Bakari Hassan - Problem Set 3 - ECE 209AS

November 4, 2018

Preliminaries

1. Github Link: `https://github.com/bakastronaut/ECE_209AS_PSETS/tree/master/Homework3`
2. No collaboration involved with my submission. I did help Solomon/Jay.
3. I referred to a paper to write the UKF code (see references)
4. All contributions were mine
5. 100 % - Bakari Hassan

Mathematical Setup

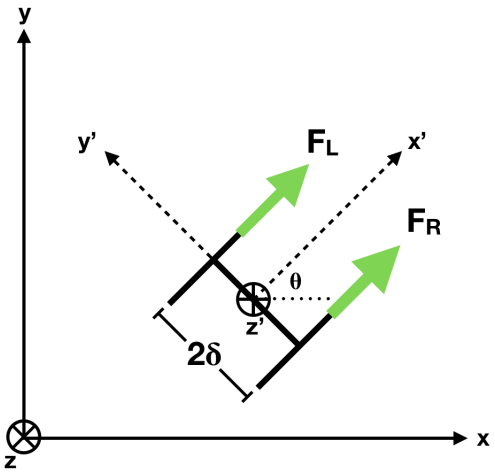
Model Definition

List of assumptions:

- No friction
- Inertial reference frames[1]

The problem was setup using two reference frames: 1) Body-fixed frame 2) Earth- (World-) fixed frame. The body-fixed frame is indicated by the primed vectors. The x'-axis is forward-facing, the y'-axis extends from the left side of the vehicle, and the z'-axis completes the orthogonal system. The Earth-fixed frame is defined with the z-axis pointing downwards as well.

[1] I was planning to relax this assumption, but did not have time.



Using Newton's second law for the body frame (primed variables), forces are only applied along the x-axis. The equation of motion (EOM) becomes:

$$\sum F_x = m\ddot{x}' = F_{left} + F_{right} \Longrightarrow \ddot{x}' = \frac{1}{m_c}\left(m_w\ddot{\phi}_L + m_w\ddot{\phi}_R\right)$$

The dynamics equations are discretely summed once (the integral analog) to get the body frame's velocity along x', which is then transformed to the world frame using a 2D rotation matrix. Those equations are then integrated to get x position as a function of the two wheel angular velocities:

$$x_{t+1} = x_t + r\frac{m_w}{m_c}\left(\dot{\phi_{L_t}} + \dot{\phi_{R_t}}\right)\Delta t \cdot cos\theta_t \ (1)$$
$$y_{t+1} = y_t + r\frac{m_w}{m_c}\left(\dot{\phi_{L_t}} + \dot{\phi_{R_t}}\right)\Delta t \cdot sin\theta_t \ (2)$$

Then, Newton's second law is applied for torques about the body z axis:

$$\sum \tau_z = I_z\ddot{\theta} = \tau_L - \tau_R \ (3)$$

where I_z is the rotational inertia about the z-axis, which is calculated assuming the car is a cylinder rotating about its primary axis:

$$I_z = \frac{1}{12}m_c\left(2\delta\right)^2 \ (4)$$

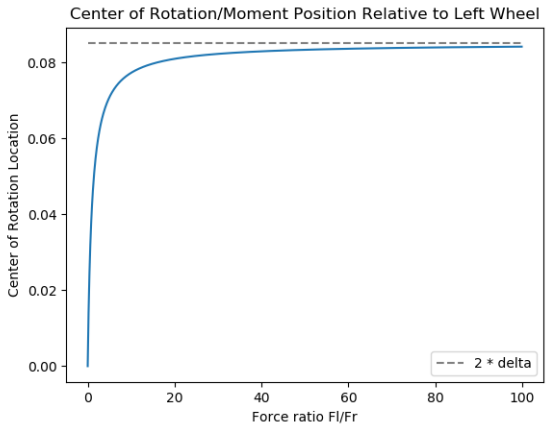
Before summing the torques, the center of torque must be found as a function of the torques produced by the individual wheels. This is done by setting the torques equal to each other and solving for the moment arm of the left motor (γ) under the constraint that the sum of the moment arms is equal to the width of the car:

$$\tau_L = \tau_R \Longrightarrow \gamma F_L = \left(2\delta - \gamma\right)F_R \ (5)$$

