Assignment 2: Sensor fusion for lawn mower

For the kinematic model, we have four states, $[V, \varphi, X, Y]$ and write it kinematic model functions as follows:

$$V_n = \frac{v_R(t) + v_L(t)}{2}$$

$$\varphi_n = \varphi_{n-1} + \frac{v_R(t) - v_L(t)}{2R} \cdot \Delta t$$

$$X_n = X_{n-1} + \frac{v_R(t) + v_L(t)}{2} \cdot \cos\left(\varphi_{n-1} + \frac{v_R(t) - v_L(t)}{2R} \cdot \Delta t\right) \cdot \Delta t$$

$$Y_n = Y_{n-1} + \frac{v_R(t) + v_L(t)}{2} \cdot \sin\left(\varphi_{n-1} + \frac{v_R(t) - v_L(t)}{2R} \cdot \Delta t\right) \cdot \Delta t$$

where $v_R(t)$ and $v_L(t)$ are control vectors, u

I use kinematic model in Kalman filter

So, $f(x_{n-1}, u)$, in my Kalman filter,

$$f(x) = \begin{bmatrix} V \\ \varphi \\ X \\ Y \end{bmatrix} = \begin{bmatrix} \frac{v_R(t) + v_L(t)}{2} \\ \varphi_{n-1} + \frac{v_R(t) - v_L(t)}{2R} \cdot \Delta t \\ X_{n-1} + \frac{v_R(t) + v_L(t)}{2} \cdot \cos\left(\varphi_{n-1} + \frac{v_R(t) - v_L(t)}{2R} \cdot \Delta t\right) \cdot \Delta t \\ Y_{n-1} + \frac{v_R(t) + v_L(t)}{2} \cdot \sin\left(\varphi_{n-1} + \frac{v_R(t) - v_L(t)}{2R} \cdot \Delta t\right) \cdot \Delta t \end{bmatrix}$$

By using Jacobians, we can get following functions

$$F(x) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -\frac{v_R(t) + v_L(t)}{2} \cdot \sin\left(\varphi_{n-1} + \frac{v_R(t) - v_L(t)}{2R} \cdot \Delta t\right) \cdot \Delta t & 1 & 0 \\ 0 & \frac{v_R(t) + v_L(t)}{2} \cdot \cos\left(\varphi_{n-1} + \frac{v_R(t) - v_L(t)}{2R} \cdot \Delta t\right) \cdot \Delta t & 0 & 1 \end{bmatrix}$$

We have crude positioning in GNSS data, which include X and Y positions, so that the measurement equation is

$$h(x) = \begin{bmatrix} X \\ Y \end{bmatrix}$$

By using Jacobians

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

After getting the matrix expressions for those above, we can Kalman filter.

In this problem, the robot has a 10 Hz GNSS sensor, so the time step is 0.1s

Predict model:

$$\hat{x}_k = f(\hat{x}_{k-1}, u_{k-1})$$

$$P_k = F_{k-1}P_{k-1}F_{k-1}^T + Q_{k-1}$$

Update model:

$$G_k = P_k H_k^T (H_k P_k H_k^T + R)^{-1}$$

$$\hat{x}_k \leftarrow \hat{x}_k G_k (z_k - h(\hat{x}_k))$$

$$P_k \leftarrow (I - G_k H_k)\, P_k$$

First, we start with custom noise.

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

$$Q = A \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R = B \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

A,B are the parameters of the matrices P,R and need to be adjusted in the course of the operation.

I implemented the EKF in python and mainly plotted in X position and Y position.

First, I defined motion model, which calculates the robot kinematic, and measurement model, which used to measure status values.

```
def MotionModel(x, u, T):
    R = 0.27
    Fx = np.array([[0, 0, 0, 0],
                   [0, 1, 0, 0],
                   [0, -((u[0] + u[1]) / 2) * np.sin(x[1] + (u[0] - u[1]) / 2 / R) * T, 1, 0],
                   [0, ((u[0] + u[1]) / 2) * np.cos(x[1] + (u[0] - u[1]) / 2 / R) * T, 0, 1]])

    fx = np.array([(u[0] + u[1]) / 2,
                   x[1] + ((u[0] - u[1]) / 2 / R) * T,
                   x[2] + ((u[0] + u[1]) / 2) * np.cos(x[1] + (u[0] - u[1]) / 2 / R) * T * T,
                   x[3] + ((u[0] + u[1]) / 2) * np.sin(x[1] + (u[0] - u[1]) / 2 / R) * T * T])

    return fx, Fx
```

Motion model

```python
def MeasurementModel(x):
    hx = np.array([x[2], x[3]])
    Hx = np.array([[0, 0, 1, 0],
                   [0, 0, 0, 1]])
    return hx, Hx
```

