

## **ELEC3210 Project 2: EKF SLAM**

**By: Young, James Yang (20740589)**

In project 2, our main objective was to implement an extended Kalman filter for a 2D landmark-based SLAM system. We were given a ROS package and our main task for this project was to fill out the skeleton code. The skeleton code had 6 to-do parts for us to fill in.

The first to-do had us initialise the covariance matrix, measurement noise matrix, and process noise matrix. I initialised the covariance variable to a 3x3 zero matrix and initialised the noise matrices both with a 2x2 identity matrix. For both of the noise matrices, I tested different noise coefficients and found that for my runs different noise coefficients could lead to very different results.

For example, figure 1 shows the result of choosing a noise coefficient of  $1e-8$  for both noise matrices. As seen in the figure, the run ran into lots of issues when turning during the middle portion of the obstacle. After trying out some different values, the run that best matched the example video given was with measurement noise set to  $1e-10$  and process noise set to  $1e-15$  (see figure 2). However, it still had some issues during the middle part where the vehicle made quick turns, as seen in how some of the point clouds aren't exactly aligned in some areas.

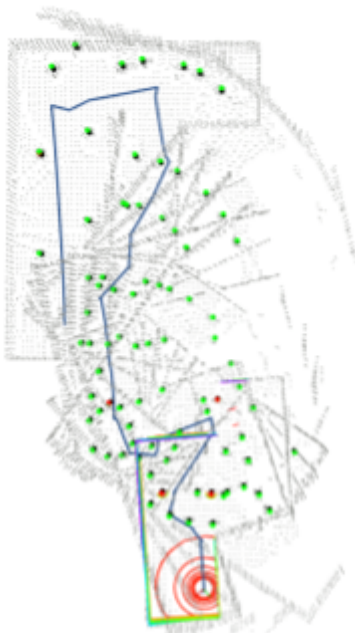


Figure 1: Run with both noise measurements set to  $1e-8$

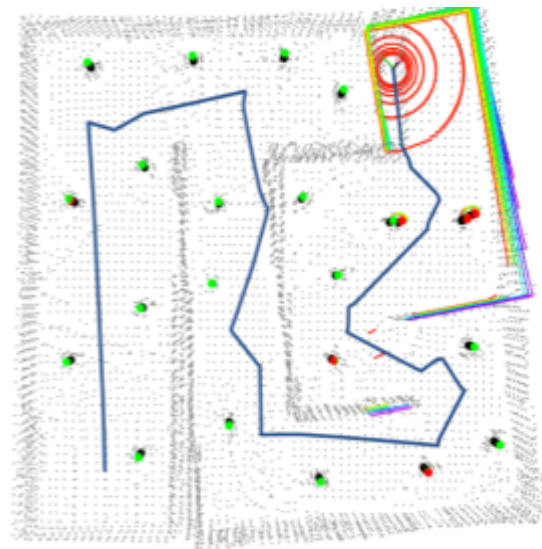
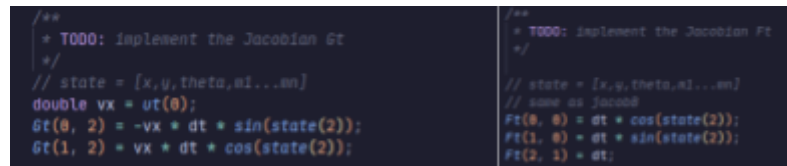


Figure 2: Run with measurement noise set to  $1e-10$  and process noise set to  $1e-15$

The second and third to-do had us implement the Jacobian matrix  $G_t$  and  $F_t$  functions to use in the `predictState` function. This was relatively simple to implement as I just followed the equations and matrices given in the assignment instructions in the motion function part. Figure 3 shows the code for both these functions. Then, I uncommented the update covariance line to finish the `predictState` function.



```
/**
 * TODO: implement the Jacobian Gt
 */
// state = [x,y,theta,m1...mN]
double vx = ut(0);
Gt(0, 2) = -vx * dt * sin(state(2));
Gt(1, 2) = vx * dt * cos(state(2));

/**
 * TODO: implement the Jacobian Ft
 */
// state = [x,y,theta,m1...mN]
// same as Jacob
Ft(0, 0) = dt * cos(state(2));
Ft(1, 0) = dt * sin(state(2));
Ft(2, 1) = dt;
```

Figure 3: Code for calculating Jacobian  $G_t$  and  $F_t$

The fourth to-do had us add the landmark to our state and covariance matrix. Implementing this for both matrices had the same process, where I first created a temporary variable Eigen matrix with expanded size to be able to store both the previous and new information. Then, I stored the previous state and covariance matrix into the temporary variable. Afterwards, I appended the new landmark to the temporary state variable and noise to the temporary covariance matrix. Finally, I set the `mCov` and `mState` variables to the temporary variables to fully update them.

The fifth to-do had us implement the data association here, where we had to find the corresponding landmark for each observation. Inside the for loop that goes through each observation, my implementation finds the closest landmark by calculating the Euclidean distance between all current observations and landmarks. Then, if a valid closest landmark is found (`closestLandmarkIndex` is not -1) and is below a distance threshold, we associate the observation with that landmark by assigning `closestLandmarkIndex` to `indices`.

The final to-do had us implement the measurement update, where we update the state vector and covariance matrix. To do so, using the equations given in the assignment instructions I had to calculate the Kalman gain ( $K$ ), range and bearing variable ( $z$ ), and jacobian measurement model ( $H$ ). This was mainly done with the Eigen library. One issue I ran into while implementing this part was forgetting to keep the angle between  $-\pi$  and  $\pi$  with the given `normalize angle` function, which ended up consuming lots of my time.

Resources Used:

- ChatGPT
- [Assignment 2 Instructions](#)
- [Lecture 9 Slides](#)
- [Lecture 10 Slides](#)
- [https://dspace.mit.edu/bitstream/handle/1721.1/119149/16-412j-spring-2005/contents/projects/1aslambblas\\_repo.pdf](https://dspace.mit.edu/bitstream/handle/1721.1/119149/16-412j-spring-2005/contents/projects/1aslambblas_repo.pdf)