When gamma is solved for, the following expression of the location of center of torque relative to the left wheel is obtained:

$$\gamma = 2\delta\left(1 - \frac{1}{\frac{F_L}{F_R}+1}\right) \quad (6)$$

When plotted, it verifies that the center of rotation remains within the body of the robot, which is valid since we will let the translation equations take care of the translational component when the center of mass is not under pure rotation:



When substituted into the EOM, an expression for angular acceleration about the z-axis follows from (3):

$$\ddot{\theta} = \frac{1}{I_z}\left(\gamma F_L + \gamma F_R - 2\delta F_R\right) \quad (7)$$

When summed twice with respect to time, the state as a function of control inputs is obtained:

$$\theta_{t+1} = \theta_t + r\frac{m_w}{I_z}\left[\gamma\dot{\phi_{L_t}} + (\gamma - 2\delta)\,\dot{\phi_{R_t}}\right]\Delta t \quad (8)$$

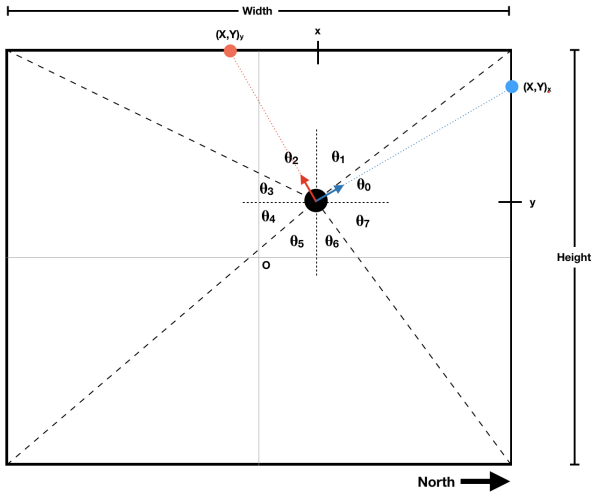
Now, consider the general expression for a dynamical system:

$$x_{t+1} = f\left(x_t, u_t\right) \quad (9)$$
$$y_{t+1} = f\left(x_{t+1}, u_t\right) \quad (10)$$

The first expression (9) in matrix form is now:

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_{t+1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_t + rm_w\Delta t\begin{bmatrix} \frac{1}{m_c}cos\theta_t & \frac{1}{m_c}cos\theta_t \\ \frac{1}{m_c}cos\theta_t & \frac{1}{m_c}cos\theta_t \\ \gamma\frac{1}{I_z} & (\gamma - 2\delta)\frac{1}{I_z} \end{bmatrix}\begin{bmatrix} \dot{\phi}_L \\ \dot{\phi}_R \end{bmatrix}_t \quad (11)$$

The output equation (10) is piecewise nonlinear and cannot be expressed in matrix format. The function is setup as follows:



where:

$$\begin{aligned}
\theta_0 &= \tan^{-1}\left(\frac{\frac{h}{2}+y}{\frac{w}{2}-x}\right) & \theta_7 &= \tan^{-1}\left(\frac{\frac{h}{2}-y}{\frac{w}{2}-x}\right) \\
\theta_1 &= \tan^{-1}\left(\frac{\frac{w}{2}-x}{\frac{h}{2}-y}\right) & \theta_2 &= \tan^{-1}\left(\frac{\frac{w}{2}+x}{\frac{h}{2}-y}\right) \\
\theta_3 &= \tan^{-1}\left(\frac{\frac{h}{2}-y}{\frac{w}{2}+x}\right) & \theta_4 &= \tan^{-1}\left(\frac{\frac{h}{2}+y}{\frac{w}{2}+x}\right) \\
\theta_5 &= \tan^{-1}\left(\frac{\frac{w}{2}+x}{\frac{h}{2}+y}\right) & \theta_6 &= \tan^{-1}\left(\frac{\frac{w}{2}-x}{\frac{h}{2}+y}\right)
\end{aligned} \quad (12)$$

