EE209AS Problem set 3

Jay Jackman

Preliminaries

0(a): https://github.com/JayJackman/UKF-Robot

0(b): Bakari Hassan

0(c): https://www.seas.harvard.edu/courses/cs281/papers/unscented.pdf, https://www.sciencedirect.com/science/article/pii/S0959152407001655

0(d): Bakari helped clarify some things about the Unscented Kalman Filter for me

0(e): 100% of the code and writeup was written by Jay Jackman, Bakari only provided some helpful clarifications on the algorithm

Mathematical Setup

2(a) Define the system model:

The system model consists of a rigid axle, with two wheels on each end. Each wheel can be operated individually, making this a differential drive robot. The robot's state is completely defined by 3 numbers:

x: the robot's position along the x axis
y: the robot's position along the y axis
θ: the robot's angle, measured clockwise from north

Note that the position is measured as the center of the axle of the robot. Although a body for the robot has been drawn in for the simulation, it is not currently modeled physically.

The robot can take two inputs from each of its FS90R servos:

ω_L: the angular velocity of the left wheel, in Revolutions Per Minute (RPM)
ω_R: the angular velocity of the right wheel, in RPM

The maximum rate of the servos is defined on the datasheet to be 130 RPM, which means:

$MAX\_RPM \triangleq 130$
$-MAX_{RPM} \leq \omega_i \leq MAX_{RPM}, i \in \{L, R\}$

The robot can make the following observations using the two VL53L0X laser sensors and the MPU-9250 IMU sensor, respectively:

D_{front}: the distance from the robot to the nearest wall, looking forward
D_{right}: the distance from the robot to the nearest wall, looking to the right
$\hat{\theta}$: the measured angle of the robot, clockwise with respect to north

The noise parameters were found online as follows:

$\sigma_L = 0.5$ represents the error in the resultant $\widetilde{\omega}_L$ for a given ω_L
$\sigma_R = 0.5$ represents the error in the resultant $\widetilde{\omega}_R$ for a given ω_R
$\sigma_d = 0.07$ represents the error in the returned distance for the distance sensor
$\sigma_\theta = 0.01$ represents the error in the magnetometer reading.

System Dynamics:

In the following, refer to Figure 1 to see visual representations of the variables.

The system dynamics were derived as follows: given an initial state $S_1 = (x_1, y_1, \theta_1)$, an input (ω_L, ω_R), and a timestep Δt, we need to determine the final state $S_2 = (x_2, y_2, \theta_2)$. For ease of simulations, Δt is assumed to be 10 ms and can be largely ignored.

I will convert the inputs (ω_L, ω_R), which are in $\frac{revolutions}{minute}$, into (V_L, V_R), which will be in $\frac{10mm}{ms}$, using the following formula: $V = \omega * \frac{1}{6000} * 2\pi r$, where r is the radius of the wheel in mm.

The first observation to make when deriving the mechanics of the system is that any combination of inputs will rotate S_1 about some point, which I will call $C = (x_c, y_c)$. The point C will be a distance away from the center of the robot, which I will call R.

In one period T, the robot's center will travel $2\pi R$ units. I will define the angular velocity of the robot as $\Omega = \frac{2\pi}{T}$. Due to the rigid nature of the robot, we know that in one period T, the left wheel will travel $2\pi\left(R - \frac{l}{2}\right)$ and the right wheel will travel $2\pi\left(R + \frac{l}{2}\right)$, where l is the length of the axle in mm. We can write this as:

$$V_L = \frac{2\pi\left(R-\left(\frac{l}{2}\right)\right)}{T} \text{ and } V_R = \frac{2\pi\left(R+\left(\frac{l}{2}\right)\right)}{T}$$

which can be simplified to:

$$V_L = \Omega\left(R - \left(\frac{l}{2}\right)\right) \text{ and } V_R = \Omega\left(R + \left(\frac{l}{2}\right)\right)$$

We can now solve for the R and Ω:

Solve for Ω in the left equation:

$$\Omega = \frac{V_L}{R-\left(\frac{l}{2}\right)}$$

Substitute:

$$V_R = \frac{V_L}{R-\left(\frac{l}{2}\right)}\left(R + \left(\frac{l}{2}\right)\right)$$

Solve for R:

$$R = \frac{l}{2} * \frac{V_L + V_R}{V_R - V_L}$$

Note: If $V_R = V_L$, then R is undefined. This corresponds to "rotating about a point infinitely far away," which means that the robot is going straight.

