}{\partial x} = \frac{-dx}{\sqrt{dx^2 + dy^2}}|_{\check{x}_k,0} \tag{21}$$
$$\frac{\partial f_1}{\partial y} = \frac{-dy}{\sqrt{dx^2 + dy^2}}|_{\check{x}_k,0} \tag{22}$$
$$\frac{\partial f_1}{\partial \theta} = \frac{(-dy * dcos(\theta) + dx * dsin(\theta))}{\sqrt{dx^2 + dy^2}}|_{\check{x}_k,0} \tag{23}$$
$$Let, A = \frac{dy}{dx} \tag{24}$$
$$\frac{\partial f_2}{\partial x} = (\frac{1}{1 + A^2})(\frac{dy}{dx^2})|_{\check{x}_k,0} \tag{25}$$
$$\frac{\partial f_2}{\partial y} = -(\frac{1}{1 + A^2})(\frac{1}{dx})|_{\check{x}_k,0} \tag{26}$$
$$\frac{\partial f_2}{\partial \theta} = (\frac{1}{1 + A^2})(\frac{-d * dxcos(\theta) + d * dysin(\theta)}{dx^2}) - 1|_{\check{x}_k,0} \tag{27}$$

and the Jacobian is :

$$\mathbf{G} = \begin{bmatrix} \frac{\partial f1}{\partial x} & \frac{\partial f1}{\partial y} & \frac{\partial f1}{\partial \theta} \\ \frac{\partial f2}{\partial x} & \frac{\partial f2}{\partial y} & \frac{\partial f2}{\partial \theta} \end{bmatrix} \tag{28}$$

The Jacobian with respect to the observation noise is:

$$\mathbf{M_k} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{29}$$
$$\mathbf{R_k'} = \mathbf{M_k} * \mathbf{R_k} * \mathbf{M_k^T} \tag{30}$$

Problem 3

The canonical EKF equations do not take into account variable number of exteroceptive measurements at each timestep. But in our case, our robot can apprehend multiple landmarks at a single timestep using the laser rangefinder measurements. Hence, we must account for these variable number of measurements and update our correction step accordingly. We can run the correction step for every measurement obtained from the laser rangefinder, and update our posterior in an iterative manner. This iterative refinement of the posterior through variable exteroceptive measurements leads to a better estimate than the one-shot estimate from the general EKF. Hence, higher the number of exteroceptive measurements from known landmarks, more accurate our estimate would be. The modified equations for the EKF would be as follows:

Algorithm 1 Modified EKF

$\check{\mathbf{P}}_\mathbf{k} = \mathbf{F}_{\mathbf{k-1}}\hat{\mathbf{P}}_{\mathbf{k-1}}\mathbf{F}_{\mathbf{k-1}}^\mathbf{T} + \mathbf{Q}'_\mathbf{k}$
$\check{\mathbf{x}}_\mathbf{k} = \mathbf{f}(\hat{\mathbf{x}}_{\mathbf{k-1}}, \mathbf{v}_\mathbf{k}, \mathbf{0})$ ▷ Prediction
for `i in l (exteroceptive measurements)` **do**
  $\mathbf{K}_\mathbf{k} = \check{\mathbf{P}}_\mathbf{k}\mathbf{G}_\mathbf{k}^\mathbf{T}(\mathbf{G}_\mathbf{k}\check{\mathbf{P}}_\mathbf{k}\mathbf{G}_\mathbf{k}^\mathbf{T} + \mathbf{R}'_\mathbf{k})^{-\mathbf{1}}$ ▷ Kalman Gain
  $\hat{\mathbf{P}}_\mathbf{k} = (\mathbf{1} - \mathbf{K}_\mathbf{k}\mathbf{G}_\mathbf{k})\check{\mathbf{P}}_\mathbf{k}$ ▷ Correction
  $\hat{\mathbf{x}}_\mathbf{k} = \check{\mathbf{x}}_\mathbf{k} + \mathbf{K}_\mathbf{k}(\mathbf{y}_\mathbf{k} - \mathbf{g}(\check{\mathbf{x}}_\mathbf{k}, \mathbf{0}))$
  $\check{\mathbf{P}}_\mathbf{k} = \hat{\mathbf{P}}_\mathbf{k}$ ▷ Reiterate over new estimates
  $\check{\mathbf{x}}_\mathbf{k} = \hat{\mathbf{x}}_\mathbf{k}$
end for
return $\hat{\mathbf{x}}_\mathbf{k}, \hat{\mathbf{P}}_\mathbf{k}$

Problem 4

a) $\hat{x_0} = x_0, (\hat{P_0}) = diag(1, 1, 0.1)$
We can see in Fig. 1, 2, 3 error plots, the error does not remain bounded by the 3-sigma bound. This shows that the estimator is not consistent. As a result of approximation of a non-linear system by linearizing about an optimal point, the ekf works well only in cases where the non-linearity is not extreme, which would cause large linearization errors. The operating point we choose for linearization is the mean, which is not the true state for a non-linear system. Hence, the EKF can be inconsistent.
For the case where multiple landmarks are seen, the estimate becomes more accurate as the number of visible landmarks increase. This is because the correction step is run for every visible landmark as explained in Alg. 1, thus improving our estimate over each iteration. Hence, in the case of rmax = 5m, the robot can see multiple landmarks at a point at a given timestep, thus giving reasonably accurate results. It is seen in Fig. 2 and Fig. 3 that as the

rmax decreases, the error values seem to increase. This is because, as rmax decreases, less number of landmarks are visible at a given timestep, thus less refinement of the posterior estimate.