The point of intersection with the walls is determined by finding the intersection of the line emanating from the robot's CM along the body axis unit vector and the equation of the wall. The equation of the line along the body x-axis (f_x) and y-axis (f_y):

$$f_y = x\tan\theta_0 - x_0\tan\theta_0 + y_o \quad (13)$$
$$f_x = x\tan\left(\theta_0 - \tfrac{\pi}{2}\right) - x_0\tan\left(\theta_0 - \tfrac{\pi}{2}\right) + y_o \quad (14)$$

where (x_0, y_0) is the robot's position.

Then, logic is used to determine which of the four angular regions the individual axes are in, as well as handling singularities in the tangent function:

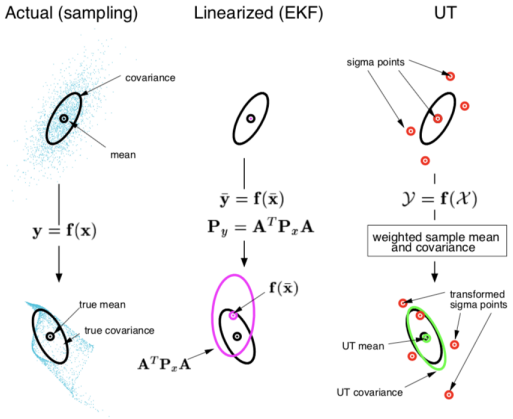
```
    tan_theta = np.tan(theta)
    if tan_theta < 0:
        tan_theta += 2*np.pi
    if theta_wall_R[0] < theta or theta <= theta_wall_R[1]:

        X = g_r[0]          # Constrain X because we know it is along the wall.

        # If axis points along y axis, Y = y so (Y-y)^2 = 0
        if tan_theta > thresh:
            Y = y
        else:
            Y = X*tan_theta - x*tan_theta + y

    elif theta_wall_T[0] < theta <= theta_wall_T[1]:

        Y = g_t[1]          # Constrain Y because we know it is along the wall.

        # If axis points along y axis, X = x so (X-x)^2 = 0
        if tan_theta > thresh:
            X = x

        # If tangent is almost zero, Y = y and X is assumed to be at either L/R
        # wall depending on theta
        elif tan_theta < 1/thresh:
            Y = y
            if theta < np.pi/2:
                X = d
            else:
                X = c
        else:
            X = (1/tan_theta) * (Y + x*tan_theta - y)

    elif theta_wall_L[0] < theta <= theta_wall_L[1]:

        X = g_l[0]
        if tan_theta > thresh:
            Y = y
        else:
            Y = X*tan_theta - x*tan_theta + y

    elif theta_wall_B[0] < theta <= theta_wall_B[1]:
        Y = g_b[1]

        # If axis points along y axis, X = x so (X-x)^2 = 0
        if tan_theta > thresh:
            X = x

        # If tangent is almost zero, Y = y and X is assumed to be at either L/R
        # wall depending on theta
        elif tan_theta < 1/thresh:
            Y = y
            if theta < 3*np.pi/2:
                X = c
            else:
                X = d
        else:
            X = (1/tan_theta) * (Y + x*tan_theta - y)
```