Then, I defined the prediction process and update process.

```python
def nonLinKFprediction(x, u, P, f, Q):
    fx, Fx = f(x, u)
    x = fx
    P = Fx @ P @ Fx.T + Q
    return x, P
```

Prediction

```python
def nonLinKFupdate(x, P, y, h, R):
    hx, Hx = h(x)
    S = Hx @ P @ Hx.T + R
    K = P @ Hx.T @ np.linalg.inv(S)
    P = P - K @ S @ K.T
    x = x + K @ (y - hx)
    return x, P
```

Update

Finally, I defined the Kalman filter. In the function, ***xf*** represents the filtered estimates for times 1 to N, ***Pf*** represents the filter error covariance, ***xp*** represents the predicted estimates for times 1 to N and Pp represents the filter predicted error covariance. In the function, I initialize all four matrices.

```python
def nonLinearKalmanFilter(Y, x_0, u, P_0, f, Q, h, R):
    n = x_0.shape[0]
    m = Y.shape[0]
    N = Y.shape[1]

    xf = np.zeros((n, N))
    Pf = np.zeros((n, n, N))
    xp = np.zeros((n, N))
    Pp = np.zeros((n, n, N))

    xp[:, 0], Pp[:, :, 0] = nonLinKFprediction(x_0, u[:, 0], P_0, f, Q)
    xf[:, 0], Pf[:, :, 0] = nonLinKFupdate(xp[:, 0], Pp[:, :, 0], Y[:, 0], h, R)

    for i in range(1, N):
        xp[:, i], Pp[:, :, i] = nonLinKFprediction(xf[:, i - 1], u[:, i], Pf[:, :, i - 1], f, Q)
        xf[:, i], Pf[:, :, i] = nonLinKFupdate(xp[:, i], Pp[:, :, i], Y[:, i], h, R)

    return xf, Pf, xp, Pp
```

Kalman filter

I plotted my Kalman filter test results which compare with the ground truth.

1. Covariance values

I choose the results of the state estimation at every 5 time steps and plot the ellipse according to the corresponding covariance matrix