Substitute back to solve for Ω:

$$\Omega = \frac{V_R - V_L}{l}$$

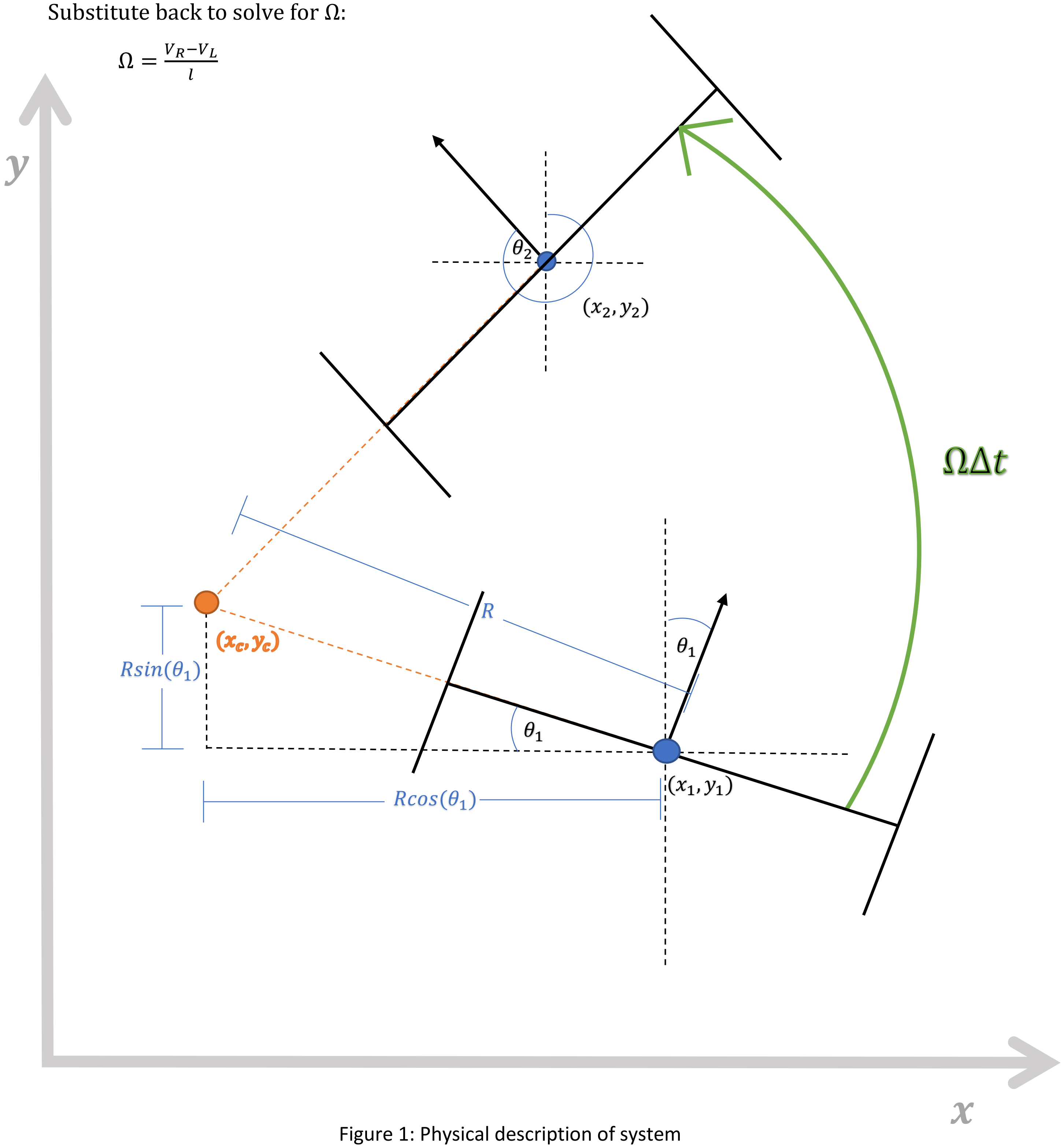

Figure 1: Physical description of system

Now, we can calculate the point C:

$$x_c = x_1 - Rcos(\theta_1)$$
$$y_c = y_1 + Rsin(\theta_1)$$

It is easy to calculate (x_2, y_2), which is the point (x_1, y_1) rotated $\Omega\Delta t\ radians$ about (x_c, y_c), using a rotation matrix with a translation:

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \cos(\Omega\Delta \mathrm{t}) & -\sin(\Omega\Delta \mathrm{t}) \\ \sin(\Omega\Delta \mathrm{t}) & \cos(\Omega\Delta \mathrm{t}) \end{bmatrix} \begin{bmatrix} x_1 - x_c \\ y_1 - y_c \end{bmatrix} + \begin{bmatrix} x_c \\ y_c \end{bmatrix}$$

It is also simple to calculate θ_2:

$$\theta_2 = (\theta_1 - \Omega\Delta t)\ mod\ 2\pi$$

To put it all into 3 decoupled update equations:

$$x_2 = x_1 + R * (\cos(\Omega\Delta t)\cos(\theta_1) + \sin(\Omega\Delta t)\sin(\theta_1) - \cos(\theta_1))$$

$$y_2 = y_1 + R * (\sin(\Omega\Delta t)\cos(\theta_1) - \cos(\Omega\Delta t)\sin(\theta_1) + \sin(\theta_1))$$

$$\theta_2 = (\theta_1 - \Omega\Delta t)\ mod\ 2\pi$$

Noise can be added to the system by replacing ω_L and ω_R with $\widetilde{\omega}_L$ and $\widetilde{\omega}_R$:

$$\widetilde{\omega}_L = \omega + v_L * MAX\_RPM \quad v_L \sim \eta(0, \sigma_L)$$

$$\widetilde{\omega}_R = \omega + v_R * MAX\_RPM \quad v_R \sim \eta(0, \sigma_R)$$

Observation Calculations:

The robot makes three different observations. It measures the distance to the nearest wall directly ahead and directly to its right with the laser sensors, and it measures its global orientation with its magnetometer.

The global orientation data does not need to be calculated in my simulation, as it is exactly a measure of the state variable. All I do is add a noise term to the true orientation of the robot:

$$\tilde{\theta} = (\theta_{true} + n_\theta * 2\pi) \quad n_\theta \sim \eta(0, \sigma_\theta)$$

Refer to Figure 4 for the following laser sensor calculations:

The laser sensor readings required a bit of work to calculate in the simulation. I needed a function that would take in the position and orientation of the robot and return the distance to the nearest wall. In order to do so, I calculated the point in which the laser intersects a wall, (x_i, y_i). Then, I can simply find the distance between that point and the robot's current position (x_r, y_r):

$$distance = \sqrt{(x_i - x_r)^2 + (y_i - y_r)^2}$$

First, consider a robot that is pointing exactly up, exactly right, exactly down, or exactly left. In each case, there is one and only one wall that could be directly in front of the robot. As such, the following logic was used (Figure 2):

```
If(up):
        return $(x_r, WIDTH)$
Else if(right):
        return $(LENGTH, y_r)$
Else if(down):
        return $(x_r, 0)$
Else if(left):
        return $(0, y_r)$
```

Figure 2

If the robot is not pointing exactly in a cardinal direction, then it is pointing in one of four quadrants, with respect to the robot's position. If we treat the walls as lines extending to $\pm\infty$, then depending on the quadrant that it is pointing in, there are exactly two walls that the robot's line of sight could intersect. However, one of the intersection points will be out of bounds. I calculated the two intersection points, and then chose the intersection point that coincided with a physical wall. The logic was as follows (Figure 3):

```
If(Upper Right Quadrant):
        If(top_intersection[x] > WIDTH):
                return right_intersection
        else:
                return top_intersection
Else if(Lower Right Quadrant):
        If(bottom_intersection[x] > WIDTH):
                return right_intersection
        else:
                return  bottom_intersection
Else if(Lower Left Quadrant):
        If(bottom_intersection[x] < 0):
                return left_intersection
        else:
                return bottom_intersection
Else if(Upper Left Quadrant):
        If(top_intersection[x] < 0):
                return left_intersection
        else:
                return top_intersection
```

Figure 3

Once the distances were calculated, noise was added:

$$\widetilde{D} = D_{true} + D_{true} * n_d \quad n_d \sim \eta(0, \sigma_d)$$

In order to find the distance to the right, I used the same logic, but entered

$$\left(\theta + \frac{\pi}{2}\right) mod\ 2\pi$$

into the function.

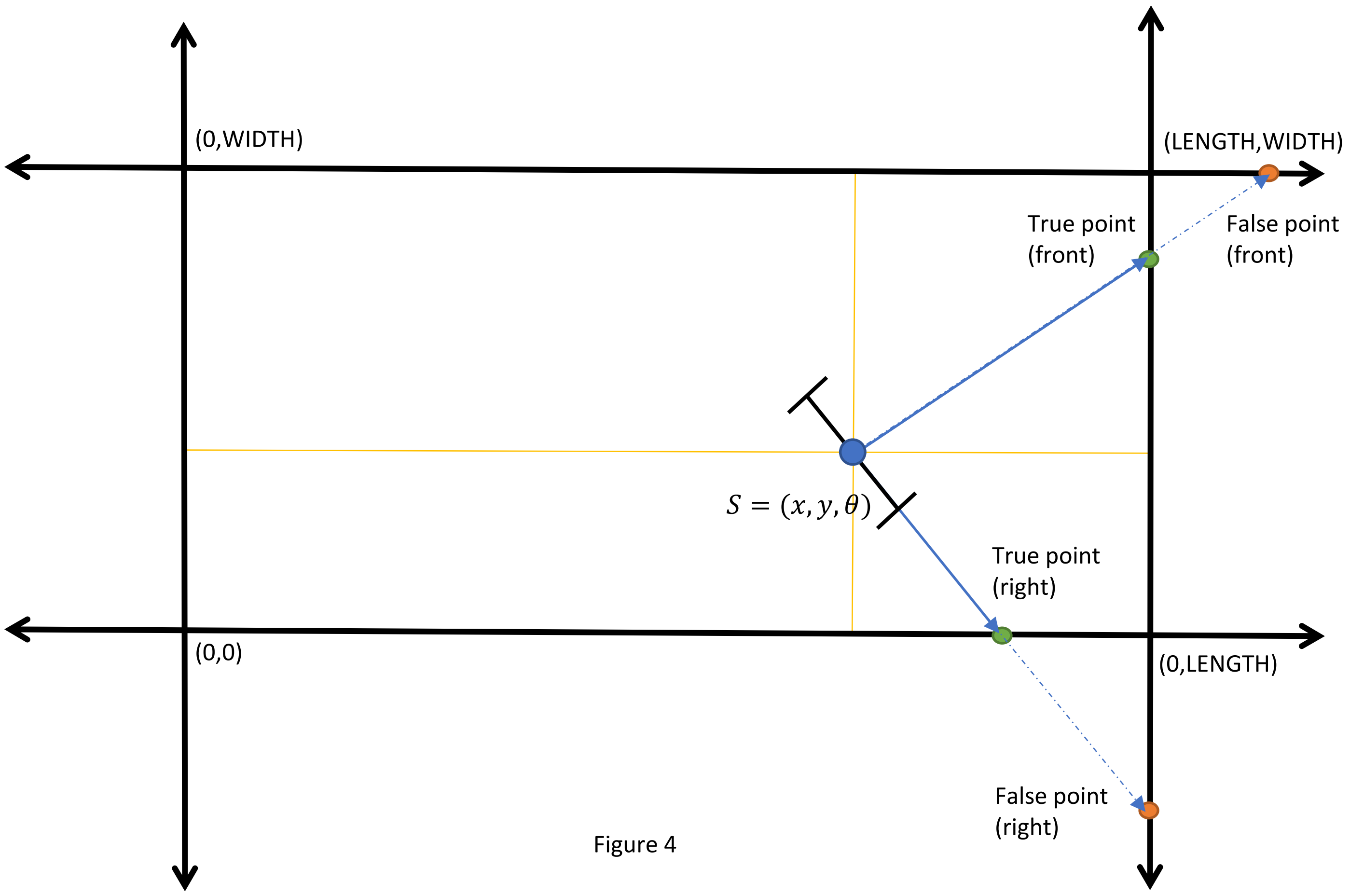

Figure 4

State Estimation: Unscented Kalman Filter:

Both the measurement equations and the state propagation equations are highly nonlinear. As such, neither a Kalman Filter or an Extended Kalman Filter will perform as well as an Unscented Kalman Filter (UKF).

