EBS 221 Final Report

Benoit Rouchy and Kevin Oghalai

June 13, 2025

1 Introduction

This project aimed to simulate an Ackerman steered mobile robot in Matlab with automated field traversal for the purpose of orchard tree diameter mapping. The information it had available to guide itself was odometry, GPS, and compass data. To map the orchard, it had a Lidar scanner which could provide range data with a 180° field of view and 0.125° resolution. Five rows of up to seven simulated trees each were created to simulate an orchard. The trees were two meters apart along each row and each row was three meters away from the other rows. Tree diameters were randomly chosen to be between 20 cm and 50 cm, with missing trees also being possible. To generate the path needed to traverse the orchard, the full set of available paths was created, weighted by travel distance, then optimized to create a quick route, based on the turning radius of the vehicle and widths of each row. The robot ended up being able to efficiently map the field with average errors in the tree diameter, x position, and y position being 2.9 cm, 2.0 cm, and 2.3 cm respectively.

2 Covariance Matrices

The covariance of the system was measured by comparing true values of the robot location to the values measured through the various sensors. The covariance matrix was then calculated according to Formula 1.

$$cov(X, X) = E[(X - E([X])(x - E[X])^T] \tag{1}$$

For the GPS measurements, the coordinates of the robot was randomly chosen, at which point the GPS and compass sensors were used to find the measured global location and orientation of the robot. Then, this was subtracted from the actual measurements to find the difference. This led to the following covariance matrix in Equation 2:

$$cov_{GPS} = \begin{bmatrix} cov_1 & cov_2 & cov_3 \\ cov_4 & cov_5 & cov_6 \\ cov_7 & cov_8 & cov_9 \end{bmatrix} = 10^{-3} * \begin{bmatrix} 0.9017 & 0.0085 & 0.0029 \\ 0.0085 & 0.9193 & 0.0066 \\ 0.0029 & 0.0066 & 0.3936 \end{bmatrix} \tag{2}$$

When doing the same for the odometry, the steering angle was randomly set to be a value from the maximum to the minimum possible, while the velocity was kept constant at 1

m/s. These were similar to the expected use case, as the steering angles frequently reach the maximum possible and the velocity of the robot was kept at a constant 1 m/s throughout all simulations. The measured travel distance and angle change measurements were compared to the true travel distance and angle changes after 100 steps of odometry were carried out. This is becase each GPS measurement occurred every second, while each simulation step was carried out ever 0.01 seconds, leading to 100 odometry steps being combined together for every GPS measurement. This led to the following covariance matrix in Equation 3.

$$cov_{odo} = \begin{bmatrix} cov_1 & cov_2 \\ cov_3 & cov_4 \end{bmatrix} = \begin{bmatrix} 12.0409 & -0.4204 \\ -0.4204 & 11.6099 \end{bmatrix} \tag{3}$$

For both sensors, the trials were carried out 10,000 times in order to ensure that they captured the randomness of the data. Shown in 1 and 2 are plots of the changes in covariance of entries in the matrices above after various numbers of trials.

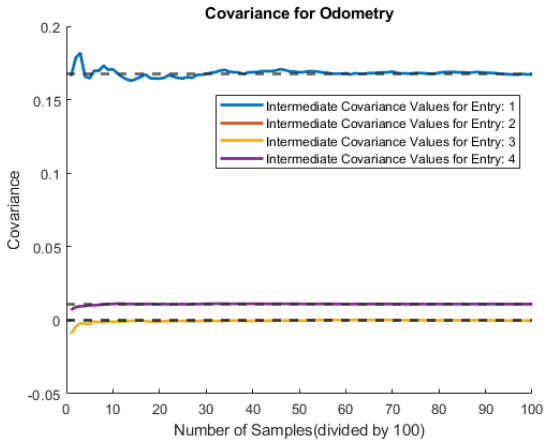

Figure 1: Covariance for odometry sensor.

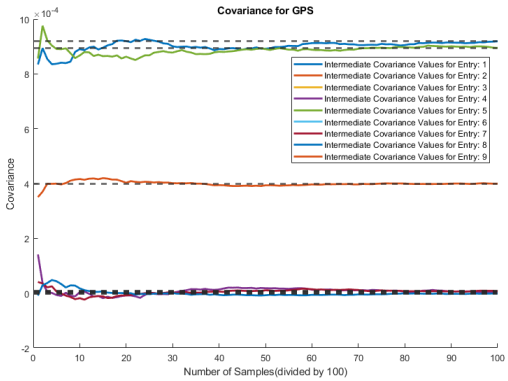

Figure 2: Covariance for GPS and compass.

The values converge after many trials, showing that our measured covariance truly did represent the randomness of the data. These covariance matrices were later used for estimating the robot position using an Extended Kalman Filter(EKF).

3 Extended Kalman Filter(EKF)

This step took the dynamics of the robotic system and attempted to estimate the position of the robot, even through the noisy measurement data. Joseph form of the EKF was used, as shown in the following equations. Z represents the measured pose from the GPS and compass, while F_q ,F_v ,H_w , and H_v are the partial derivatives of the state transition matrix and measurement matrix with respect to the q, v, and w, which are the state, noise, and disturbance respectively. A diagonal process noise matrix was added at each step to eliminate rounding issues with the P matrix.

$$P = F_q * P * F_q' + F_v * cov_{dist} * F_v' \tag{4}$$

$$\nu = z - H * Q_{estimate} \tag{5}$$

$$S = H_q * P * H_q' + H_w * cov_{sensor} * H_w' \tag{6}$$

$$K = P * H_q'/S \tag{7}$$

$$Q_{estimate} = Q_{estimate} + K * \nu \tag{8}$$

$$P = (I - K * H_q) * P * (I - K * H_q)' + K * cov_{sensor} * K' + Q_{process} \tag{9}$$

The odometry covariance ended up being multiple orders of magnitude larger than the GPS and compass covariance, which resulted in the robot being unable to localize itself accurately at almost all timesteps. The exception was when a new GPS and compass measurement were received every second, at which point the estimate position became extremely close to the measured position.

4 Laser Beam Grid

The `updateLaserBeamGrid` function is based on the `updateLaserBeamBitmap` given to us. Rather than setting pixels to 1 or 0 based on a single Lidar measurement, odds were used to capture the noisy, probabilistic nature of the data. If the Lidar returned a range measurement, the pixel at that range would be classified as occupied, while all pixels between the robot and that point would be unoccupied. If occupied, the odds of a pixel were multiplied by $P_{Occ} = P_{high}/(1 - P_{high})$, where $P_{high} = 0.8$. If a pixel was unoccupied, it was multiplied by $P_{Unocc} = (1 - P_{high})/P_{high}$. To prevent this value from going too high or low, limits on the odds of each pixel were set. As the map updates, unoccupied pixels have their probability decrease to nearly zero, while filled pixels become nearly 1. To deal with noise in the sensor, a median filter was used. Matlab's inbuilt function `medfilt1` was used for this purpose.

5 Simulation loop

The simulation loop uses a path slicer to cut up the full path into a chunk for the robot to follow. This prevents the robot from jumping ahead and following the wrong path. Next, a pure pursuit controller was used to generate movement commands. Every 0.01 seconds, the robot was moved according to its dynamics and noisy odometry was then used to update the estimated state of the robot. Every 1 sec the GPS position was measured and the Extended Kalman Filter updated the estimated state. Using the true pose, the noisy laser scanner runs, then using the estimated pose the laser grid is updated. The Lidar was only used once a second. Although the Lidar can run faster, it is only used to update the bitmap directly after the GPS signal, as this is when the estimated pose is the most accurate. Lastly, the robot checks if it has reached the end of the path, in which case the simulation ends. To test the simulation, the file `FP_Main`, provided in the submission, can be used.

6 Finding Trees

While visualization toolbox in Matlab has a function called `imfindcircles`, this function has trouble if the scanner grid is noisy. To produce more reliable results a new function was created: `[XC, YC, RadiusC] = findTree(x_approx, y_approx, r_check, Xmax, Ymax, R, C)`. This function takes the approximate position of where a tree should be and creates a bounding box around that point. The function will only check in this area for points that are occupied. Using the `nndist` function, which finds the distance of the nearest neighbor, points that are not near any other points are removed and assumed to be noise. This smaller collection of occupied pixels will be sent to the `enclosingCircle` function that finds the smallest circle that can enclose all the points. This circle posistion and radius is converted from pixel index back to XY index.

6.1 Generating Output file

The output file was generated using Matlab's `fprintf` function to write the results to an output text file, making sure to fit the proper format.

7 Results

After completing all described steps, the robot was able to successfully traverse the orchard and measure tree diameters. Shown in Figure 4 is a plot of the simulation results with various important features. It shows the desired path, GPS measurements along the path, estimated position along the path, Lidar data of the trees, and the estimated tree diameters based on the Lidar data. For comparison, a plot of the true orchard is shown in Figure 4.

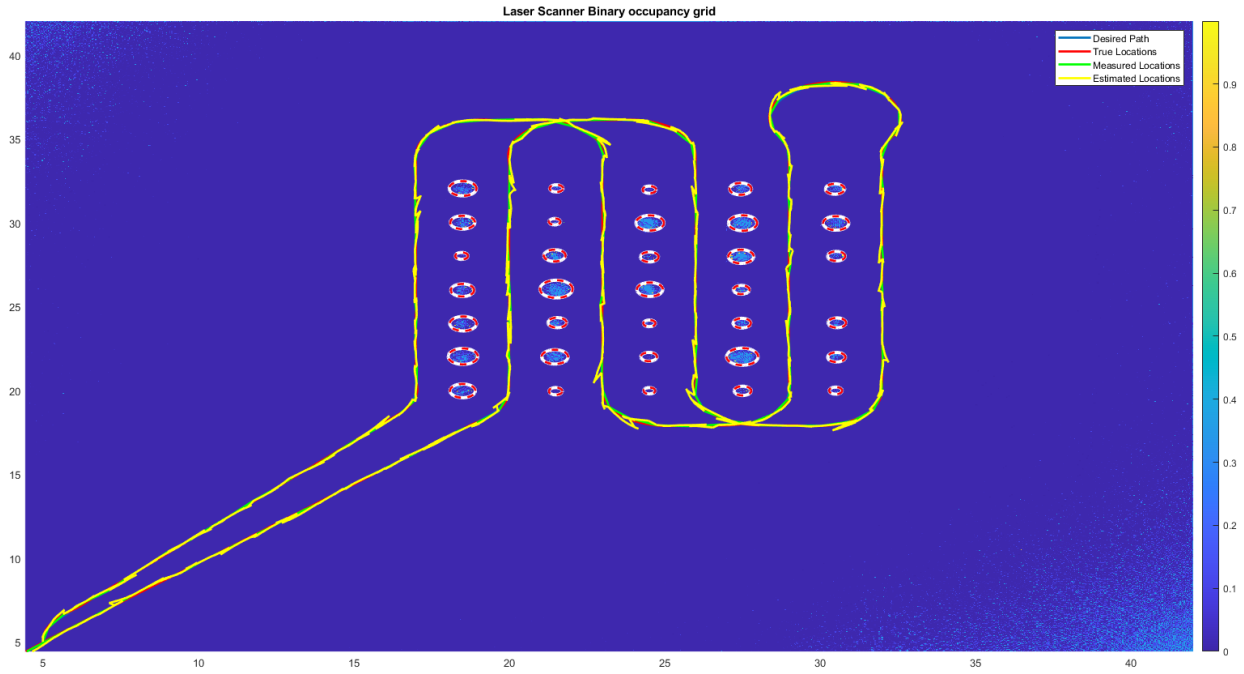

Figure 3: Simulated Orchard(measured).

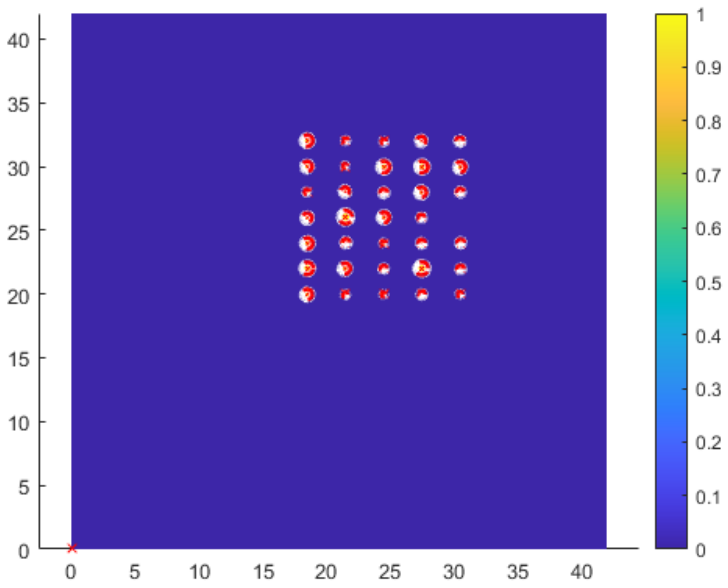

Figure 4: Simulated Orchard(actual).

The robot follows the desired path fairly closely and does end up mapping the tree diameters well. It correctly distinguishes trees that are missing and does seem to capture the sizes of the trunks. A zoomed in section of the path is shown in Figure 5.

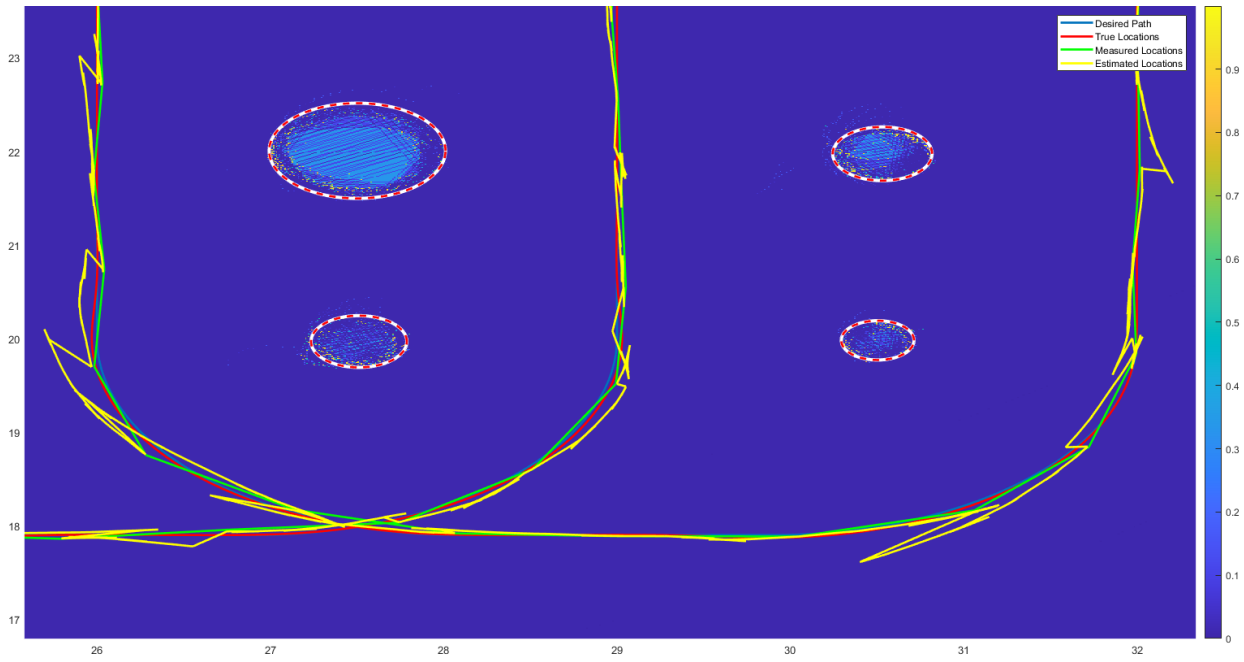

Figure 5: Section of path. Blue shows the desired path, red shows the true locations, green shows the measured locations, and yellow shows the estimated locations.

Proper EKF behavior is observed here. The robot's estimated location deviates from the the actual location when the GPS and compass sensor isn't available, before coming closer to the correct measurement after each sensor reading. The robot is able to track the path fairly well, although the laser scans do suffer from the incorrect estimated locations. In an ideal system, the trees would show up as fully filled circles of the same color, as the Lidar would never be able to get measurements there. Instead, the estimation inaccuracies cause some of the trees to become only partly filled. Although the measured bitmap seems similar to the actual one visually, to quantify the accuracy of the robotic system, various measures are shown in Figures 6, 7, and 8.

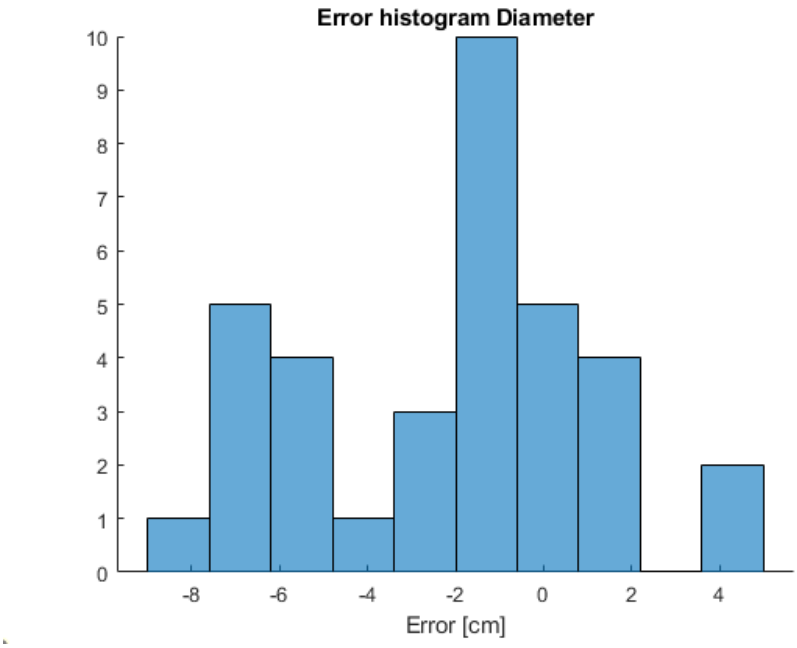

Figure 6: Error histogram of measured tree diameters.

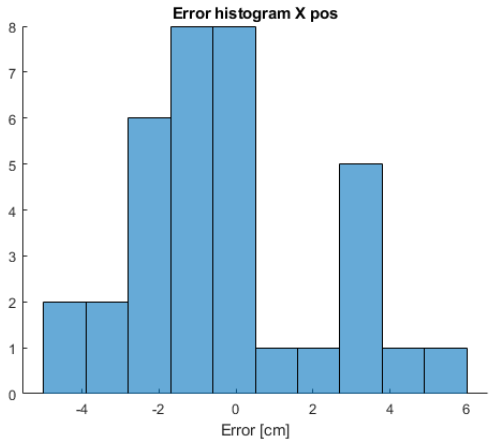

Figure 7: Error histogram of tree x location.

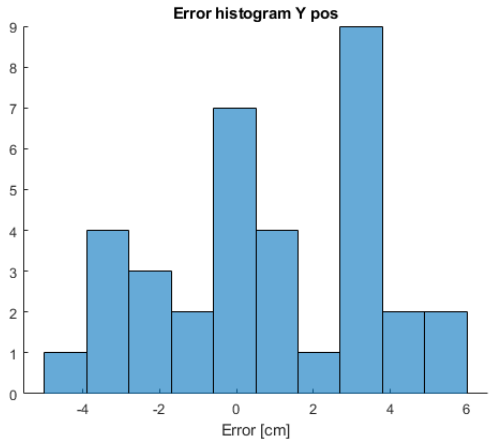

Figure 8: Error histogram of tree y location.

Based on these errors, some statistics were gathered in Table 1.

Table 1: Error Statistics [cm]

	Mean	Std	RMS	Min	Max	95th Percentile
x	1.98	1.47	2.45	0	5.76	4.80
y	2.29	1.58	2.77	0	5.44	5.25
Diameter	2.93	2.56	3.86	0	8.58	7.30

The errors in position ended up being fairly low, topping out at 6 cm, but the errors in diameter were as high as 8 cm here. The results of the simulation show that the code can be reasonably accurate. The simulation results did tend to be inconsistent, the robotic system can theoretically map the orchard and provide reasonable measurements for the tree diameters and locations. The noisy sensors and poor odometry did not lend themselves well to accurate measurements, but there are potential steps that could be taken in the future to fix this. For example, the robot could approach each tree more closely to map its diameter and location, then saving the results. The robot was able to provide good mapping when it was near trees, but tended to overwrite them when it was further away, mapping other parts of the orchard. By saving the results and preventing them from being overwritten, better bitmaps could be created. Additionally, vision based localization could be used to better track the location of the robot in between the GPS and compass measurements. While this might not directly improve the bitmap, it would allow for the Lidar to be used even when there was not a GPS measurement in that timestep, as the robot pose uncertainty wouldn't increase so fast. This would allow for quicker movement, which could reduce the time needed to complete the task.