Extended Kalman Filters

The Extended Kalman Filter is a version of the Kalman Filter that can handle nonlinear systems, by linearizing the system with a first order approximation.

Ex: We have a planar quadrotor that has dynamics we model with

$$g(x, u) = x + f(x, u) * \Delta t$$

If we commanded a zero control input, the quadcopter would fall down, which violates zero input zero output property of linear systems. So to linear the system about (x^*, u^*), we have:

$$g(x, u) = g(x^*, u^*) + \frac{\partial g}{\partial u}_{(x^*, u^*)}(x - x^*) + \frac{\partial g}{\partial u}_{(x^*, u^*)}(u - u^*) + H.O.T$$

$$\approx \bar{A}(x^*, u^*)x + \bar{B}(x^*, u^*)u + E(x^*, u^*),$$

where
$\bar{A}(x^*, u^*) := \frac{\partial g}{\partial x}_{(x^*, u^*)}$
$\bar{B}(x^*, u^*) := \frac{\partial g}{\partial u}_{(x^*, u^*)}$
$E(x^*, u^*) := g(x^*, u^*) - \bar{A}(x^*, u^*)x^* - \bar{B}(x^*, u^*)u^*$

Here is the algorithm in full implementation:

$t \leftarrow 0$
$\hat{x}[t] \leftarrow x_0$
$P[t] \leftarrow P_0$
while $t \leq T - 1$ do
  $\hat{x}[t+1|t] \leftarrow g(\hat{x}[t], u[t])$ (state extrapolation)
  $A[t+1] \leftarrow \left.\frac{\partial g}{\partial x}\right|_{(\hat{x}[t], u[t])}$ (dynamics linearization)
  $P[t+1|t] \leftarrow A[t+1]P[t]A[t+1]^\top + Q$ (covariance extrapolation)
  $C[t+1] \leftarrow \left.\frac{\partial h}{\partial x}\right|_{\hat{x}[t+1|t]}$ (measurement linearization)
  $K[t+1] \leftarrow P[t+1|t]C[t+1]^\top \left(C[t+1]P[t+1|t]C[t+1]^\top + R\right)^{-1}$ (Kalman
  $\hat{x}[t+1] \leftarrow \hat{x}[t+1|t] + K[t+1](y[t+1] - h(\hat{x}[t+1|t]))$ (state update)
  $P[t+1] \leftarrow (I - K[t+1]C[t+1])P[t+1|t]$ (covariance update)
  $t \leftarrow t + 1$
end while
return $\hat{x}[t]$ for $t = 0, \ldots, T$

For the planar quadrotor model, we can model the dynamics as follows:

$$\dot{x} = \begin{bmatrix} \dot{x} \\ \dot{z} \\ \dot{\phi} \\ \ddot{x} \\ \ddot{z} \\ \ddot{\phi} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{z} \\ \dot{\phi} \\ 0 \\ -g \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ -\frac{\sin(\phi)}{m} & 0 \\ \frac{\cos(\phi)}{m} & 0 \\ 0 & \frac{1}{J} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = f(x, u)$$

Which then gives us

$$g(x, u) = \begin{bmatrix} x + \dot{x}\Delta t \\ z + \dot{z}\Delta t \\ \phi + \dot{\phi}\Delta t \\ \dot{x} - \frac{\sin\phi}{m} u_1 \Delta t \\ \dot{z} + \left(\frac{\cos\phi}{m} u_1 - g\right)\Delta t \\ \dot{\phi} + \frac{u_2}{J}\Delta t \end{bmatrix}$$

So for step 6, we find A to be

$$\frac{\partial g}{\partial x} = \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \cdots & \frac{\partial g_n}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_1}{\partial x_n} & \cdots & \frac{\partial g_n}{\partial x_n} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & -\frac{\cos\phi}{m} u_1 \Delta t & 1 & 0 & 0 \\ 0 & 0 & -\frac{\sin\phi}{m} u_1 \Delta t & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Which can be solved by plugging in the values of $\hat{x}, u$

