IGVC Kalman Filter Design

Justin Kleiber

March 16, 2020

State Representation

$$\hat{\boldsymbol{x}} = \begin{bmatrix} \phi & \lambda & \theta & x & y & \psi & v & \dot{\psi} & v_l & v_r & a \end{bmatrix}^T \tag{1}$$

ϕ: latitude
λ: longitude
θ: heading in global coordinate space (clockwise is positive)
x: x position, moving forward increases it, moving backward decreases it
y: y position, left increases it, right decreases it
ψ: yaw - heading in local coordinate space (counterclockwise is positive)
v: robot forward velocity
$\dot{\psi}$: the angular velocity of the robot around the z-axis (yaw rate)
v_l: velocity of left wheel
v_r: velocity of right wheel
a: robot forward acceleration

Motion Model

Overview

The motion model defines the behavior of the robot when it moves through space. The following equations and derivation leads us to the equation to calculate the robot's state, and the linearized system used to predict the state using the EKF.

General Form

$$\hat{\boldsymbol{x}}_k = f(\hat{\boldsymbol{x}}_{k-1}, \mathbf{u}, \Delta t) + w_k \tag{2}$$

$\hat{\boldsymbol{x}}_k$: the new state estimate
$\hat{\boldsymbol{x}}_{k-1}$: the last state estimate
$\mathbf{u}$: the control signal
Δt: the time step (time between each state estimate)
w_k: the process noise associated with the filter

where $\mathbf{u}$ is of the form $\mathbf{u} = \begin{bmatrix} u_l \\ u_r \end{bmatrix}$ and u_l is the commanded angular velocity for the left wheel and u_r is the commanded angular velocity for the right wheel.

Supporting Equations

The following equations are simple physics equations that will be used to approximate the robot's location. For the purposes of this filter, everything will be assumed to be in metric units.

GPS coordinate frame switching:

First, displacement is given by the following equation (these x variables are the x-coordinate, not the state vector):

$$d = \frac{\sqrt{(x_k - x_{k-1})^2 + (y_k - y_{k-1})^2}}{1000} \tag{3}$$

Note: displacement is converted to kilometers here, for use by the rest of the conversion

Bearing from the $k-1$-th point to the k-th point is assumed to be θ_{k-1}, and the earth's polar radius is denoted by $R_{earth} = 6356km$ (For reference, the equatorial radius is 6378.1 km. Either value works).

Latitude

$$\phi_k = sin^{-1}(sin(\phi_{k-1})cos(d/R_{earth}) + cos(\phi_{k-1})sin(d/R_{earth})cos(\theta)) \tag{4}$$

Longitude

$$\lambda_k = \lambda_{k-1} + atan2(sin(\theta)sin(d/R_{earth})cos(\phi_{k-1}), cos(d/R_{earth}) - sin(\phi_{k-1})sin(\phi_k)) \tag{5}$$

where ϕ_k represents the calculation made by Equation 4.

The above equations assume the bearing, latitude, and longitude are measured in radians. This also assumes the displacement is in units of kilometers.

Differential Drive Approximation

The robot's position can be approximated using wheel velocities and first order physics equations. The Component velocity vectors of x and y can be found by linearizing the motion, as shown by http://planning.cs.uiuc.edu/node659.html, where r is wheel radius.

$$\dot{x} = \frac{r}{2}(v_l + v_r)cos(\psi) \tag{6}$$

$$\dot{y} = \frac{r}{2}(v_l + v_r)sin(\psi) \tag{7}$$

$$\dot{\psi} = \frac{r}{L}(v_r - v_l) \tag{8}$$

From this we can find new values of x, y and ψ:

$$x = x_{k-1} + \dot{x}\Delta t \tag{9}$$

$$y = y_{k-1} + \dot{y}\Delta t \tag{10}$$

$$\psi = \psi + \dot{\psi}\Delta t \tag{11}$$

Since the global heading changes at the same rate as local heading, the same equation for ψ works for θ (just make sure units match between these two values):