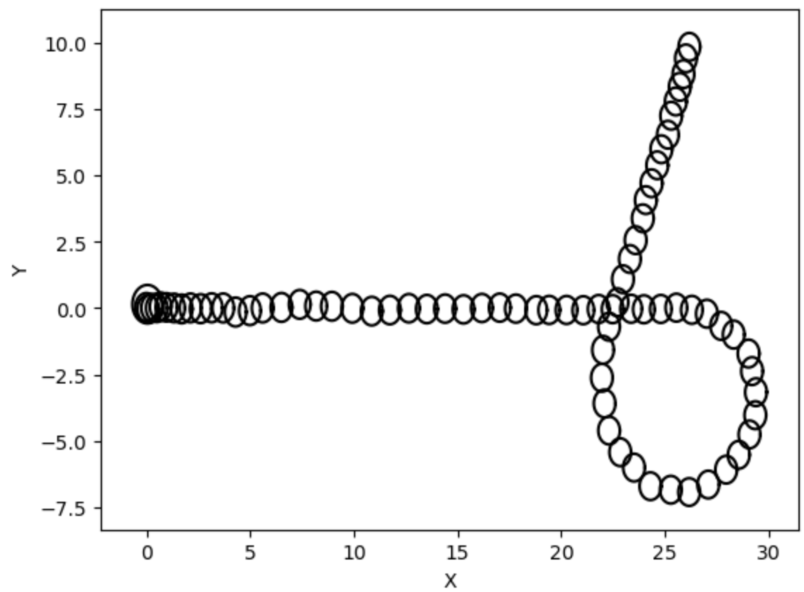

Figure 1.1 X and Y Covariance values

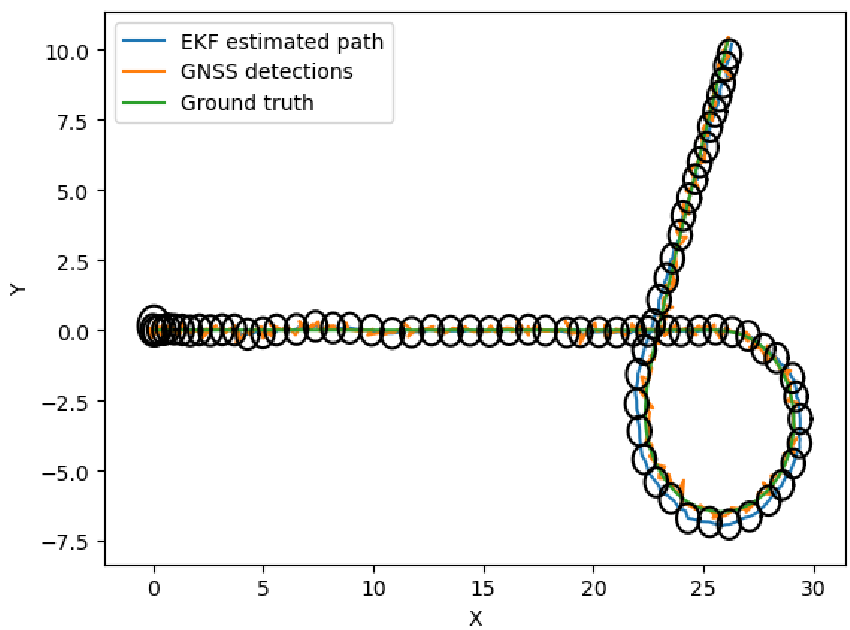

Figure 1.2 X and Y position

2. Mean square error

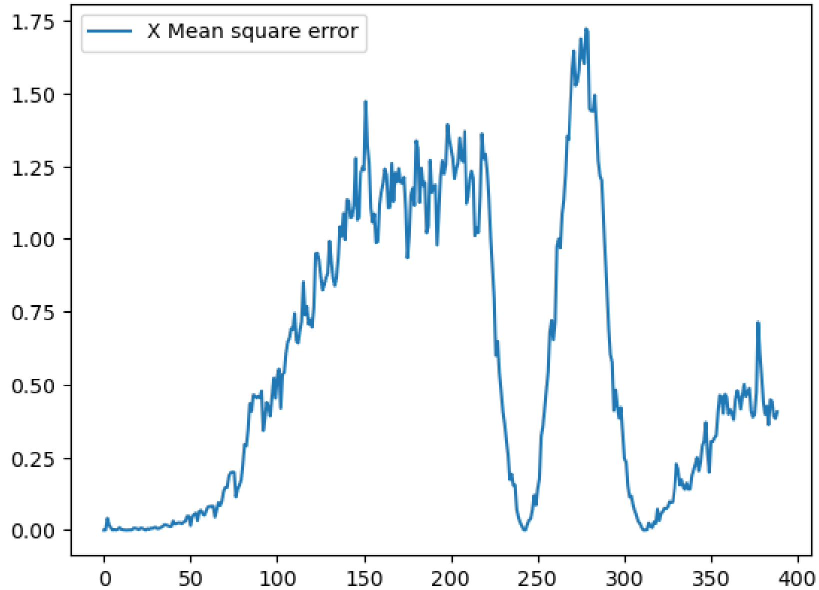

Figure 2.1 X position mean square error

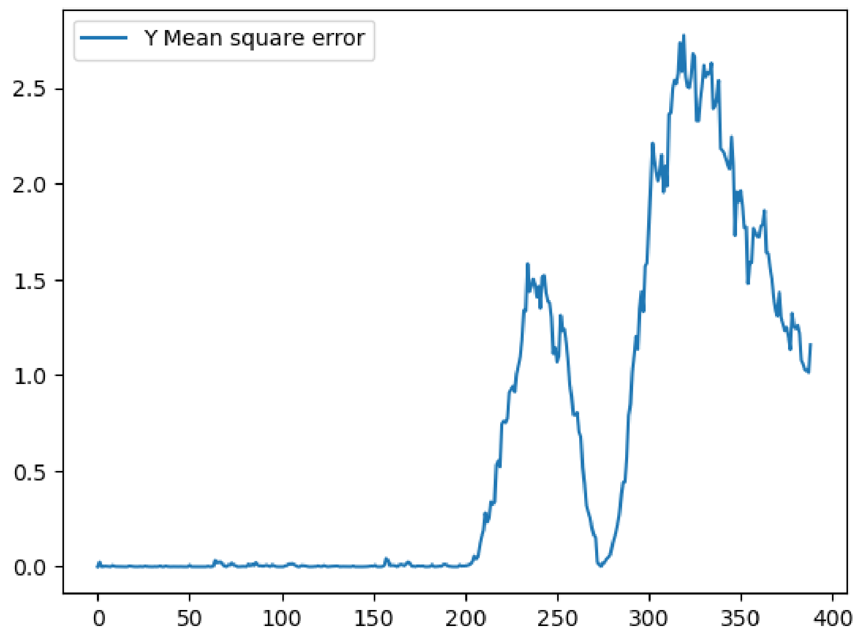

