University of California, Los Angeles

ECE 183DA

Lab 2

Gabriel Baltazar, 904754939

Aditi Mittal, 104736807

Everett Sheu, 704796167

Table of Contents

Background and Purpose

The purpose of this lab was to develop an Extended Kalman Filter to be used as a state estimator for our model car. The Kalman Filter is used to estimate state variables by combining knowledge of previous states, sensor measurements, and noise models of the robot. The purpose of the Kalman Filter is to filter out the noise from the data to find the best estimate. The Extended Kalman Filter is the nonlinear version of the Kalman Filter, and linearizes the data around the current mean and covariance. The following are the key equations for the state estimation:

$$\boldsymbol{x}_k = f(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) + \boldsymbol{w}_k$$

$$\boldsymbol{z}_k = h(\boldsymbol{x}_k) + \boldsymbol{v}_k$$

x refers to the state transition model, **z** refers to the observation model, and **u** refers to the input. **w** and **v** are the process and observation noises; their respective covariances are referred to as **Q** and **R**, which are used to estimate the covariances for successive observations. Function ***f*** is used to compute the new predicted state based on the previous state, and function ***h*** is used to compute the new predicted state measurements from the predicted state computed by ***f***.

However, because functions ***f*** and ***h*** cannot be applied to the covariance directly, the actual state transition and observation matrices are obtained using matrices of partial derivatives, namely the Jacobian matrices ***F*** and ***H***:

$$\boldsymbol{F}_k = \left.\frac{\partial f}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{u}_k}$$

$$\boldsymbol{H}_k = \left.\frac{\partial h}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{k|k-1}}$$

Materials and Method Overview

In this lab, we used the PaperBot from the previous lab, which includes the following components: a paper chasis, the ESP8266 microcontroller, its shield, two servo motors, two V15310X laser range sensors, and the MPU9250 inertial measurement unit (IMU). These components went into the paper chasis and were wired as per the circuit diagram in **Figure 2** below.

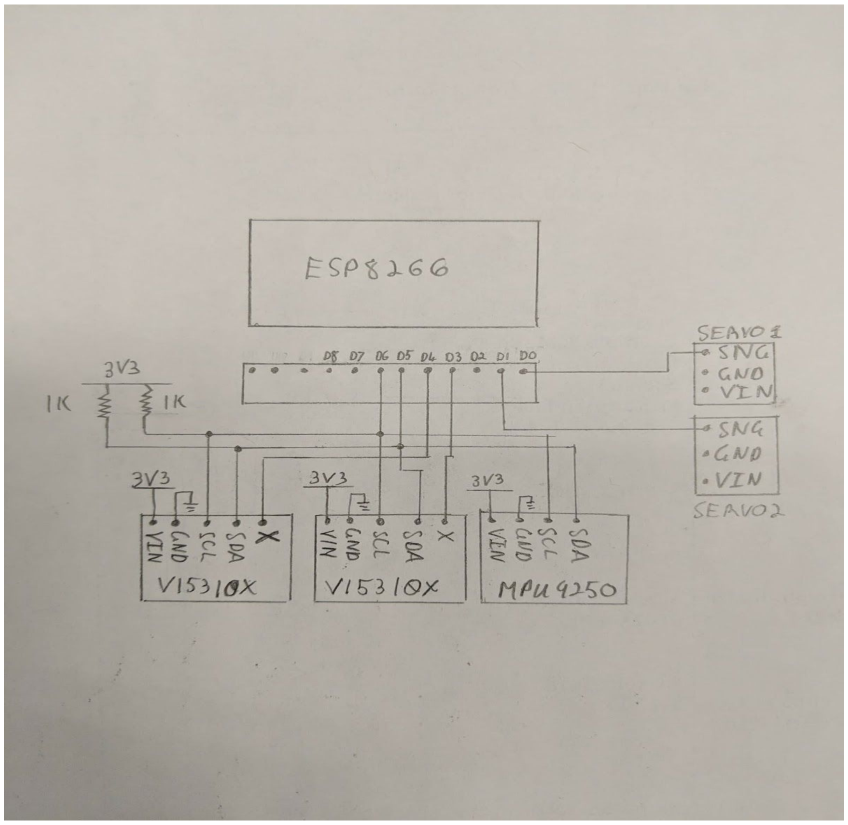

Figure 2: Circuit Diagram

In terms of software, from the previous lab we already had Arduino code that controlled the Paperbot's servo motors, made laser range sensor readings, and angular measurements. From these measurements, we obtained our PaperBot's actuation and measurement models. Our task for this lab was to incorporate our Extended Kalman Filter state estimator so as to augment our PaperBot's ability to track its own state in the world.