1. rmax = 5m

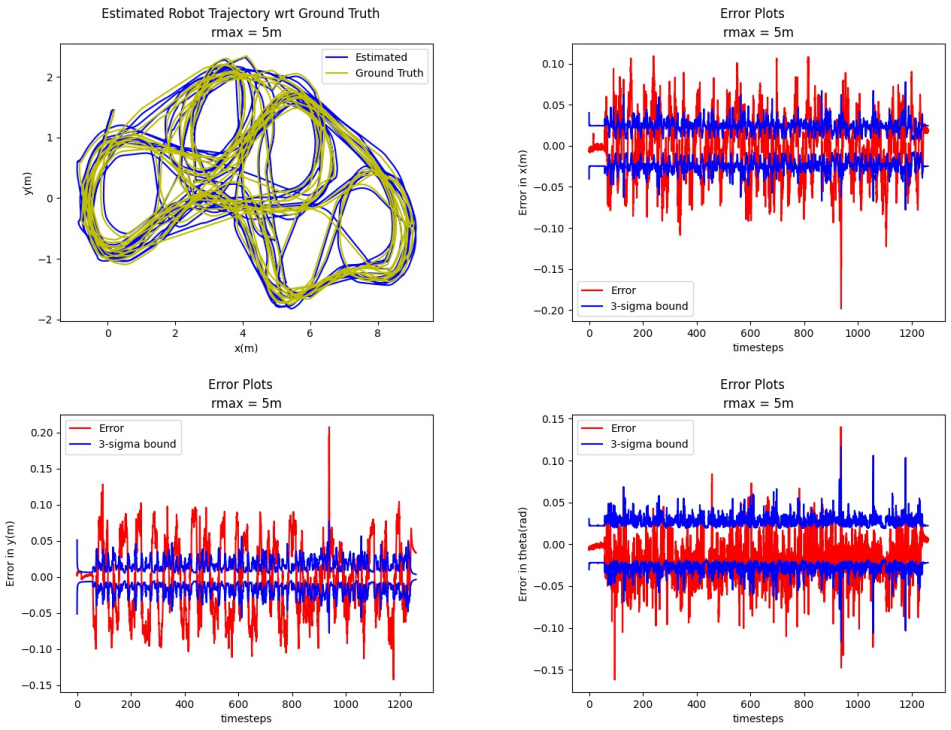

Figure 1: Estimated Trajectory and Estimated State Error Plots (rmax = 5m)

2. rmax = 3m - Fig. 2
3. rmax = 1m - Fig. 3

b) $\hat{x_0} = (1, 1, 0.1)$ - Poor initial condition
In this case, we have a bad initial estimate of our state. We can see in Fig. 4, 5, 6, the error seems to be very big initially, but slowly reduces. This is because initially, the robot does not have any correction from the landmarks and has a bad estimate. But as the robot moves and gets measurements from visible landmarks, the estimator corrects itself and tries to refine the state estimate. Hence, over a period of time the error starts reducing as landmarks are seen. You can notice that similar trends explained in part a can be seen in this case too.

1. rmax = 5m - Fig. 4
2. rmax = 3m - Fig. 5

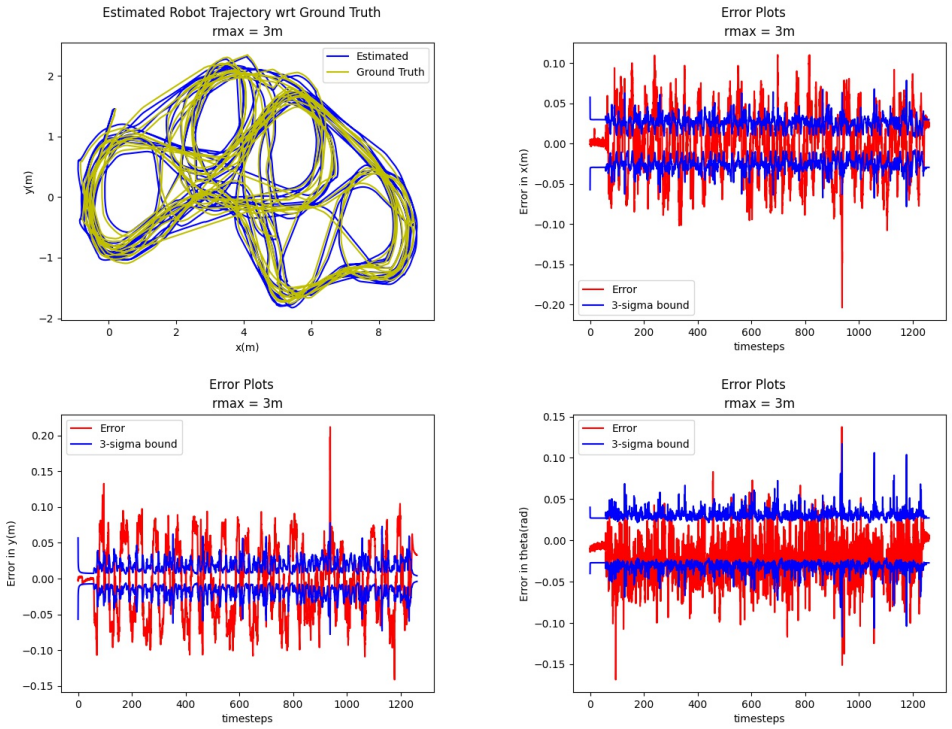

Figure 2: Estimated Trajectory and Estimated State Error Plots (rmax = 3m)

3. rmax = 1m - Fig. 6

   c) CRLB:
   In this case, all the Jacobians are calculated using the true robot state. We can see in Fig. 7, 8, 9, the estimates are bounded by the 3-sigma bound and are consistent. This is the Cramer-Rao Lower Bound condition for the EKF. We can still notice that the value of rmax governs the error plots giving better estimates for higher values of rmax and slightly bad for lower values. In Fig. 9 we can clearly see that initially, the error is very high, but reduces as the robot moves and correction is done when a landmark is seen. In contrast to part a, we now have a consistent and an unbiased system with mean error near zero.

   (a) rmax = 5m

   (b) rmax = 3m

   (c) rmax = 1m

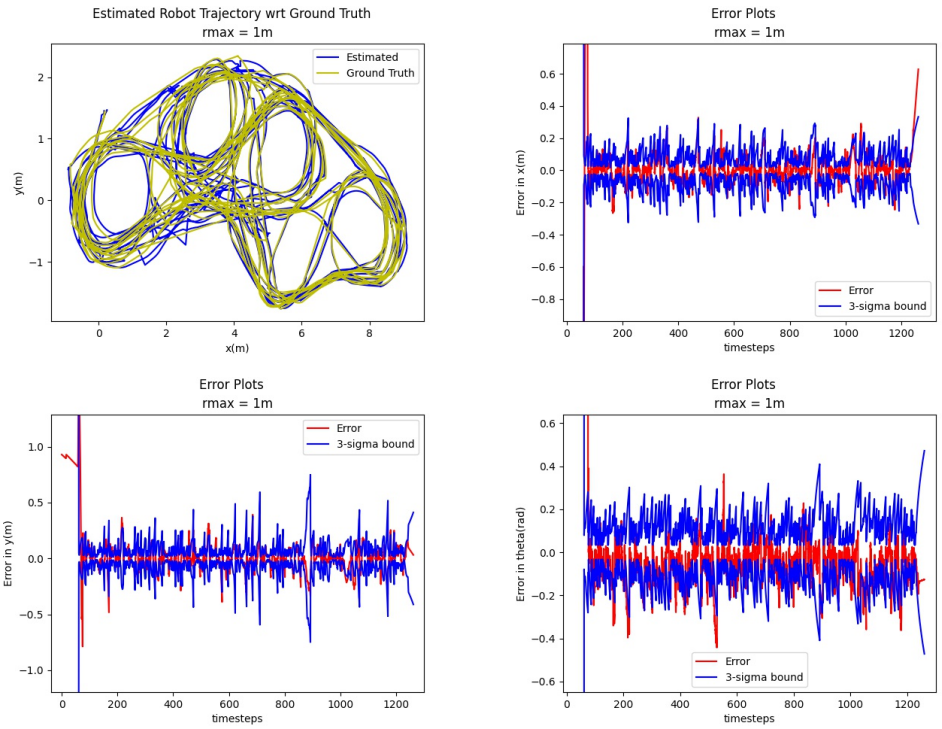

Figure 3: Estimated Trajectory and Estimated State Error Plots (rmax = 1m)

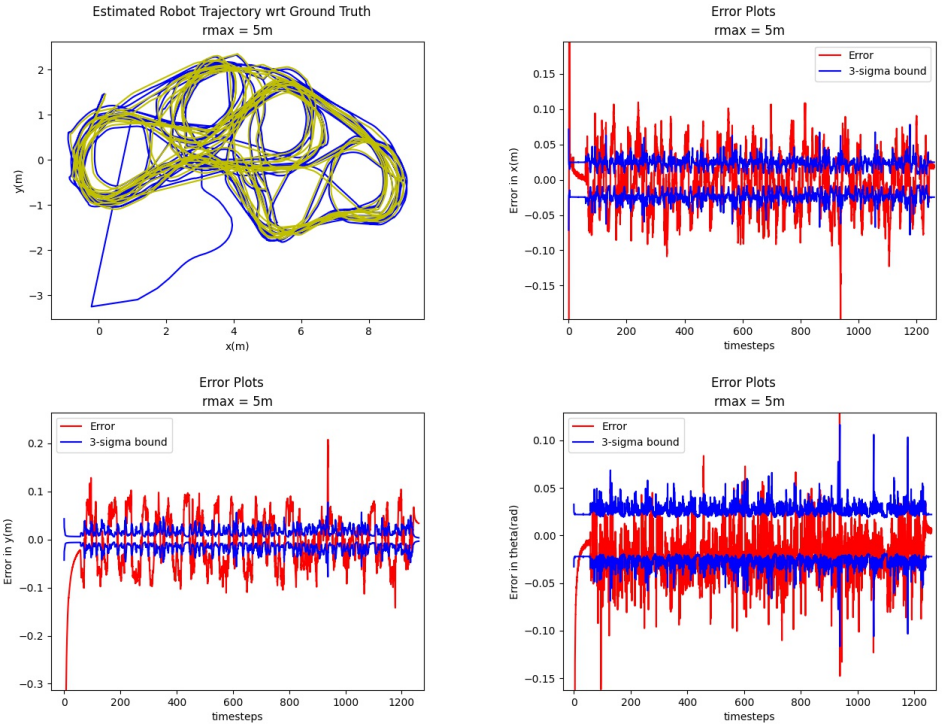

Figure 4: Estimated Trajectory and Estimated State Error Plots (rmax = 3m)

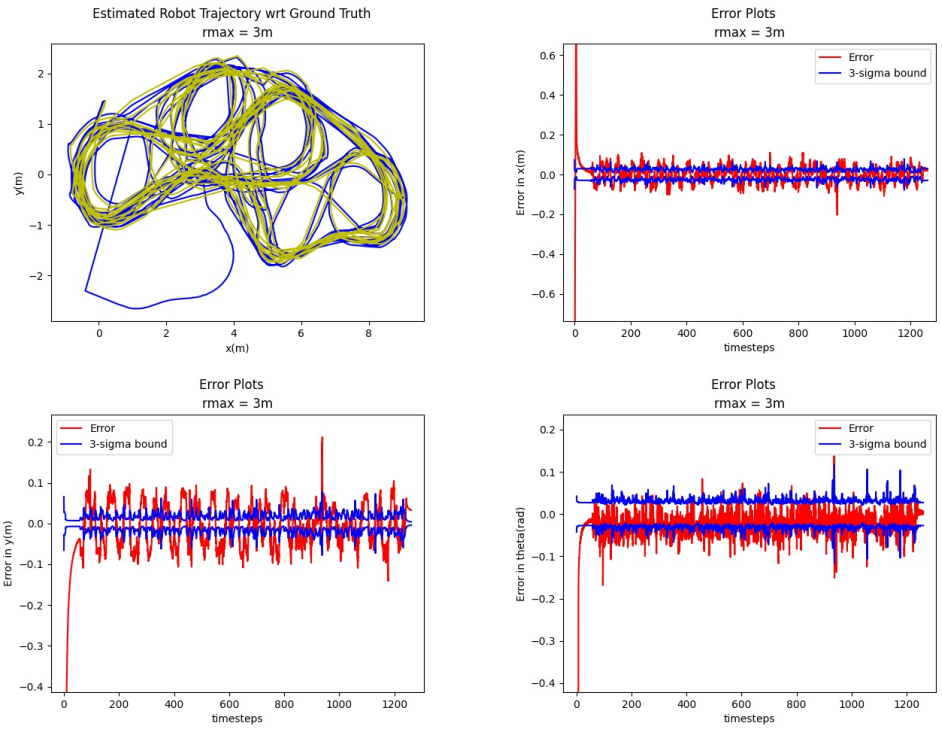

Figure 5: Estimated Trajectory and Estimated State Error Plots (rmax = 3m)

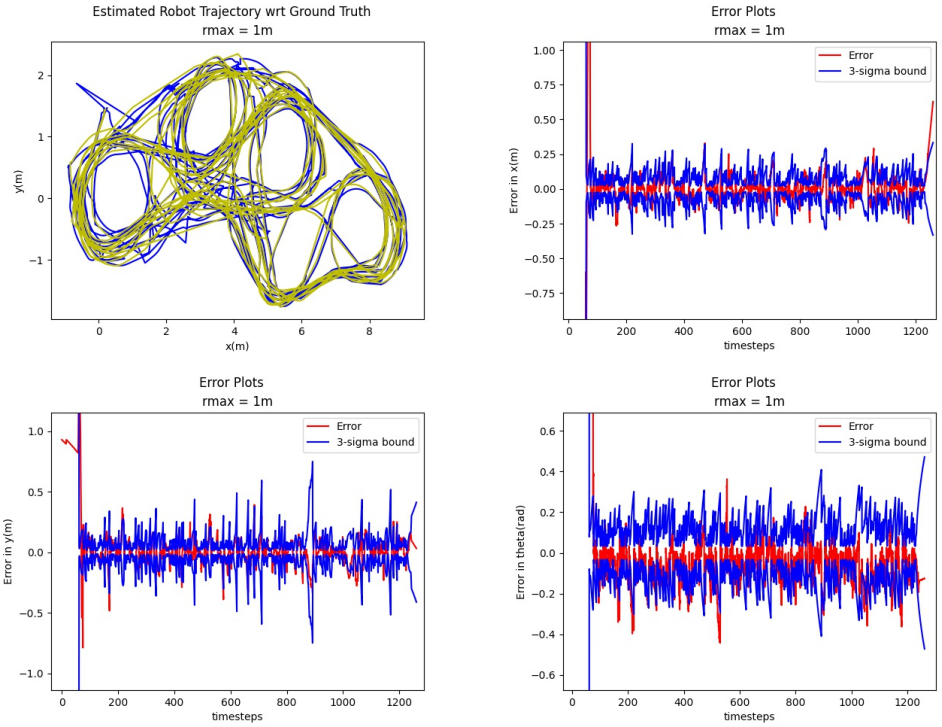

Figure 6: Estimated Trajectory and Estimated State Error Plots (rmax = 1m)

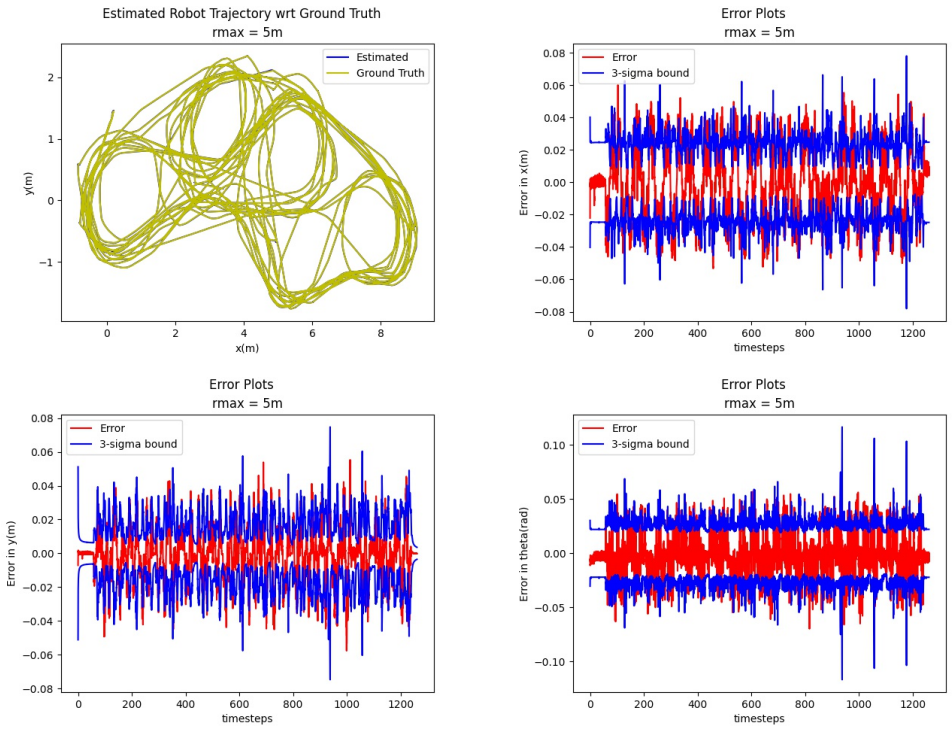

Figure 7: Estimated Trajectory and Estimated State Error Plots (rmax = 5m)