Figure 2.2 Y position mean square error

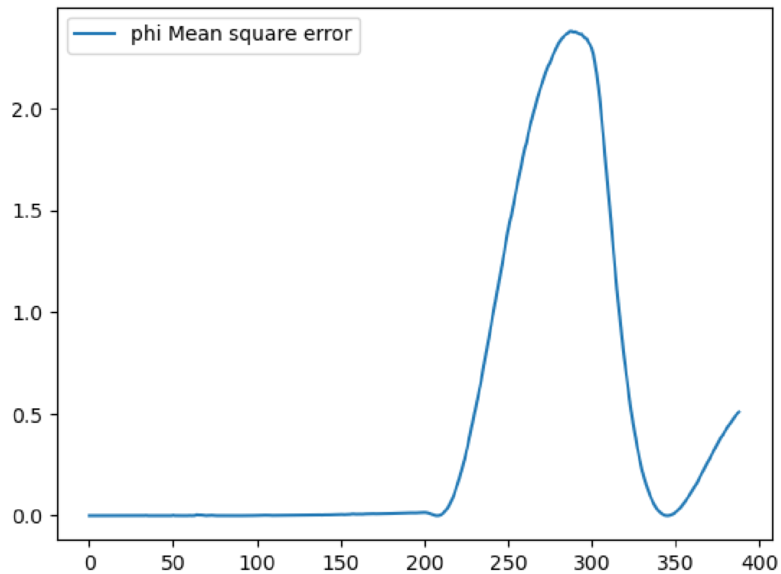

Figure 2.3 phi mean square error

3. Path

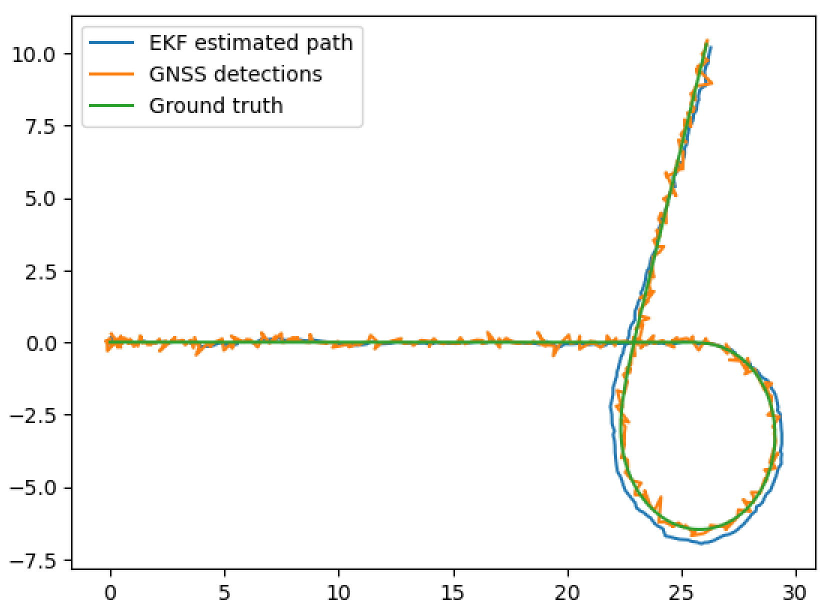

Figure 3.1 The ground truth, the GNSS detections, and the estimated path

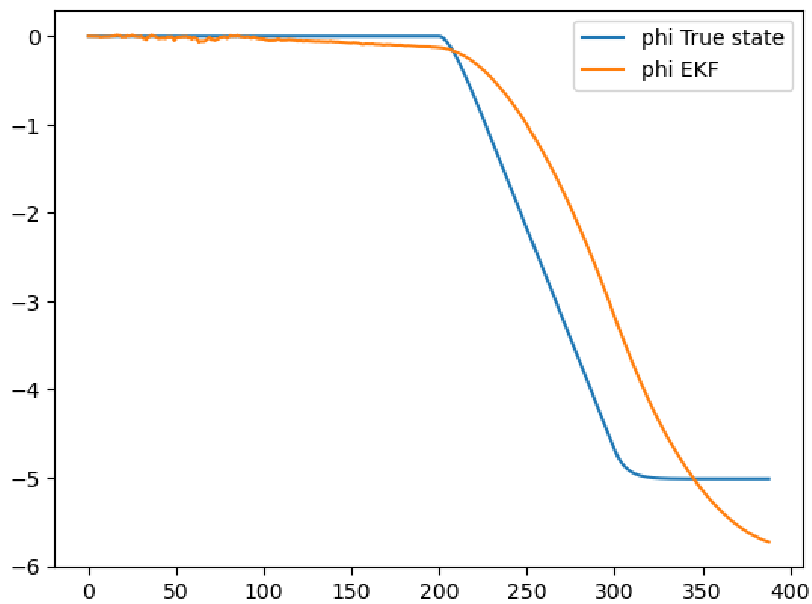

Figure 3.2 The ground truth phi and the estimated phi