$$\theta = \theta + \dot{\psi}\Delta t \tag{12}$$

Model Function

To estimate the current state, we solve (2). Given the supporting equations above, the following dynamics is used to predict the robot's current state based on the previous state information:

$$f(\hat{\boldsymbol{x}}_{k-1}, \mathbf{u}_k, \Delta t) = \begin{bmatrix} sin^{-1}(sin(\phi_{k-1})cos(d/R_{earth}) + cos(\phi_{k-1})sin(d/R_{earth})cos(\theta)) \\ \lambda_{k-1} + atan2(sin(\theta)sin(d/R_{earth})cos(\phi_{k-1}), cos(d/R_{earth}) - sin(\phi_{k-1})sin(\phi_k)) \\ \theta_{k-1} + \dot{\psi}_{k-1}\Delta t \\ x_{k-1} + \dot{x}\Delta t \\ y_{k-1} + \dot{y}\Delta t \\ \psi_{k-1} + \dot{\psi}_{k-1}\Delta t \\ v_{k-1} + a_{k-1}\Delta t \\ \frac{v_{r_{k-1}} + v_{l_{k-1}}}{L} \\ \mathbf{u}_{l_k} \\ \mathbf{u}_{r_k} \\ a_{k-1} \end{bmatrix} \tag{13}$$

We need to linearize this system so the EKF can predict the covariance matrix. This is accomplished by calculating the Jacobian, F_k, of $f(x, \mathbf{u}, \Delta t)$. In this case, the Jacobian is found using a Python package named sympy. Sympy is able to solve symbolic equations, and for this application it calculates the Jacobian of equation 13 symbolically. In the filter implementation, the Jacobian matrix generated by sympy will be used by substituting in the most recent state and control variable values like so:

$$F_k(\hat{\boldsymbol{x}}_{k-1}, \mathbf{u}, \Delta t) \tag{14}$$

This will be used later to calculate covariance. Note: the jacobian is not used to calculate the predicted state - that is the role of equation 13

Measurement Model

The measurement model seeks to correlate a given state with the available sensor values (measurements). A function is used to compute the sensor data, and then noise is added. The following equation demonstrates this.

General Form

$$z_k = h(\hat{\boldsymbol{x}}_k) + v_k \tag{15}$$

z_k: sensor values
$\hat{\boldsymbol{x}}_k$: current state estimate (from the prediction above)
v_k: measurement noise of the sensors

Sensors

Our robot has a few sensors to utilize when taking measurements:

- Wheel Encoders
- Accelerometer
- Gyroscope
- Magnetometer
- GPS

Model Function

We need a way to convert a given state estimate to the sensor values. In the case of our robot, we choose to represent the state estimate as a one-to-one mapping with the sensor values. For most of these variables, this is an intuitive mapping (i.e. the GPS estimates match up directly with the GPS sensor's readouts). However, for the x and y coordinates, we keep a dead-reckoning estimate of the position using raw sensor values, and then compare it to the EKF's estimate. Thus,

$$h(\hat{\boldsymbol{x}}_k) = \hat{\boldsymbol{x}}_k \tag{16}$$

Like with the motion model, this system needs to be linearized in order for it to work. The Jacobian of $h(\hat{\boldsymbol{x}}_k)$ is found by inspection: $H_k = \mathbf{I}_{11}$ (the identity matrix in 11-space).

Extended Kalman Filter Formulation

Overview

The modelling above will help run the EKF. To make an EKF, two phases are needed - the prediction phase and the update phase. First, the prediction phase estimates the state of the robot, and then the update phase uses sensor data to gain feedback on that prediction.

Reading Notes

To this point, we've been dealing with things we know to be true. For example, in the measurement model, a state was used to calculate the sensor values. In the EKF, this is no longer deterministic - predicted values will be used throughout the filter. In order to differentiate things that are predicted from things that are not predicted, a little "hat" will be used over the variable name:

$\hat{\boldsymbol{x}}_k$: estimated state
$\boldsymbol{x}_k$: actual, real-life state

This is generally consistent with the literature on EKFs out there right now. Since the modelling part is very concrete and the EKF uses similar equations (but with predictions), it is easy to get the two mixed up.