The distance is then found by taking the ℓ-2 norm of the differences between the perimeter coordinates (X, Y) and the robot coordinates (x_0, y_0):

$$\text{dist} = \sqrt{(X - x_0)^2 + (Y - y_0)^2}$$

Unscented Kalman Filter

The unscented Kalman filter (UKF) was chosen for this project due to an incremental increase in complexity relative to the extended Kalman Filter (EKF), but its significant improvement in accuracy over the EKF. The difference is that instead of linearizing the dynamical system, which can cause large errors for large rates of change, it tracks a few points (called sigma vectors whose values depend on scaling parameters and the state covariance) by sending them through the nonlinear transformation, along with the noise estimates, and then calculating their mean and covariance using a weighted sum to get the expected mean and covariance of the state. The following 3 images were taken from [1]:



The equations have been inserted from in the figure below. The first image is the general algorithm and the second is how the sigma vector and weights are calculated:

Initialize with:

$$\begin{aligned}\hat{\mathbf{x}}_0 &= E[\mathbf{x}_0] \\ \mathbf{P}_0 &= E[(\mathbf{x}_0 - \hat{\mathbf{x}}_0)(\mathbf{x}_0 - \hat{\mathbf{x}}_0)^T] \\ \hat{\mathbf{x}}_0^a &= E[\mathbf{x}^a] = [\hat{\mathbf{x}}_0^T \; \mathbf{0} \; \mathbf{0}]^T\end{aligned}$$

$$\mathbf{P}_0^a = E[(\mathbf{x}_0^a - \hat{\mathbf{x}}_0^a)(\mathbf{x}_0^a - \hat{\mathbf{x}}_0^a)^T] = \begin{bmatrix} \mathbf{P}_0 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{P_v} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{P_n} \end{bmatrix}$$

For $k \in \{1, \ldots, \infty\}$,

Calculate sigma points:

$$\boldsymbol{\mathcal{X}}_{k-1}^a = \left[\hat{\mathbf{x}}_{k-1}^a \;\; \hat{\mathbf{x}}_{k-1}^a \pm \sqrt{(L+\lambda)\mathbf{P}_{k-1}^a}\right]$$

Time update:

$$\begin{aligned}\boldsymbol{\mathcal{X}}_{k|k-1}^x &= \mathbf{F}[\boldsymbol{\mathcal{X}}_{k-1}^x, \boldsymbol{\mathcal{X}}_{k-1}^v] \\ \hat{\mathbf{x}}_k^- &= \sum_{i=0}^{2L} W_i^{(m)} \mathcal{X}_{i,k|k-1}^x \\ \mathbf{P}_k^- &= \sum_{i=0}^{2L} W_i^{(c)} [\mathcal{X}_{i,k|k-1}^x - \hat{\mathbf{x}}_k^-][\mathcal{X}_{i,k|k-1}^x - \hat{\mathbf{x}}_k^-]^T \\ \boldsymbol{\mathcal{Y}}_{k|k-1} &= \mathbf{H}[\boldsymbol{\mathcal{X}}_{k|k-1}^x, \boldsymbol{\mathcal{X}}_{k-1}^n] \\ \hat{\mathbf{y}}_k^- &= \sum_{i=0}^{2L} W_i^{(m)} \mathcal{Y}_{i,k|k-1}\end{aligned}$$

Measurement update equations:

$$\begin{aligned}\mathbf{P}_{\tilde{\mathbf{y}}_k \tilde{\mathbf{y}}_k} &= \sum_{i=0}^{2L} W_i^{(c)} [\mathcal{Y}_{i,k|k-1} - \hat{\mathbf{y}}_k^-][\mathcal{Y}_{i,k|k-1} - \hat{\mathbf{y}}_k^-]^T \\ \mathbf{P}_{\mathbf{x}_k \mathbf{y}_k} &= \sum_{i=0}^{2L} W_i^{(c)} [\mathcal{X}_{i,k|k-1} - \hat{\mathbf{x}}_k^-][\mathcal{Y}_{i,k|k-1} - \hat{\mathbf{y}}_k^-]^T \\ \mathcal{K} &= \mathbf{P}_{\mathbf{x}_k \mathbf{y}_k} \mathbf{P}_{\tilde{\mathbf{y}}_k \tilde{\mathbf{y}}_k}^{-1} \\ \hat{\mathbf{x}}_k &= \hat{\mathbf{x}}_k^- + \mathcal{K}(\mathbf{y}_k - \hat{\mathbf{y}}_k^-) \\ \mathbf{P}_k &= \mathbf{P}_k^- - \mathcal{K} \mathbf{P}_{\tilde{\mathbf{y}}_k \tilde{\mathbf{y}}_k} \mathcal{K}^T\end{aligned}$$

where, $\mathbf{x}^a = [\mathbf{x}^T \; \mathbf{v}^T \; \mathbf{n}^T]^T$, $\boldsymbol{\mathcal{X}}^a = [(\boldsymbol{\mathcal{X}}^x)^T \; (\boldsymbol{\mathcal{X}}^v)^T \; (\boldsymbol{\mathcal{X}}^n)^T]^T$, λ=composite scaling parameter, L=dimension of augmented state, $\mathbf{P_v}$=process noise cov., $\mathbf{P_v}$=measurement noise cov., W_i=weights

$$\begin{aligned}\mathcal{X}_0 &= \bar{\mathbf{x}} \\ \mathcal{X}_i &= \bar{\mathbf{x}} + \left(\sqrt{(L+\lambda)\mathbf{P}_{\mathbf{x}}}\right)_i \qquad i = 1, \ldots, L \\ \mathcal{X}_i &= \bar{\mathbf{x}} - \left(\sqrt{(L+\lambda)\mathbf{P}_{\mathbf{x}}}\right)_{i-L} \qquad i = L+1, \ldots, 2L \\ W_0^{(m)} &= \lambda/(L+\lambda) \\ W_0^{(c)} &= \lambda/(L+\lambda) + (1 - \alpha^2 + \beta) \\ W_i^{(m)} &= W_i^{(c)} = 1/\{2(L+\lambda)\} \qquad i = 1, \ldots, 2L\end{aligned}$$

Evaluation

Demonstration

Five trials were run to test the effectiveness of the UKF:

1. Initial state uncertainty (no environment or sensor noise)
2. Finite enviornment covariance (no initital state uncertainty or sensor noise)
3. Finite sensor covariance (no initial state uncertainty or enviornment noise)
4. Initial state uncertainty with sensor noise (no environment noise)
5. Initial state uncertainty with sensor and environment noise

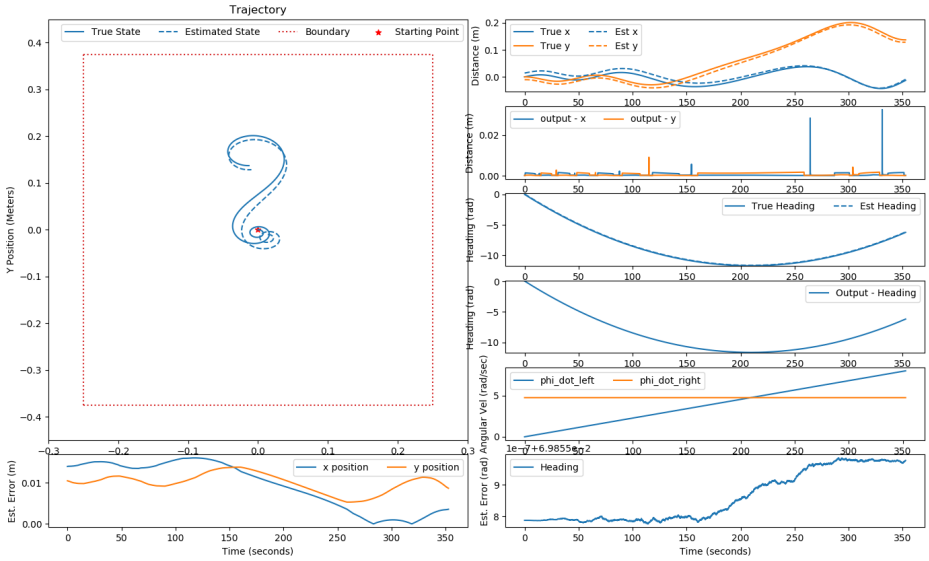
The control inputs for each scenario are identical. The right wheel is kept at a constant rotational velocity of 5 radians per second with the left wheel velocity ramped from 0 to 4.75 radians per second. This creates a nice spiral-out, spiral-in trajectory.