The UKF belongs to a class of Sigma-Point Kalman Filters. In essence, the UKF takes the current (estimated) state, and then draws several points in its neighborhood. The specifics of how and why these points are drawn are above the scope of this report, but they can be found in [1], [2], and the papers cited within them. After these sigma points are drawn, they are then directly propagated through the state equation. The propagated sigma points are then used to fit a normal random variable to predict the new a-priori state.

Next, observations are drawn from the propagated sigma points. Again, these observations are combined to form a normal random variable to help create the Kalman innovation, which is used to update the a-posteriori state.

This algorithm is detailed in [1], and is provided here for reference:

Initialize with:

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

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

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

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:

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

Measurement update equations:

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

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 as calculated in Eqn. 15.

$$\begin{aligned}
\mathcal{X}_0 &= \bar{\mathbf{x}} & & (15) \\
\mathcal{X}_i &= \bar{\mathbf{x}} + \left(\sqrt{(L+\lambda)\mathbf{P_x}}\right)_i & i = 1, \dots, L & \\
\mathcal{X}_i &= \bar{\mathbf{x}} - \left(\sqrt{(L+\lambda)\mathbf{P_x}}\right)_{i-L} & i = L+1, \dots, 2L & \\
W_0^{(m)} &= \lambda/(L+\lambda) & & \\
W_0^{(c)} &= \lambda/(L+\lambda) + (1 - \alpha^2 + \beta) & & \\
W_i^{(m)} &= W_i^{(c)} = 1/\{2(L+\lambda)\} & i = 1, \dots, 2L &
\end{aligned}$$

I used a different weighting system, provided in [2], as I found it obtained better results

$$W_0^m = W_0^c = \lambda$$
$$W_i^m = W_i^c = \frac{1 - W_0^m}{2L}$$

$\lambda \in (0,1)$ is a parameter to be chosen. I chose 0.5 in my simulations.

Lastly, I implemented an extremely simple “smoothing” algorithm. At every step of the UKF, the robot does a “sanity check” to make sure that it’s predicted state is within the bounds of the area. If it is not, it discards that step. This allows the robot to throw away some noisy estimates that can arise from the highly nonlinear state and measurement functions.

3. Experiments

❖ **SETUP:**

I programmed in a sequence of events that would keep the robot in the boundaries, as the robot currently has no "boundary physics" built in. The sequence is robust, involving sub-sequences of quick rotations, slow rotations, forward/backward movement, and no movement.

I considered four different scenarios:

I. **Perfect knowledge of the initial state, with an accurate magnetometer**: In this experiment, the robot begins with an initial state estimate that perfectly coincides with the actual state. As such, the initialized state covariance matrix in the UKF is very small, indicating that the robot is sure of his state. The magnetometer had a standard deviation of 1% in this experiment.

II. **Perfect knowledge of the initial state, with an inaccurate magnetometer:** in this experiment, the robot again begins with a perfect initial state estimate, but the magnetometer is replaced with one that has a standard deviation of 5%.

III. **No knowledge of the initial state, with an accurate magnetometer**: In this experiment, the robot begins with a randomized initial state. Because of the unknown state, the initialized state covariance matrix is set to the identity, a relatively large value. This forces the robot to rely more on the information gained from observations

IV. **No knowledge of the initial state, with an inaccurate magnetometer**: In this experiment, the robot begins with a randomized initial state. As with experiment III, the initial covariance matrix is set to the identity.

❖ **RESULTS:**

I. The results can be summarized in Figure 5. With the initial state known and an accurate magnetometer, the estimated robot was able to start on the correct path and stay there, never straying farther than 4 mm from the actual robot.

II. The results can be summarized in Figure 6. With a noisy magnetometer, the estimated robot can get wildly thrown off course. It is able to correct itself after a while, but the path taken to get back on track is largely random. A more complex smoothing algorithm, that throws away or reduces large jumps in position, could be implemented to help in this case.