Now for step 8, we are given measurements in the form of

$$y = \begin{bmatrix} r \\ \theta \end{bmatrix}$$

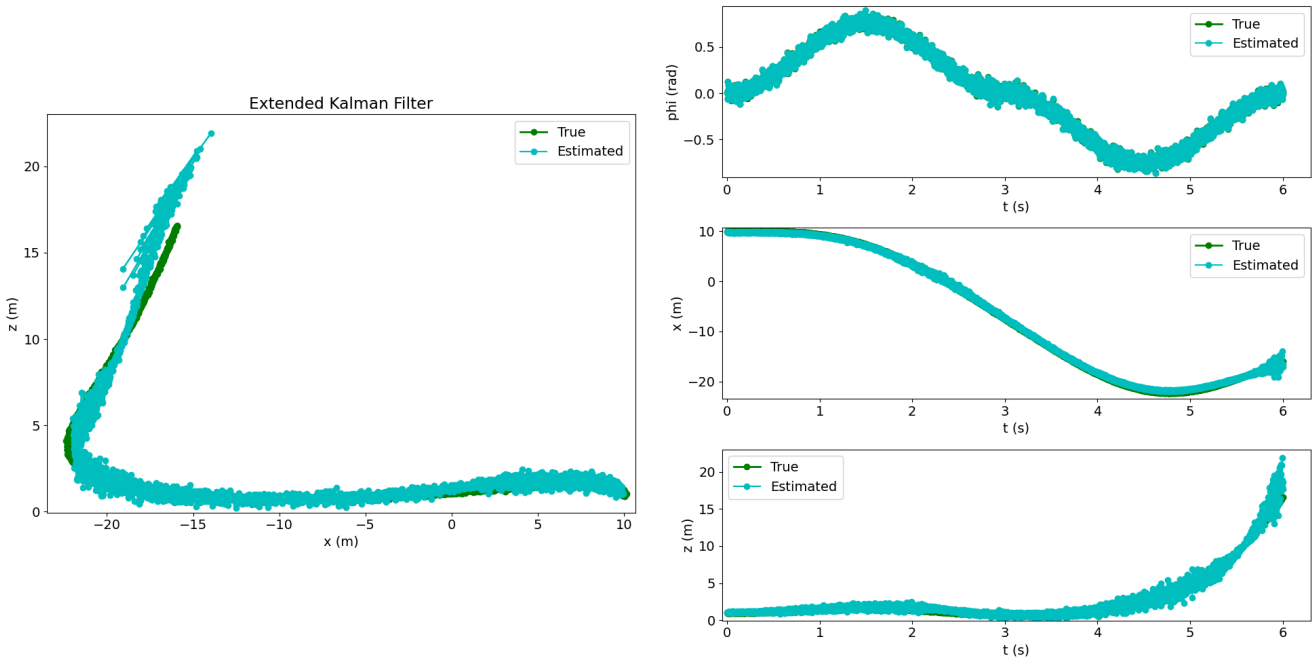
Where r is distance to the landmark, and θ is relative bearing with respect to the landmark, which is positioned at (0, 5, 5)
So we can then turn this into measurements of the drone's state, as follows:

$$y = h(x) = \begin{bmatrix} \sqrt{(l_x - x)^2 + l_y^2 + (l_z - z)^2} \\ \phi \end{bmatrix}$$

