Kalman Filter for Position Estimation

By Sophie Bredenkamp, ECE '21

Introduction

The Shamrock team has been developing a hazard detection system for rover landing on an asteroid this semester in senior design. The project focuses on image and LiDAR processing to determine a safe landing site on a hazardous surface. While most of the progress so far has been in processing simulated data for both the image and LiDAR systems, we plan to implement our algorithms on data collected from a physical experimental set up. We have been working with various resources at Tufts to develop a model asteroid in Halligan Hall and will fly a drone over the set up to collect image and LiDAR data for post-flight processing. We will collect terrain relative navigation data as well to place our measurements on a larger global map.

An important component for correctly interpreting the data collected will be precise locations of the drone when each frame is collected. For this part of the project, we chose to develop a Kalman Filter, taking in inertial measurement unit (IMU) and terrain relative navigation (TRN) data, and returning an estimated position for each frame. This will help develop a robust view of the terrain we are looking at as compared to the global map. For the simulation stage of algorithm development, the Kalman Filter can be implemented using simulated IMU data that can closely resemble the IMU performance.

Research

The Kalman Filter uses state space algorithms to determine correct measurements in systems with noise. Using previous sensor data, estimated changes in parameters, and covariance information, the Kalman Filter estimates the actual output as compared to an input measurement from reality. This algorithm can be applied to position, direction, timing offsets, and accelerometer offsets to name a few.

The Kalman Filter algorithm is a discrete time system that consists of a set of state space equations. Each equation represents a system of equations, including matrices to dictate the effect of each variable on the output. The equations in the system are a state prediction, a covariance prediction, an innovation, an innovation covariance, a Kalman gain equation, a state update equation, and a covariance update equation. While this sounds like a very complex system, it is made up of purely linear equations of reasonably limited size. The only necessary technical knowledge required for understanding the mechanics of this algorithm is in matrix algebra and limited statistics.

The algorithm works by taking in a vector of control inputs and measurement inputs, so input vectors from each of the two inputs to the system. It uses the control input and matrices that determine the effect of inputs and previous states to determine a predicted state. The covariance (or error) prediction for the state is determined using estimated process error and the state transition matrix (the matrix used to determine the effect of a previous state on the next). Using the measurement input, the innovation equation is used to come up with a comparison value between the prediction and reality. The innovation covariance equation determines the comparison between the between real error and predicted error. The transition matrix, covariance prediction, and innovation covariance are used to determine the Kalman gain which is then in turn used to moderate the state prediction and output the state output. Finally, the covariance update is determined using the Kalman gain, the transition matrix, and the covariance prediction. The final outputs of the system are the state update and the covariance update. These are then used to determine the future time steps of the algorithm. The equations described are as follows:

Prediction

$$\hat{x}_k = A x_{k-1} + B u_k$$
$$\hat{P}_k = A P_{k-1} A^T + Q$$

Observation

$$y_k = z_k - H\hat{x}_k$$
$$S_k = H\hat{P}_k H^T + R$$

Update

$$K = P_k H^T / S_k$$
$$x_k = \hat{x}_k + K y_k$$
$$P_k = (I - KH)\hat{P}_k$$

As an example, determining position of a moving object with known position is a common application of the Kalman Filter. For this example, the control input would be an expected acceleration and the measurement input would be an acceleration pulled from an IMU (this happens to be the first application developed for our project). The states for this application are position, velocity, and acceleration bias. This most simplified example only tracks one dimension but can be easily expanded into two or three dimensions as the intermediate equations are the same. This example uses simple mechanics equations to construct the transition matrix, as well as other matrices involved, and determine position from acceleration alone.

Algorithm

My approach for implementing this algorithm in MATLAB was to create classes for each parameter I want to track, a class for the Kalman Filter itself, and a script to run the filter on each parameter. For the Kalman Filter class, I included a function to take in all the constants from the parameter class and a function to step through the filtering. This class holds all the constant matrices and intermediate vectors as properties.

The position class follows the position algorithm described above. This class holds the time step, position measurement noise and acceleration measurement noise as properties and contains a single function to set all the matrix values to be fed into the Kalman Filter. These values depend entirely on the sample period, time step, position measurement noise and acceleration measurement noise.

The IMU data collected from the flight is converted to a stream of positions that are then fed to the filter with periodic correction by the Terrain Relative Navigation (TRN), which is a separate algorithm. The preliminary results are shown in the figure below. The Kalman Filter output smooths the IMU data while still following it closely. Further improvements can still be made to the error estimates to provide a more accurate correction.

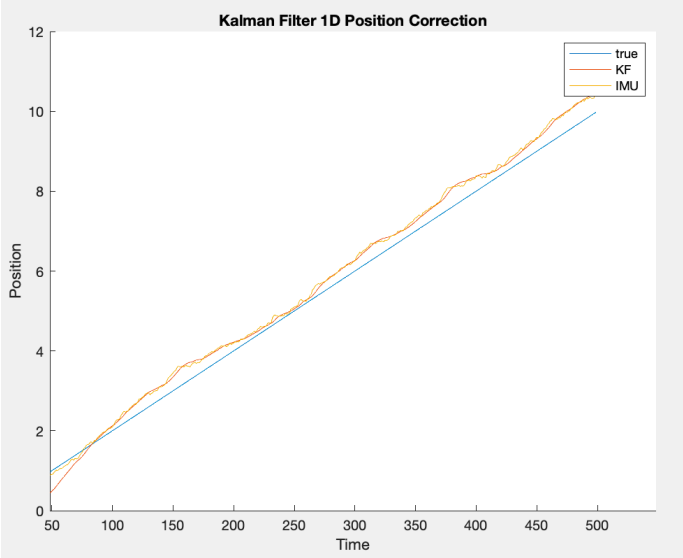

Figure 1. Kalman Filter Implementation for 1 dimensional position

Conclusion

In conclusion, the Kalman filter is a very powerful tool for eliminating noise in a system. Since our project relies heavily on knowing exactly where the drone is in space to determine the correct landing site, the Kalman filter is incredibly important. It is a versatile algorithm that can be applied across the project, beyond just motion estimation. While there are many moving parts, the algorithm is fundamentally a series of linear equations that can be implemented easily in software to enhance the accuracy of the project output.

References

1. Czerniak, Gregory. “Greg Czerniak's Website.” *Greg Czerniak's Website - Kalman Filters for Undergrads 1*, greg.czerniak.info/guides/kalman1/.

2. “How a Kalman Filter Works, in Pictures.” *Bzarg*, 11 Aug. 2015, www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/.

3. Welch, Greg, and Gary Bishop. *An Introduction to the Kalman Filter*.

4. “Introduction to Simulating IMU Measurements.” *Introduction to Simulating IMU Measurements - MATLAB & Simulink*, www.mathworks.com/help/nav/ug/introduction-to-simulating-imu-measurements.html.

5. Romaniuk, S., & Gosiewski, Z. (n.d.). *Kalman Filter Realization for Orientation and Position Estimation on Dedicated Processor* (pp. 88-94, Rep.). Bialystok, Poland: Department of Automatic Control and Robotics. doi:0.2478/ama-2014-0016