This was accomplished by first developing the mathematical model of the Kalman Filter and determining all of the appropriate parameters for the prediction equations. This was then translated into Python code. This script is called ekf.py and handles the Extended Kalman Filter's gain update and state estimation procedures. This Extended Kalman Filter went through two major iterations which are further explained in the Results and Discussion section. At first, we believed that we theoretically had a satisfactory version of the Estimated Kalman Filter that would provide an accurate representation of the state for the path of the paperbot; however, with experimentation, it became evident that this Kalman Filter V1 would not perform to an adequate level of its goal. Due to this first attempt leading to failure, we decided to completely overhaul the system and develop the Kalman Filter V2. This new version of the Kalman Filter took advantage of code external to Arduino and with this new implementation, the state estimation was significantly more accurate, and it performed its function of improving its state representation as time passed on.

Extended Kalman Filter

As stated previously, the Kalman Filter is used to estimate state variables by combining knowledge of previous states, sensor measurements, and noise models of the robot. The purpose of the Kalman Filter is to filter out the noise from the sensor data and input streams to find the best state estimate. The Extended Kalman Filter is the nonlinear version of the Kalman Filter.

For our implementation, we chose the following:

Input

$$U = [\ \tau_L,\ \tau_R\ ]$$

Where $\tau_L,\ \tau_R$ are the servo inputs in metrics of PWM. To simplify the Kalman Filter, we decided to keep the default PWM values of each of the directional inputs. Thus, the only thing to account for is what orientation each wheel is spinning in and for how long. These values capture both aspects by denoting how long the command was held for in milliseconds as well as negating the value if the wheel is spinning backwards. For example, holding right turn for one second would yield the following:

$$\tau_L = 1000,\ \tau_R = \ -1000$$

Sensor Measurements

$$Y = [\ l_x,\ l_y,\ \theta\ ]$$

Where $l_x,\ l_y,\ \theta$ are the readings from the front range sensor, right range sensor, and IMU sensor respectively. The range sensors read distances in millimeters while the IMU reads the bearing in degrees

State

$$X = [\ x_g,\ y_g,\ \theta\ ]$$

Where $x_g,\ y_g,\ \theta$ are the global coordinates and bearing. These values are also in millimeters and degrees respectively. (Note, we can directly use θ from the sensor measurements given by the magnetometer).

Procedure

The Kalman Filter can be broken down into two main components: the Gain Update and the State Estimation.

Kalman Gain Update:

1. Initialize $t = 1$, Initialize Error Covariance $P_{1|0} = P_0$
2. Compute the Gain

   $K_t = P_{t|t-1} H_t^T (H_t^T P_{t|t-1} H_t^T + R_t)^{-1}$
3. Update Error Covariance

   $P_t = (I - K_t H_t) P_{t|t-1}$

   $P_{t+1} = A_t P A_t^T + G_t Q_t G_t^T$
4. Increment t

   $t += 1$
5. Jump back to step 2. Until Stop Condition

   $Q_t = E[w_t w_t^T] = 0$

   $R_t = E[v_t v_t^T]$

State Estimation:

1. Initialize $t = 1$, Initialize state estimate $\widehat{X}_0$
2. Poll measurements into Y_t
3. Update State Estimate with the new measurement and Kalman Gain.

   $\widehat{X}_{t|t-1} = A_{t-1} \widehat{X}_{t-1}$

   $\widehat{X}_t = \widehat{X}_{t|t-1} + K_t (Y_t - H_t \widehat{X}_{t|t-1})$
4. Increment t

   $t += 1$
5. Jump back to step 2. Until Stop Condition

   $Q_t = E[w_t w_t^T] = 0$

   $R_t = E[v_t v_t^T]$

These procedures are combined to compose the Kalman Filter.

Results and Discussion

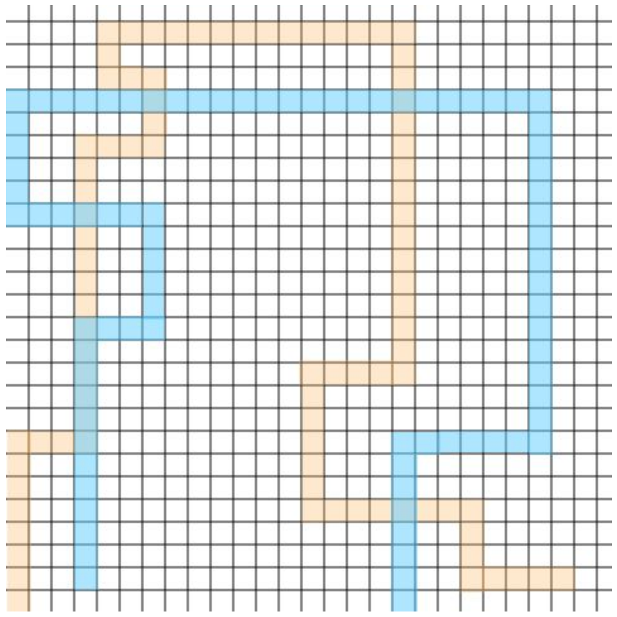

Figure 3: Dynamics Model Estimated Path vs Actual Path

For our first attempt at state estimation, we simply used our Dynamics Model. We placed our PaperBot in a 400mm by 400mm region bounded by cardboard. We placed our PaperBot in the bottom left corner of this bounded region to initiate its initial position as (0, 0) with the x and y limits being 400 and 400 respectively. Because we had difficulties in transmitting commands to the PaperBot from our Python script, we manually steered the bot by using the original basic html server controls.

Figure 3 compares the estimated path by the Dynamics Model with the actual path. The blue path shows the Kalman Filter Estimate while the orange line showing the real-life path results provided by the PaperBot. As we can see, the estimate gets off to a bad start by already being considerably off from the starting position. As time progressed, this model continued to struggle to adapt to turns and hone in on the actual state of the PaperBot. After multiple attempts with this Kalman Filter, we realized that we needed a revamp which led us to our Kalman Filter shown in **Figure 4**.

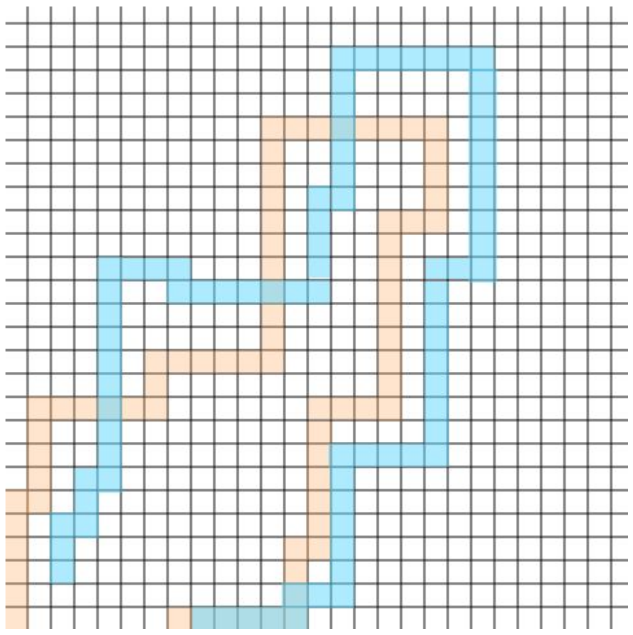

Figure 4: Kalman Filter Estimated Path vs Actual Path

For our second attempt at state estimation, we implemented the Kalman Filter. We once again placed our PaperBot in a 400mm by 400mm region bounded by cardboard. We placed our PaperBot in the bottom left corner of this bounded region to initiate its initial position as (0, 0) with the x and y limits being 400 and 400 respectively. Because we had difficulties in transmitting commands to the PaperBot from our Python script, we manually steered the bot by using the original basic html server controls. We polled the sensor readings from the bot and manually input these to our Extended Kalman Filter.

Figure 4 compares the estimated path by the Kalman Filter with the actual path. The blue path shows the Kalman Filter Estimate while the orange line showing the real-life path results provided by the PaperBot. As we can see, the estimate starts rather far off from the PaperBot's true location. But, as the test goes on, we notice that the estimate begins to converge upon the actual path. By the end of the PaperBot's path, we see that the Kalman Filter Estimate starts to converge upon the true path. It begins to be able to track the actual location of the PaperBot, therefore it provided a substantially accurate state representation of the PaperBot.

Conclusion

We encountered a number of issues that hindered rapid progress of the lab. These included both hardware and software problems that we had to debug.

A major issue we had in implementing the Extended Kalman Filter was a lack of memory. Originally, we had tried to implement the Extended Kalman Filter and perform all of its calculations locally onboard. However we soon realized that our PaperBot was experiencing stability issues and was crashing. Finally, we realized the issue: the Arduino does not have very much RAM to work with. We had utilized so many of the resources in trying to implement the Kalman Filter onboard that the PaperBot could not function properly without flushing and restarting.

Another obstacle of fulfilling our Extended Kalman Filter was the difficulty we had in having our Python script properly transmitting and receiving data to the PaperBot. This forced us to have to manually input data to the Python script that ran the Kalman Filter and manually control the robot. Ideally, we would have had the PaperBot and Python script communicating with each other properly and performing the tests in an automated fashion.

In addition, we had issues calculating the variances for the covariance matrices of the noise models. This was because in order to calculate variances for our state variables, we would need to keep track of sensor readings for each time instance due to the summation nature of the mathematical formula for variance. We implemented a series of arrays to keep track of data for each of the state variables, however, there seemed to be an Arduino memory issue with implementing multiple arrays into our code. This issue was resolved by isolating each array by commenting the other arrays (and their subsequent associated calculations) out, but it would have been much more efficient and seamless if the arrays and their calculations were able to coexist.

In the future, with the implementation of the Extended Kalman Filter, it would make sense for us to perform self-driving capabilities. Since the Estimated Kalman Filter allows us to determine an accurate representation of the paperbot's current state and estimation for the next state based on inputs, our Paperbot should theoretically be able to perform motion planning functions such as driving through an obstacle course or parallel parking between two other obstructions.

Github Repository: https://github.com/esheu2/183DA_lab2