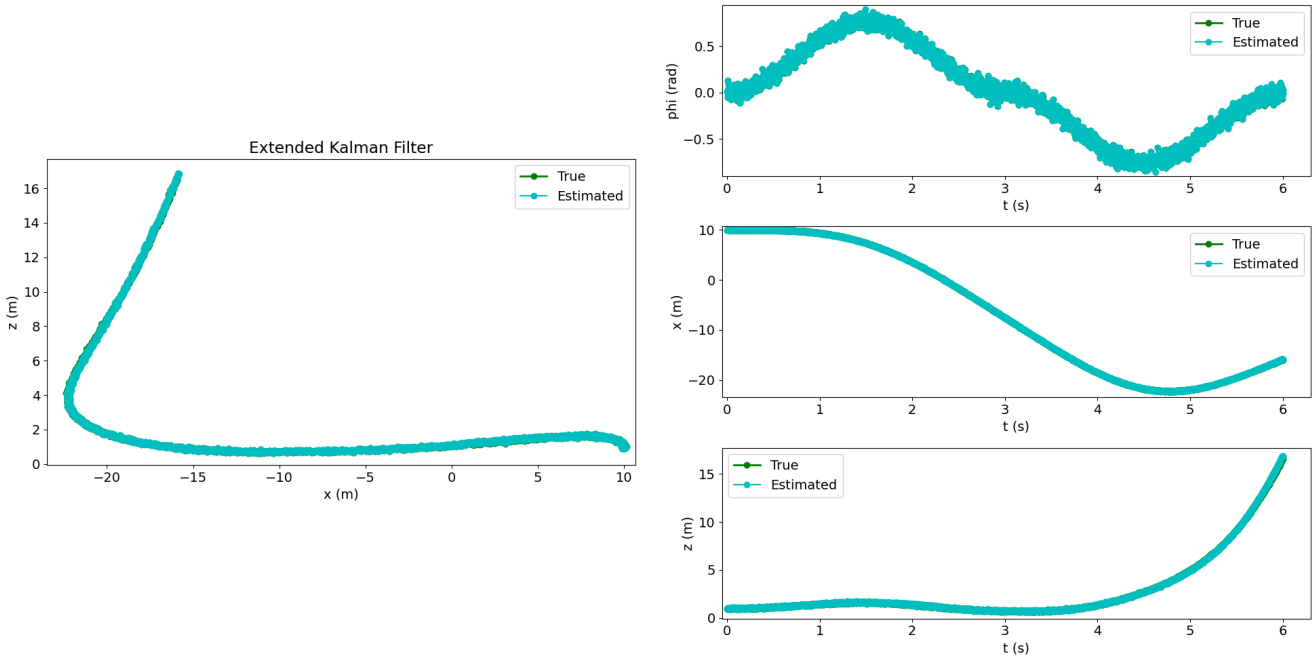
So taking the jacobian of this, we get some semi-annoying derivatives to yield:

$$\frac{\partial h}{\partial x} = \begin{bmatrix} \frac{-(l_x - x)}{\sqrt{(l_x-x)^2+l_y^2+(l_z-z)^2}} & \frac{-(l_z - z)}{\sqrt{(l_x-x)^2+l_y^2+(l_z-z)^2}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

After fighting with np.newaxis and np.flatten, we eventually get a working EKF!



After some careful tuning, we can achieve results that look like this:



Now let's try for the Unicycle model:

$$x := \begin{bmatrix} \phi \\ x \\ y \\ \theta_L \\ \theta_R \end{bmatrix}$$

The turtlebot has dynamics model:

$$\dot{x} = f(x,u) := \begin{bmatrix} -\frac{r}{2d} & \frac{r}{2d} \\ \frac{r}{2}\cos\phi & \frac{r}{2}\cos\phi \\ \frac{r}{2}\sin\phi & \frac{r}{2}\sin\phi \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_L \\ u_R \end{bmatrix}$$

We can now get $g(x,u)$, using the discrete dynamics model:

$$g(x,u) = x + f(x,u)\Delta t = \begin{bmatrix} \phi + \frac{r}{2d}(u_L + u_R)\Delta t \\ x + \frac{r\cos\phi}{2}(u_L + u_R)\Delta t \\ y + \frac{r\sin\phi}{2}(u_L + u_R)\Delta t \\ \theta_L + u_L\Delta t \\ \theta_R + u_R\Delta t \end{bmatrix}$$

Now to approximate A in step 6, we find $\frac{\partial g}{\partial x}$ to be:

$$A(x,u) \approx \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -\frac{r}{2}(u_L + u_R)\sin(\phi)\cdot\Delta t & 1 & 0 & 0 & 0 \\ \frac{r}{2}(u_L + u_R)\cos(\phi)\cdot\Delta t & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Then we have the measurements, which we recieve from the Turtlebot odom transform. Fortunately, this gives us the turtlebots ϕ, x, y. So in step 8, we can define C to simply extract those elements from the state

$$y[t] = h(x[t], v[t]) := Cx[t] + v[t]$$

Where

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

Where $v[t]$ is measurement noise, and since C is the almost identity matrix, it simply means we are extracting the exact state we recieve from odom, plus the measurement noise.
There is no need to linearize the measurement model here, as we are directly given the states.

Implementing this, we get the following results: