Preamble

This lab focuses on mapping and state estimation using a Turtlebot's sensors, an external indoor positioning system (IPS) and an Extended Kalman Filter (EKF). Specifically, for mapping, the lab aims at generating an occupancy grid using laser scan data and Bresenham's line plotting algorithm, in conjunction with a logodds update. The state estimation portion is based on implementing an EKF using position data modified with additive gaussian noise, requiring the engineer to test various noise covariance matrices to produce suitable state predictions.

Mapping

The requirements of the mapping portion of the lab are detailed as follows:

i. Create line plotting code using Bresenham's line plotting algorithm
ii. Implement a log odds update algorithm
iii. Using Kinect sensor data, construct a 2D occupancy grid of the area being mapped.

Assumptions

i. The robot follows a non-holonomic, two-wheel differential-drive model
ii. The IPS data used for positioning is sufficiently accurate such that filtering of data is not required.

Motion Model and Control Input

The robot is assumed to follow the motion model:

$$x_t = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} x_{1,t} \\ x_{2,t} \\ x_{3,t} \end{bmatrix}$$
$$= \begin{bmatrix} x_{1,t-1} + Tu_{1,t}\cos x_{3,t-1} \\ x_{2,t-1} + Tu_{1,t}\cos x_{3,t-1} \\ x_{3,t-1} + Tu_2 \end{bmatrix}$$

Where T is the timestep, and u is the input vector defined as:

$$\begin{bmatrix} u_{1,t} \\ u_{2,t} \end{bmatrix} = \begin{bmatrix} v \\ \omega \end{bmatrix}$$

With v being the forward speed of the robot and ω being the rotational speed.

The EKF requires that this motion model is linearized. This is done by taking the Jacobian, which results in

$$G = \begin{bmatrix} 1 & 0 & -Tu_{1,t}\sin x_{3,t-1} \\ 0 & 1 & Tu_{1,t}\cos x_{3,t-1} \\ 0 & 0 & 1 \end{bmatrix}\Bigg|_{x_{t-1}=\mu_{t-1}}$$

Measurement Model

The measurement model represents how the robot's positional data is transformed into a sensor reading. In this lab, the sensor used was the IPS, which directly measures the robot's position. The measurement model is therefore

$$z_t = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{1,t} \\ x_{2,t} \\ x_{3,t} \end{bmatrix}$$

Mapping Results

With the above, the mapping algorithm is implemented by functionalizing each part of the requirements and executing them sequentially. First, the algorithm is tested in a simulation environment in Gazebo, and then in the live system.

The simulation is performed in a random Gazebo world with cylinders and cubes the results of the mapping are shown below:

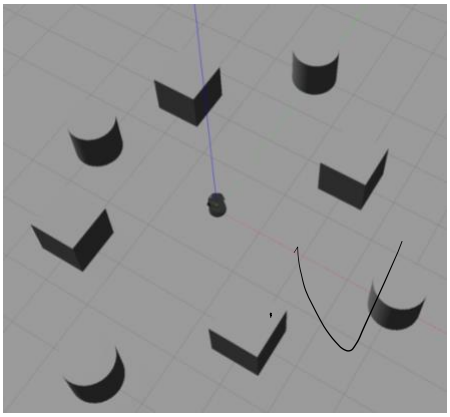

Figure 1 - Gazebo Simulation World

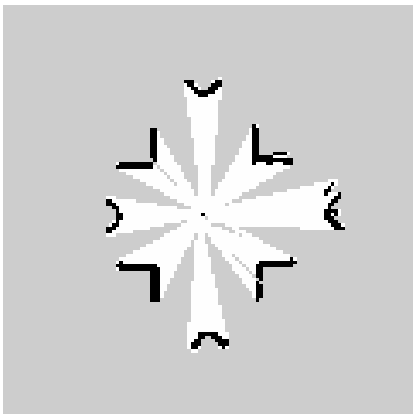

Figure 2 - Simulation Map

The results of the mapping in the live system are shown in three phases:

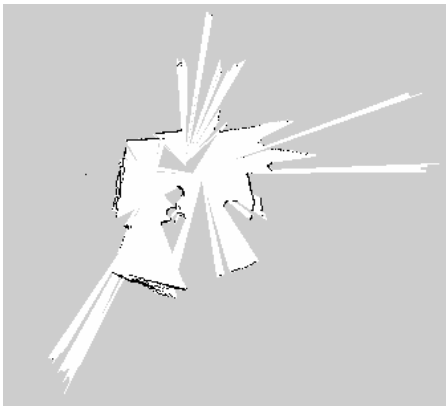

Figure 3 - Initial Mapping Stage (1/3)

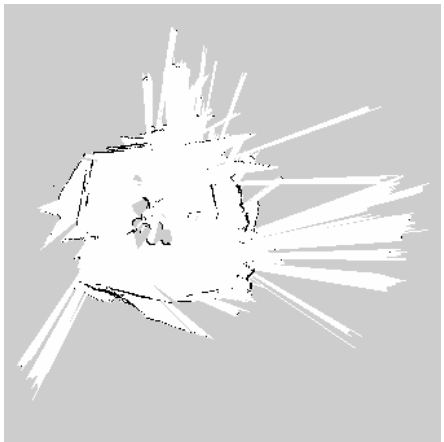

Figure 4 - Second Mapping Stage (2/3)

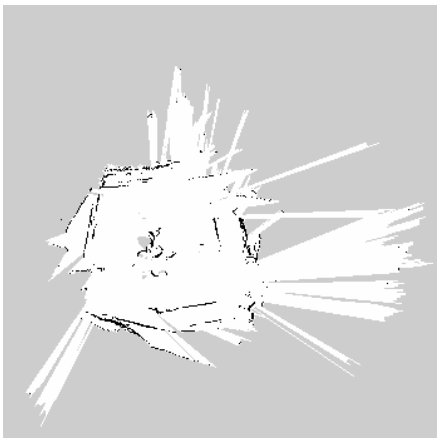

Figure 5 - Final Mapping Stage (3/3)

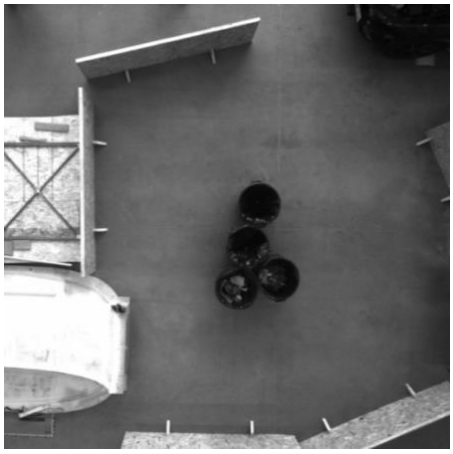

Figure 6 - Mapping Location

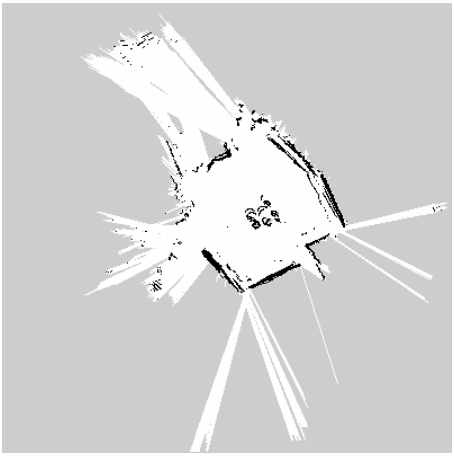

Figure 7 - Mapping Using Odometer Readings

As a point of comparison, the same mapping was performed using only the odometer reading from the Turtlebot; Figure 7 shows that odometer

readings result in less noise, and thicker boundaries than with the maps created using the IPS readings. The mapping with odometry performs similarly to the simulation shown in Figure 2, however the maps using IPS data are worse. A reason for this is that greater processing power is required for IPS mapping, compared to odometer mapping. The algorithms are run in a VirtualBox environment, which severely limits the map-rendering capabilities of the system.

State Estimation

This portion of the lab requires the following:

i. Design an EKF algorithm for the robot's state estimation
ii. Create an algorithm for drawing the EKF covariance ellipse
iii. Introduce additive gaussian noise with a standard deviation of 10 cm to the IPS data
iv. Using wheel odometer readings and the degraded IPS data, perform EKF-based state estimation
v. Tune the noise covariance matrices to produce suitable results for the filter.

As such, the code for each of these requirements has been functionalized and the steps have been performed sequentially. Note that an assumption made for iv above is that the sensor data must be *fused*, i.e. the IPS provides position data, while the wheel odometer provides linear and angular velocities.

EKF Parameters

The following are the parameters used in the EKF:

Measurement Noise Covariance Matrix:

$$\begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 0.01 \end{bmatrix}$$

This parameter is set based on the injected noise, with standard deviation of 10 cm (0.1m), and thus, variance of 0.01 m.

Process Noise Covariance Matrix:

$$\begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 0.01 \end{bmatrix}$$

This parameter should be tuned, so as a starting point, this is set to the equivalent of the measurement noise covariance matrix.

State Covariance Matrix, Σ:

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

Since there is a large uncertainty about the system, this is initially set to represent the worst conditions, i.e. a covariance of 1.

Timestep, *dt*:

The timestep is set based on the time between the EKF function calls; when both position and odometer data is available, the timestep is set to the temporal difference in function calls. In this manner, the timestep is specifically dynamic, and does not require an assumption of data transmission times, though this creates additional processing requirements.

State Estimation Results

The following results are based on an EKF-based state estimation, performed simultaneously with mapping using the IPS:

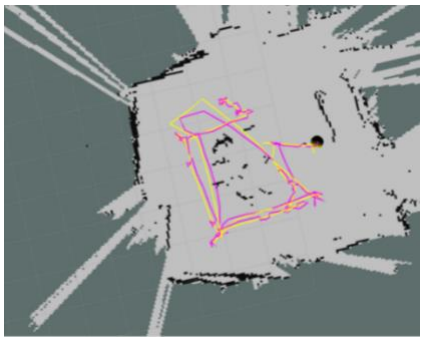

Figure 8 - State Estimation Results; Magenta: Estimated Position; Yellow: True Position

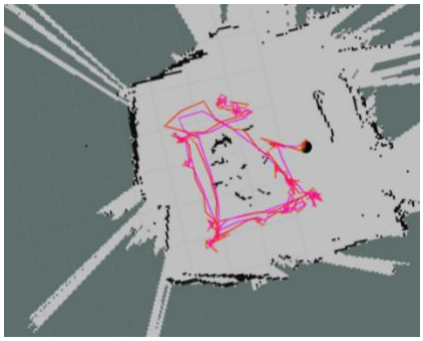

Figure 9 - State Estimation Results; Magenta: Estimated Position; Red: Degraded Position

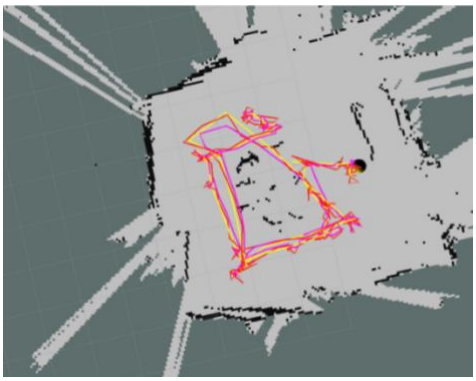

Figure 10 - State Estimation Results; Red: Degraded Position; Magenta: Estimated Position; Yellow: True Position

As seen in the results, the estimated results track slightly closer to the true position than the degraded position. Particularly, where the EKF truly performs well is in its ability to filter the jitter in the degraded position data (Figure 10). Note that the covariance ellipses are shown in the figures, at the end of each path. They are calculated at a 95% confidence interval, and appears almost circular, indicating little covariance in the data.

These results are obtained using a process noise covariance matrix of

$$\begin{bmatrix} 0.05 & 0 & 0 \\ 0 & 0.05 & 0 \\ 0 & 0 & 0.1 \end{bmatrix}$$

Which was obtained by iterating over a range of values between the original matrix and the identity matrix (which represents significant noise).

Simultaneous Localization and Mapping

Simultaneous Localization and Mapping (SLAM) was attempted as a bonus to this lab. This involved combining the two earlier stages of the lab without using the IPS data. The map can be started based on an assumed initial position. After driving and using the EKF to update the predicted state, another scan is performed. Using the Iterative Closest Point (ICP) algorithm, the difference between the map and the scan can be used to obtain a transformation matrix. This matrix is used to align the map to scan data, and in so doing provide a position measurement for the EKF. Implementation of this algorithm has not yet worked in code, but is a matter of interest for further investigation.

Recommendations & Conclusion

Both mapping and EKF-based state estimation have been performed with sufficiently accurate results. The execution of these tasks in Python has been computationally expensive; consequences are observed in the mapping results of Figure 5 vs. Figure 7. It is recommended that the code be written in a more performant language, such as C++, and run in a more powerful environment outside a Virtual Machine.