Prediction Phase

The prediction phase is described by two main steps - the state prediction step and the covariance update step. Put simply, the covariance of the state prediction is how uncertain the prediction is.

The Process

1. Predict the next state:
$$\hat{\boldsymbol{x}}_k = f(\hat{\boldsymbol{x}}_{k-1}, \mathbf{u}_k, \Delta t) \tag{17}$$
2. Calculate the Jacobian, F_{k-1}, of $f(\hat{\boldsymbol{x}}_{k-1}, \mathbf{u}_k, \Delta t)$
3. Then, update the covariance matrix, P_k:
$$P_k = F_{k-1} P_{k-1} F_{k-1}^T + Q_{k-1} \tag{18}$$

It is important to note that Q is a diagonal square matrix representing the process noise of the motion model (see equation 2). Setting Q to even a small value is really helpful for tuning the filter.

Update Phase

With a predicted state and its uncertainty calculated, the sensor measurements need to be considered. Then, once sensors are integrated, we will weight the uncertainties to find the tradeoff between the motion and measurement models. This tradeoff is called the Kalman Gain.

The Process

1. Calculate the innovation (the amount learned from this update). This is the difference in the actual sensor values and the predicted ones:
$$y_k = z_k - h(\hat{\boldsymbol{x}}_k) \tag{19}$$
Note here that z_k is not predicted. It will be the vector defined in equation **??**
2. Calculate the covariance of the innovation (the uncertainty due to measurement noise):
$$S_k = H_k P_{k-1} H_k^T + R_k \tag{20}$$
Note here that R is a diagonal square matrix representing the measurement noise of the system (see equation 15). Data sheets or experimentation can be used to find the best values of R. In general, the square of each sensors standard standard deviation (called the variance and calculated as σ^2) is used on the corresponding entry.
3. Calculate the Kalman Gain
$$K_k = P_{k-1} H_k^T S_k^{-1} \tag{21}$$

4. Update the predicted state

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k + K_k y_k \tag{22}$$

5. Update the estimated covariance

$$P_k = (I - K_k H_k) P_k \tag{23}$$

Running the EKF

When testing the robot, observe the P_k matrix value on a graph to make sure the EKF is converging. If it is not converging, or converging slowly, adjusting Q and R might be necessary.

Appendix

A Exact Differential Drive Calculation

Local Robot Dynamics

While traditional physics equations are used for this application, the exact motion model of a differential drive were considered. The main reason the following equations were not used is due to the potential for a divide by zero when wheels turn the same speed. The remedy for this issue results in a function that cannot be differentiated (as it is discontinuous). These equations are from
http://www.cs.bham.ac.uk/internal/courses/int-robot/2014/lectures/14-ir-kinematics.pdf. The notation for the constants is also below.

Notation:
L: wheelbase length

Angular Velocity

$$\dot{\psi}_k = \frac{v_{r_{k-1}} + v_{l_{k-1}}}{L} \tag{24}$$

Turn Radius

$$R = \frac{L(v_l + v_r)}{2(v_r - v_l)} \tag{25}$$

Next, the instantaneous center of curvature (ICC) is found using the above three equations

$$ICC = [X, Y] = [x - Rsin(\psi), y + Rcos(\psi)] \tag{26}$$

Equation for x_k (note: this is the x position, not the state):

$$x_k = [(x_{k-1} - ICC_x)cos(\dot{\psi}\Delta t) - (y - ICC_y)sin(\dot{\psi}\Delta t)] + ICC_x \tag{27}$$

Equation for y_k (note: the x used here is the position, not the state):

$$y_k = [(x_{k-1} - ICC_x)sin(\dot{\psi}\Delta t) + (y - ICC_y)cos(\dot{\psi}\Delta t)] + ICC_y \tag{28}$$

Equation for ψ_k:

$$\psi_k = \psi_{k-1} + \dot{\psi}\Delta t \tag{29}$$