Case 1: Initial State Uncertainty

The first scenario is one in which there is uncertainty in the initial state. Overall, the Kalman filter did a good job estimating the state, with the estimation error converging over the duration from the beginning to end of the trajectory. This is apparent in the trajectory plot (top left) when comparing the initial offset to the final offset. The following covariance matrix was used:

$$\sum = \begin{matrix} 0.001 & 0 & 0 \\ 0 & 0.001 & 0 \\ 0 & 0 & 0.001 \end{matrix} \quad m^2$$

It can also be seen in the absolute error plot below the trajectory plot with the final errors beginning above 0.01 and both ending below. Finally, the plot at the top right shows the offset (both positive and negative) through time. At the end, the estimated x position is almost indiscernible from the true x position. Although the heading increased, it was by a negligible amount, on the order of a micro radian.

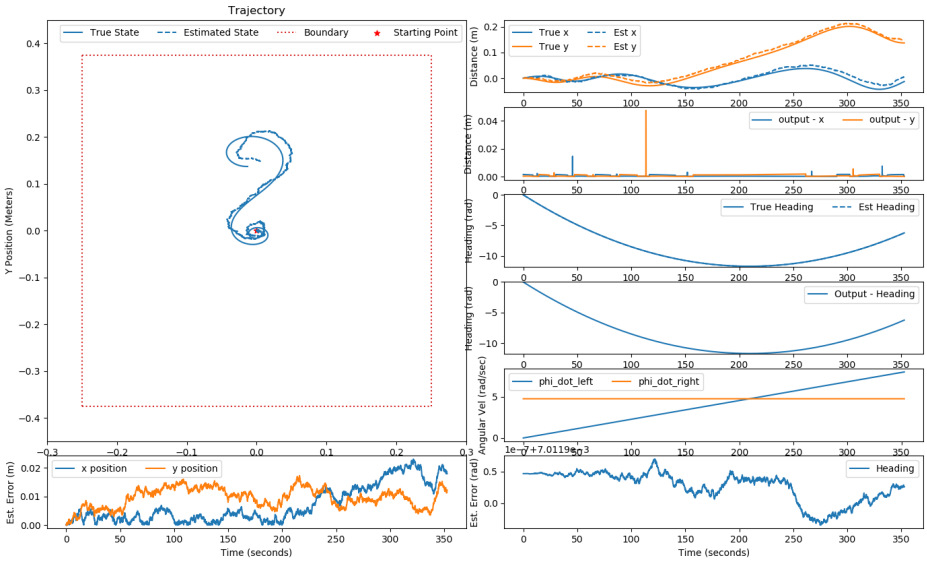


Case 2: Environment Noise

The second scenario is one in which there is environmental noise, meaning the robot's actions result in uncertain actions such as slipping. Overall, the Kalman filter performed similarly to the case with state uncertainty, with the estimation error converging over the duration from the beginning to end of the trajectory. This is apparent in the trajectory plot (top left) when comparing the initial offset to the final offset. The following covariance matrix was used:

$$\sum = 1 \times 10^{-5} \begin{matrix} 0.0025 & 0 & 0 \\ 0 & 0.0025 & 0 \\ 0 & 0 & 0.0000 \end{matrix} \; m^2$$

However, different from the previous trial, the error increases from 0 to values similar to the previous trial. The initial low value is expected because there was no uncertainty in the initial state. But the fact that they ended at the same position highlights the divergence. Finally, the plot at the top right shows the offset (both positive and negative) through time. In this case, the y error was smaller than the x error, likely due to the trajectory including more y translation than x translation. This can affect the measurements used to gain confidence in estimations. In this case, the heading error essentially remained constant on the scale of sub-micro radian errors.

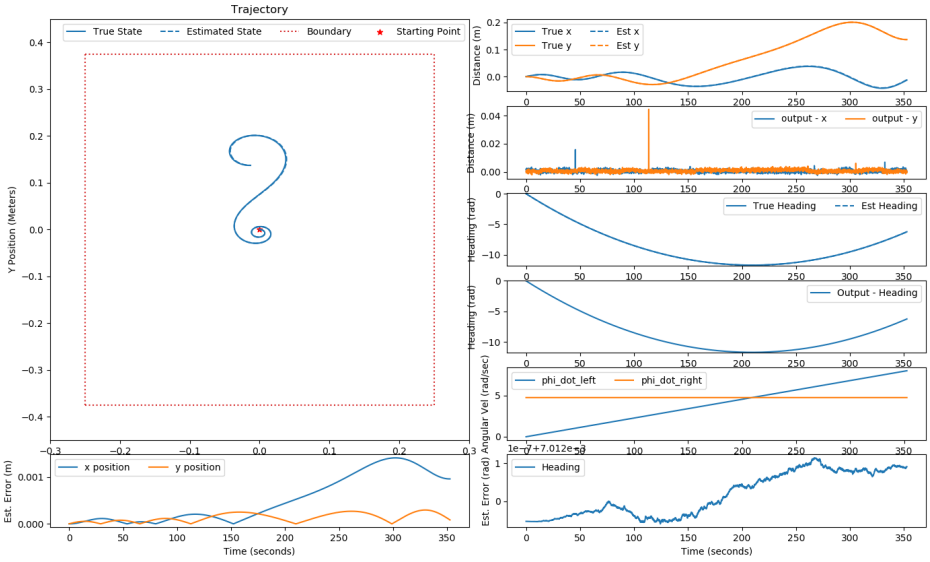


Case 3: Sensor Noise

This scenario is one in which there is sensor noise. This case had the best performance overall. The true and estimated trajectories are almost identical, with all estimation errors less than 1 mm in magnitude outside of a brief period between 350 and 325 seconds. The following covariance matrix was used:

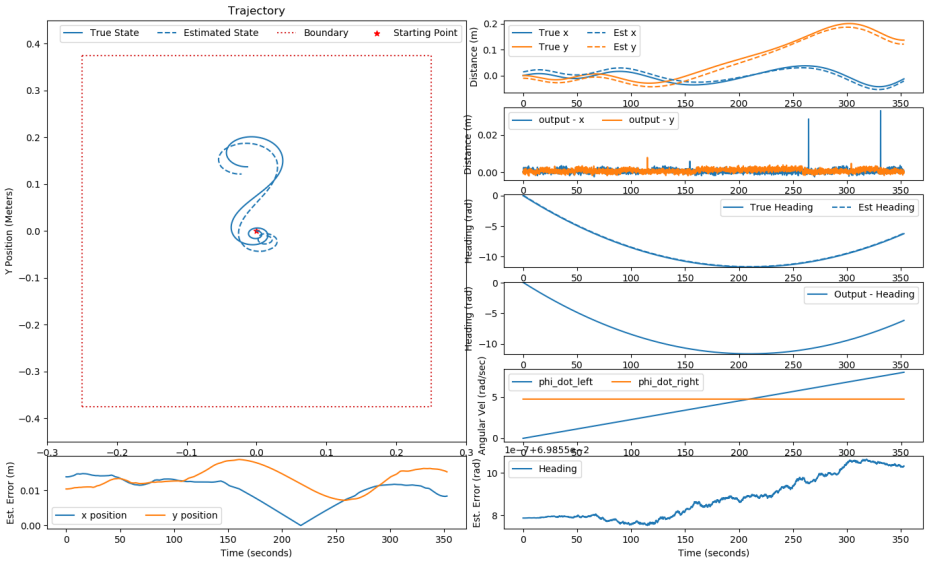
$$\sum = 1 \times 10^{-4} \begin{matrix} 0.006 & 0 & 0 \\ 0 & 0.006 & 0 \\ 0 & 0 & 1 \times 10^{-6} \end{matrix} m^2$$

Sensor noise did not appear to have a large effect on the state estimator's performance. Additionally, the estimated trajectory is very smooth, meaning the sensor noise did not affect the mapping from control inputs to physical actions. This makes sense because the noise term for sensors is not in the state update equation. They should only affect the confidence in the current state.



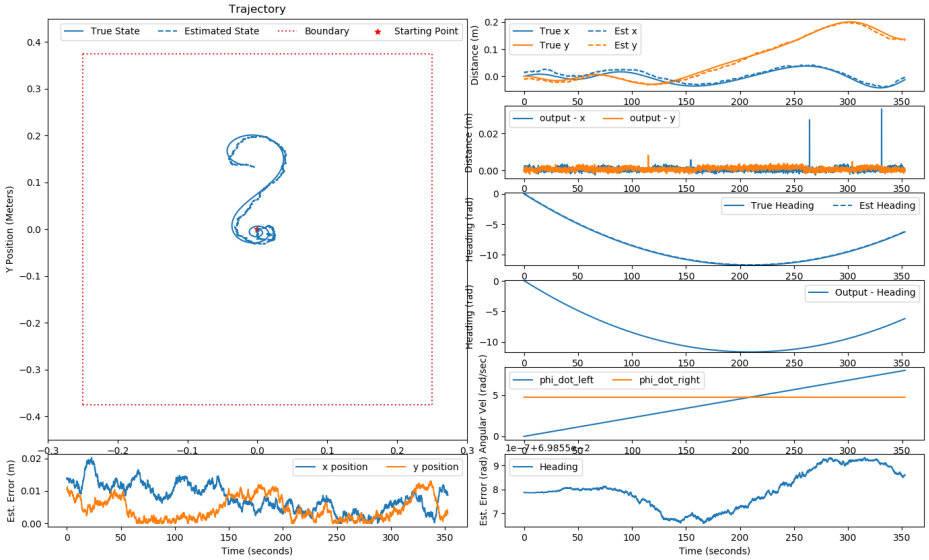
Case 4: State Uncertainty with Sensor Noise

Scenario four was run as a follow-on to scenario three to see how the unscented Kalman filter performed with both initial state uncertainty and sensor noise. As I expected, the sensor noise made the state uncertainty higher, resulting in larger absolute errors for translation state estimation. The same covariance matrices were used for this case that were used for the two cases above for the individual scenarios involving initial state uncertainty and sensor noise.



Case 5: State Uncertainty with Sensor and Environment Noise

The fifth scenario combined all noise and uncertainty sources. The unscented Kalman filter did very well in this situation. It outperformed all cases other than the third case that only contained sensor noise. This was unexpected, as the combined case was expected to be the worst-performing case.



Conclusion

Overall, the UKF did a really good job on the state estimation. It managed to keep all estimation errors below 0.02 meters (20 mm) for a maximum y position offset of 200 mm, which is 10 percent of the traveled distance. This would be suitable as long as no precision position knowledge is needed for an appliaction.

One way to improve this filter could be to use machine learning to learn the noise characteristics of the sensors. If learned well, the filter could predict the noise for each time step more effectively than a pure statistical approach based on mean and covariance.

References

[1] E. A. Wan, R. Van der Merwe, "The unscented Kalman filter for nonlinear estimation", Proc. Symp. Adaptive Syst. Signal Process. Commun. Contr., Oct. 2000.