III. The results can be summarized in Figure 7. With the position unknown, the estimated robot's state is inaccurate until it converges onto the true state. With the accurate magnetometer, it is able to do this quickly.

IV. The results can be summarized in Figure 8. This experiment takes longer to converge to the true state than experiment III, due to the noisy angle measurement. Also, as with experiment II, the robot is subject to falling extremely off course. But again, it is able to eventually correct itself.

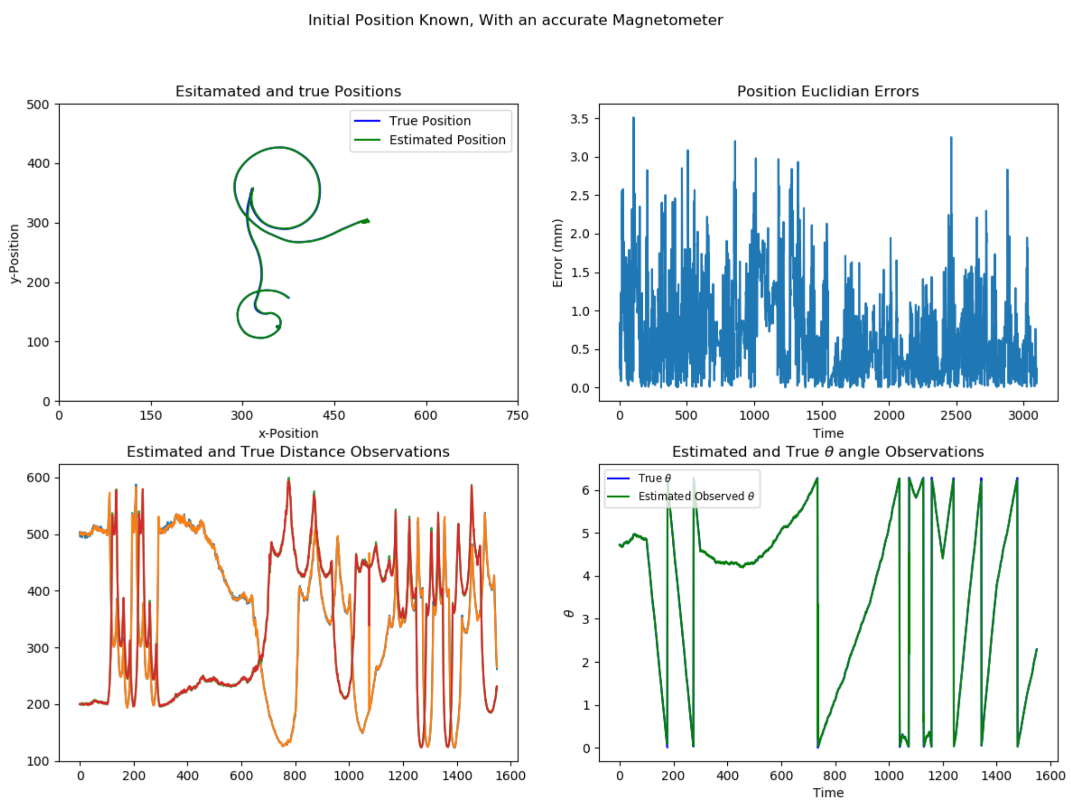

Figure 5

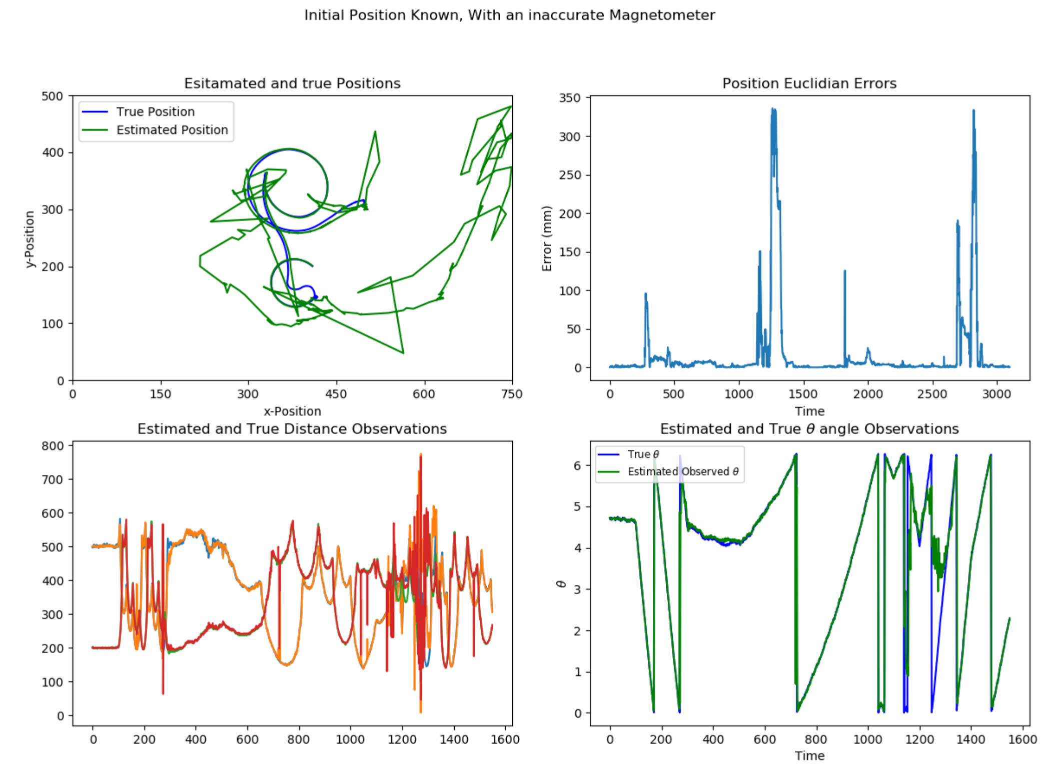

Figure 6

Initial Position Unknown, Accurate Magnetometer

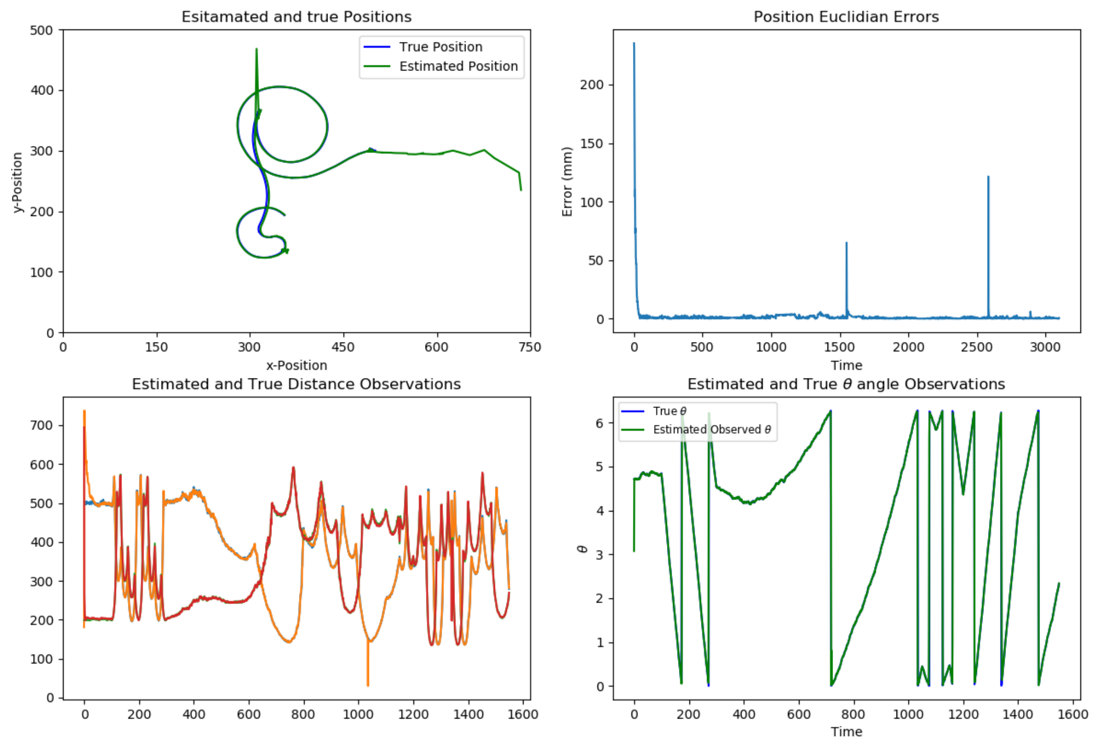

Figure 7

Initial Position Unknown, Inaccurate Magnetometer

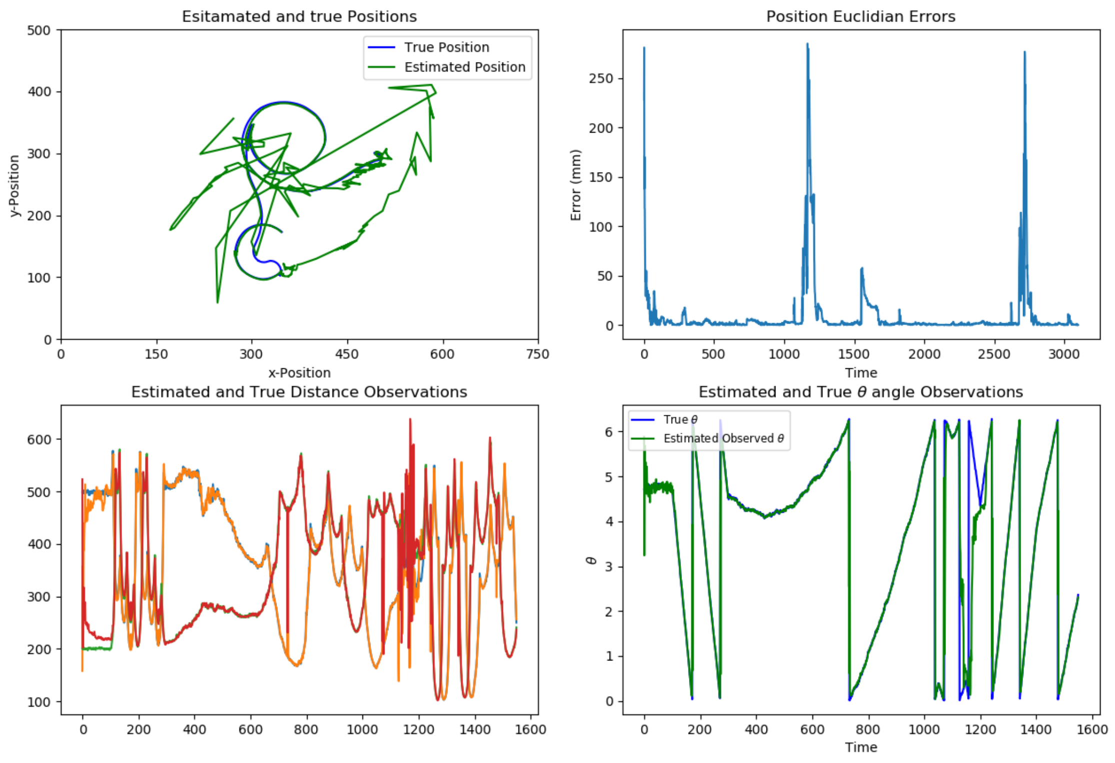

Figure 8

Conclusion:

The algorithm is very sensitive to inaccurate readings from the magnetometer. I believe this is due to the large nonlinearity in the angle measurement function when the angle jumps from $2\pi^-$ to 0^+. I believe a separate state estimator, applied to the angle, would be able to increase the overall accuracy of the filter. Also, I think it would be interesting to instead measure a transformation of the angle, such as $\cos(\theta)$. This leads to a problem where two states can have the same state measurement, but it removes the large jump discontinuity in the measurement function.

With an accurate magnetometer, my robot can very quickly converge to the true state, even when it is unsure of its location (as demonstrated by experiment III). This also applies to noisy state measurements that incur large errors: the robot can quickly get back on track, as demonstrated by experiments I and III.

References:

[1]: Eric A. Wan and Rudolph van der Merwe, *The Unscented Kalman Filter for Nonlinear Estimation*

[2]: Rambabu Kandepu, Bjarne Foss, and Lars Imsland, *Applying the unscented Kalman filter for nonlinear state estimation*