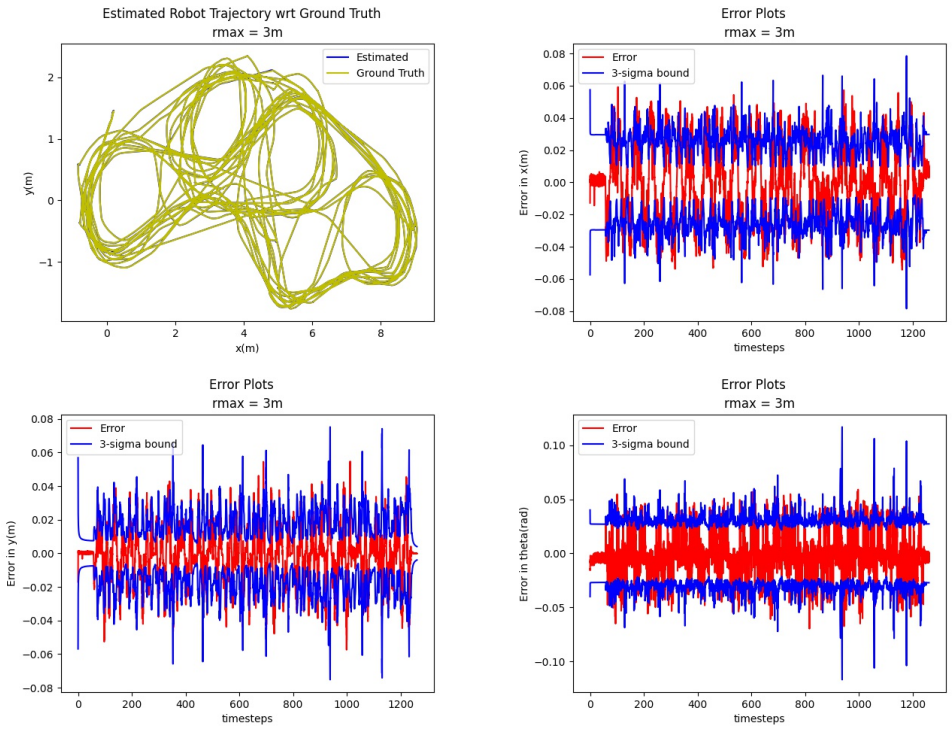

Figure 8: Estimated Trajectory and Estimated State Error Plots (rmax = 3m)

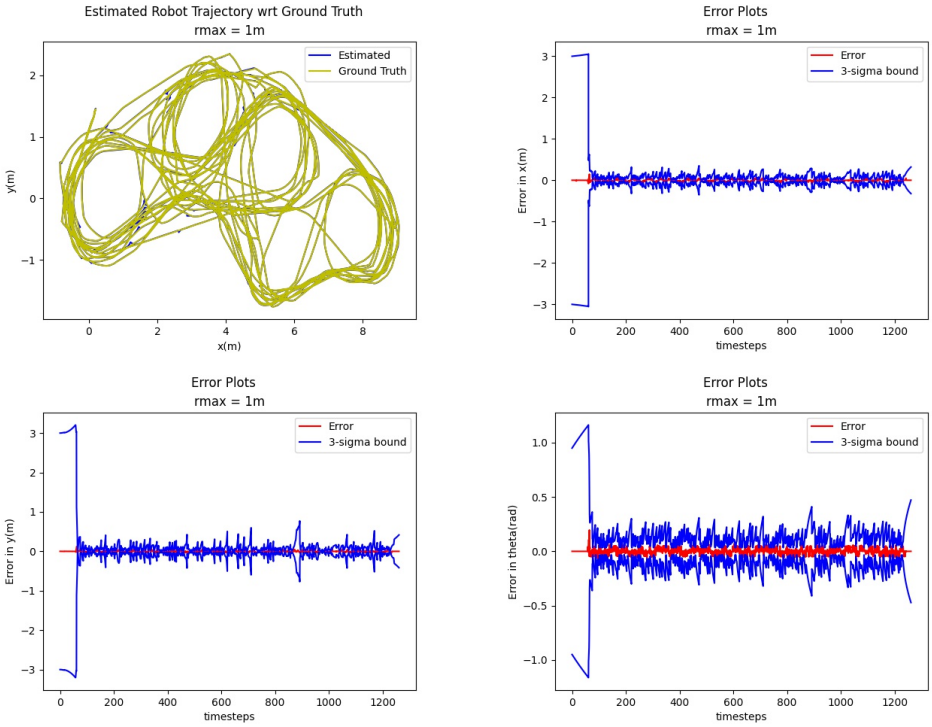

Figure 9: Estimated Trajectory and Estimated State Error Plots (rmax = 1m)

Appendix

Code for estimating the state and covariance values using EKF.

```
import scipy.io as scp
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Ellipse
import matplotlib.transforms as transforms

# global data

data = scp.loadmat('dataset2.mat')
#  seperate out
t = data['t']
t = t.reshape((12609,1))
x_true = data['x_true']
y_true = data['y_true']
th_true = data['th_true']
true_valid = data['true_valid']
l = data['l']
r = data['r']
r_var = data['r_var']
b = data['b']
b_var = data['b_var']
v = data['v']
v_var = data['v_var']
om = data['om']
om_var = data['om_var']
d = data['d']

# ask for rmax
val = input("r_max: ")
rmax = val
# CRLB
CRLB = input('CRLB (y/n): ')


# visualization script
def plotting(x_hat, sigx, sigy, sigth):
    # plot trajectory
```

```
plt.figure(0)
plt.plot(x_hat[:, 0], x_hat[:, 1], 'b', label='Estimated')
plt.plot(x_true, y_true, 'y', label='Ground Truth')
plt.suptitle("Estimated Robot Trajectory wrt Ground Truth")
plt.title("rmax = {}m".format(rmax))
plt.xlabel("x(m)")
plt.ylabel("y(m)")
plt.legend()
# plt.savefig("/home/aniket/Desktop/Courses_Fall/AER1513/Assignments/State

# error plots
ex = (x_hat[:, 0].reshape((x_true.shape)) - x_true)
ey = (x_hat[:, 1].reshape(y_true.shape) - y_true)
etheta = (x_hat[:, 2].reshape(th_true.shape) - th_true)
etheta = [wrap(x) for x in etheta]
print(np.mean(ex), np.mean(ey), np.mean(etheta))

plt.figure(1)
plt.plot(t[1:], ex[1:], 'r', label='Error')
plt.plot(t[1:], np.sqrt(sigx[1:])*3, 'b', label="3-sigma bound")
plt.plot(t[1:], -np.sqrt(sigx[1:])*3, 'b')
plt.suptitle('Error Plots')
plt.title('rmax = {}m'.format(rmax))
plt.xlabel('timesteps')
plt.ylabel('Error in x(m)')
plt.legend()
# plt.savefig("/home/aniket/Desktop/Courses_Fall/AER1513/Assignments/State

plt.figure(2)
plt.plot(t[1:], ey[1:], 'r', label='Error')
plt.plot(t[1:], np.sqrt(sigy[1:])*3, 'b', label="3-sigma bound")
plt.plot(t[1:], -np.sqrt(sigy[1:])*3, 'b')
plt.suptitle('Error Plots')
plt.title('rmax = {}m'.format(rmax))
plt.xlabel('timesteps')
plt.ylabel('Error in y(m)')
plt.legend()
# plt.savefig("/home/aniket/Desktop/Courses_Fall/AER1513/Assignments/State

plt.figure(3)
plt.plot(t[1:], etheta[1:], 'r', label='Error')
plt.plot(t[1:], np.sqrt(sigth[1:])*3, 'b', label="3-sigma bound")
```

```
    plt.plot(t[1:], -np.sqrt(sigth[1:])*3, 'b')
    plt.suptitle('Error Plots')
    plt.title('rmax = {}m'.format(rmax))
    plt.xlabel('timesteps')
    plt.ylabel('Error in theta(rad)')
    plt.legend()
    # plt.savefig("/home/aniket/Desktop/Courses_Fall/AER1513/Assignments/State

    # plt.show()

# function to draw the covariance ellipse
def cov_ellipse(P_hat, nstd=3):
    a, b, c = P_hat[0, 0], P_hat[0, 1], P_hat[1, 1]
    lambda_1 = 0.5 * (a + c) + np.sqrt((0.5*(a - c))**2 + b**2)
    lambda_2 = 0.5 * (a + c) - np.sqrt((0.5*(a - c))**2 + b**2)
    if (b == 0 and a >= c):
        th = 0
    elif(b == 0 and a < c):
        th = np.pi / 2
    else:
        th = np.arctan2(lambda_1 - a, b)

    ra = np.sqrt(lambda_1)
    rb = np.sqrt(lambda_2)


    return ra, rb, th

# wrap angles from [-pi, pi]
def wrap(theta):
    if theta > np.pi:
        theta = theta - 2*np.pi
    elif theta < -np.pi:
        theta = theta + 2*np.pi

    return theta

# initializations
def initialization():
    init_params = {}

    # number of timesteps
```

```
    init_params['T'] = data['t'].size

    # x0_hat, P0_hat (use right answer for initialization)
    init_params['x0_hat'] = np.array([[data['x_true'][0][0]], [data['y_true']
    # part b
    # init_params['x0_hat'] = np.array([[1], [1], [0.1]])
    init_params['P0_hat'] = np.diag([1, 1, 0.1])

    # Process noise at kth timestep
    init_params['Q_k'] = np.diag([data['v_var'][0][0], data['om_var'][0][0]])

    # Correction noise at kth timestep
    init_params['R_k'] = np.diag([data['r_var'][0][0], data['b_var'][0][0]])


    return init_params


#correction step
def correction(lk, rk, bk, R_k, x_check, P_check, k):

    # landmark positions
    x_l = lk[0]
    y_l = lk[1]

    # robot states from prediction
    if(CRLB == 'n'):
        x_k = x_check[0][0]
        y_k = x_check[1][0]
        theta = x_check[2][0]
    elif(CRLB == 'y'):
        x_k = x_true[k][0]
        y_k = y_true[k][0]
        theta = th_true[k][0]

    # wrap theta
    theta = wrap(theta)
    bk = wrap(bk)

    # distances
    dx = x_l - x_k - d[0][0]*np.cos(theta)
    dy = y_l - y_k - d[0][0]*np.sin(theta)
```

```
    # Jacobians
    G_k = np.zeros((2, 3))
    G_k[0][0] = -dx / np.sqrt(dx**2 + dy**2)
    G_k[0][1] = -dy / np.sqrt(dx**2 + dy**2)
    G_k[0][2] = (-dy*d*np.cos(theta) + dx*d*np.sin(theta)) / np.sqrt(dx**2 +
    A = dy/dx
    G_k[1][0] = (1/(1 + A**2))*(dy / dx**2)
    G_k[1][1] = - (1 / (1 + A**2)) * (1 / dx)
    G_k[1][2] = (1 / (1 + A**2)) * ((-d*np.cos(theta)*dx + dy*d*np.sin(theta)

    M_k = np.identity(2)
    R_k_prime = M_k @ R_k @ M_k.T

    # Kalman gain
    K_k = P_check @ G_k.T @ (np.linalg.inv(G_k @ P_check @ G_k.T + R_k_prime)

    # Correction
    Phat_k = (np.identity(3) - K_k @ G_k) @ P_check

    g_xcheck = np.array([[np.sqrt(dx**2 + dy**2)], [wrap(np.arctan2(dy, dx) -
    y_meas = np.array([[rk],[wrap(bk)]])

    xhat_k = x_check + K_k @ (y_meas - g_xcheck)
    xhat_k[2] = wrap(xhat_k[2])
    return Phat_k, xhat_k

def prediction(x_hat, P_hat, v, om, Qk,  delta_t, k):
    # CRLB condition
    if(CRLB == 'n'):
        x_hat = x_hat
    else:
        x_hat = np.array([x_true[k], y_true[k], th_true[k]])

    # wrap theta
    theta = wrap(x_hat[2])

    ## Prediction Step
    # get the Jacobians
    # wrt X
    F_km = np.array([[1, 0, -delta_t*np.sin(theta)*v], [0, 1, delta_t*np.cos(
    # wrt process noise
```

```
    W_k_prime = np.array([[delta_t*np.cos(theta), 0], [delta_t*np.sin(theta),
    Qk_prime = W_k_prime @ Qk @ W_k_prime.T

    Pcheck_k = F_km @ P_hat @ F_km.T + Qk_prime

    Z = np.array([[np.cos(theta), 0], [np.sin(theta), 0], [0, 1]], dtype='floa
    ip = np.array([[v[0]], [om[0]]], dtype='float32')

    xcheck_k = x_hat + delta_t * Z @ ip

    return xcheck_k, Pcheck_k



# ekf
def ekf():
    fig, ax = plt.subplots(1, 1)

    # initializations
    init_params = initialization()
    x_hat = init_params['x0_hat']
    P_hat = init_params['P0_hat']

    sig_x = init_params['P0_hat'][0][0]
    sig_y = init_params['P0_hat'][1][1]
    sig_th = init_params['P0_hat'][2][2]

    x_hat_plot = init_params['x0_hat'].T
    P_hat_plot = init_params['P0_hat']

    # main loop
    for k in range(1, init_params['T']):
        delta_t = t[k] - t[k-1]

        # Prediction Step
        xcheck_k, Pcheck_k = prediction(x_hat, P_hat, v[k], om[k], init_param

        # Check for landmark availability
        n = np.count_nonzero(r[k])
        idx = np.nonzero(r[k])
        idx_less = np.where(np.logical_and(r[k] > 0, r[k] < float(rmax)))
        if (n > 0):
```

```
        for i in idx[0]:
            if (r[k][i] < float(rmax)):
                # Correction Step
                P_hat, x_hat = correction(l[i], r[k][i], b[k][i], init_par

                xcheck_k = x_hat
                Pcheck_k = P_hat
            else:
                P_hat, x_hat = Pcheck_k, xcheck_k

        x_hat_plot = np.vstack([x_hat_plot, x_hat.T])
        sig_x = np.vstack([sig_x, P_hat[0][0]])
        sig_y = np.vstack([sig_y, P_hat[1][1]])
        sig_th = np.vstack([sig_th, P_hat[2][2]])
    else:
        # Prediction Step
        P_hat, x_hat = Pcheck_k, xcheck_k
        x_hat_plot = np.vstack([x_hat_plot, x_hat.T])
        sig_x = np.vstack([sig_x, P_hat[0][0]])
        sig_y = np.vstack([sig_y, P_hat[1][1]])
        sig_th = np.vstack([sig_th, P_hat[2][2]])


    #animation
    plt.clf()
    ax = plt.gca()

    plt.suptitle('Animation')
    plt.title('rmax = 1m')
    plt.xlim(-3, 11)
    plt.ylim(-3, 4)
    plt.scatter(l[:, 0], l[:, 1], zorder=100, s=30, c='black', label='Lan
    plt.scatter(l[idx_less[0]][:, 0], l[idx_less[0]][:, 1], zorder=100, s
    plt.scatter(x_hat[0][0], x_hat[1][0], zorder=100, s=20, c='r', label=
    plt.scatter(x_true[k][0], y_true[k][0], zorder=100, s=20, c='b', labe

    # variance ellipse
    ra, rb, th = cov_ellipse(P_hat)
    ellipse = Ellipse(xy = (x_hat[0][0], x_hat[1][0]), zorder=0, width =
    ax.add_artist(ellipse)

    # find end point with angle
```

```
            endx = 0.5*np.cos(wrap(x_hat[2][0]))
            endy = 0.5*np.sin(wrap(x_hat[2][0]))
            endx_true = 0.5*np.cos(wrap(th_true[k][0]))
            endy_true = 0.5*np.sin(wrap(th_true[k][0]))
            plt.arrow(x_hat[0][0], x_hat[1][0], endx, endy, color='r')
            plt.arrow(x_true[k][0], y_true[k][0], endx_true, endy_true, color='b'

            plt.legend()
            plt.xlabel('x(m)')
            plt.ylabel('y(m)')
            plt.draw()
            plt.pause(0.000000001)

        plotting(x_hat_plot, sig_x, sig_y, sig_th)

        return P_hat, x_hat


if __name__ == "__main__":
    ekf()
```