CSE 276A HW3

Tingyu Shi(A59023729)
Jiajun Li(A16635772)

December 26, 2024

1 Video URL

- Drive Square Trajectory 1 Time: https://youtu.be/XtGfgcn-BM4
- Drive Square Trajectory 2 Times: https://youtu.be/CeTvbCNL6RY
- Drive Octagon Trajectory 1 Time: https://youtu.be/KUkwJv53jok
- Drive Octagon Trajectory 2 Times: https://youtu.be/-mqgJ0ypuGE

2 Groud Truth Map

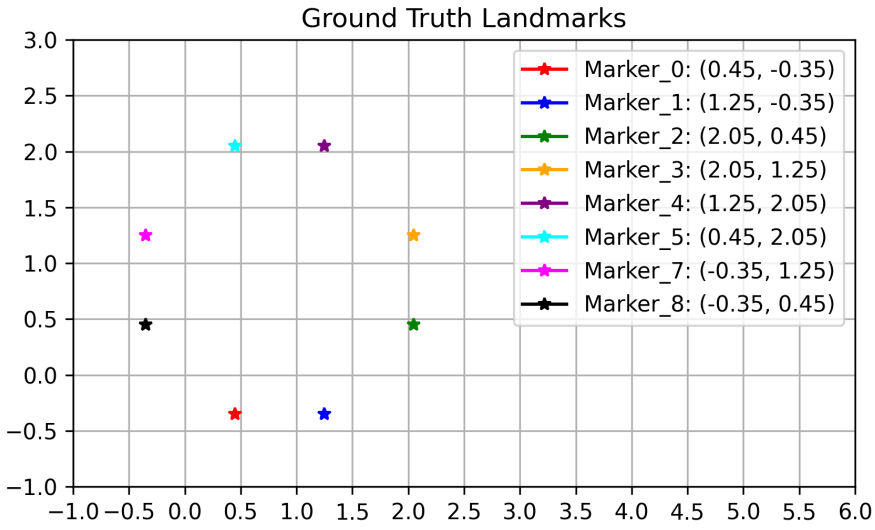


Figure 1: Ground Truth Landmarks

3 Algorithm

For the EKF-SLAM algorithm, we referred to the following two resources:

- Probabilistic Robotics book, section 7.6.1 for EKF-SLAM algorithm.
- EKF-SLAM simulation from PythonRobotics.

3.1 State Vector x and Measurement Vector z

For the car itself, the state vector is

$$x = (x, y, \theta)^T$$

which represents the car's position and orientation in the world coordinate system. For the whole system, the state vector

$$s = (x, y, \theta, x_{lm1}, y_{lm1} \cdots x_{lmN}, y_{lmN})^T$$

represents the car's state and positions of N landmarks. The measurement vector

$$z = (q, \theta')^T$$

Here, q means the distance between the car and the landmark, and θ' means the angle of the vector from the car to the landmark in the camera coordinate system.

3.2 System Matrix F

Without the control command, the state will not change, so the system matrix is the Identity matrix:

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

3.3 Control Matrix G

Our control signal u is the following:

$$u = \begin{bmatrix} v_x \\ v_y \\ w_z \end{bmatrix}$$

The control matrix G is the following:

$$G = \begin{bmatrix} dt * cos(\theta) & -dt * sin(\theta) & 0 \\ dt * sin(\theta) & dt * cos(\theta) & 0 \\ 0 & 0 & dt \end{bmatrix}$$

Since we use EKF, we also need the Jacobian matrix of G, which is the following:

$$G_{jacobian} = \begin{bmatrix} 0 & 0 & -dt * v_x * sin(\theta) - dt * v_y * cos(\theta) \\ 0 & 0 & dt * v_x * cos(\theta) - dt * v_y * sin(\theta) \\ 0 & 0 & 0 \end{bmatrix}$$

3.4 Measurement Matrix H

Since we use EKF, we only used the Jacobian matrix of H, which is the following:

$$H_{jacobian} = \frac{1}{q}\begin{bmatrix} -\sqrt{q} * \delta_x & -\sqrt{q} * \delta_y & 0 & \sqrt{q} * \delta_x & \sqrt{q} * \delta_y \\ \delta_y & -\delta_x & -q & -\delta_y & \delta_x \end{bmatrix}$$

- q represents the square of the distance between the car and the landmark
- δ_x represents the x coordinate difference between car's position and landmark's position
- δ_y represents the y coordinate difference between car's position and landmark's position

3.5 $x_{0|0}$ and $\Sigma_{0|0}$

We initialized $x_{0|0}$ and $\Sigma_{0|0}$ as the following:

$$x_{0|0} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\Sigma_{0|0} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3.6 Determine System Noise and Measurement Noise

To estimate the system noise R_T, we conduct several measurements to determine the variance in the robot's motion along x, y, and θ when commanded with a single twist. Specifically, we run multiple experiments for each type of motion x, y, and θ and compute the variances σ_x^2, σ_y^2, and σ_θ^2 from the resulting data. However, the system noise increases significantly when these three motions are combined, particularly when the robot is using a PID controller. Therefore, we choose better values for the noise matrix when combining the controller. The variances obtained are then scaled using appropriate coefficients to enhance the accuracy of localization and mapping. The resulting system noise matrix is:

$$R_T = \begin{pmatrix} 0.251 & 0 & 0 \\ 0 & 0.248 & 0 \\ 0 & 0 & 0.216 \end{pmatrix}$$

For the measurement noise Q_T, we place one tag one meter in front of the robot. The robot keeps measuring the distance and angle from the robot to the tag multiple times, and then we calculate the variance of the distance and angle. The variances obtained are scaled with a large coefficient due to increased uncertainty when the robot is in motion. The final measurement noise is:

$$Q_T = \begin{pmatrix} 0.247 & 0 \\ 0 & 0.136 \end{pmatrix}$$

3.7 How to handle landmark detection

- If a new landmark is detected, we first estimate its position according to the current car's position estimation and camera reading. Assume the current car's position is $x = [x, y, \theta]^T$ and the observation is $z = [q, \theta']$, then the new landmark's estimated position is

$$\begin{bmatrix} x + cos(\theta + \theta') \\ y + sin(\theta + \theta') \end{bmatrix}$$

  We also initialize the covariance matrix to be the identity matrix. Then, we insert the landmark's estimated position and the landmark's covariance matrix into the system's state vector and the system's covariance matrix.

- If a landmark is not detected, then we will not update its position and variance at this iteration.

- If a landmark is detected, then we will update its position and variance at this iteration according to the observation vector.

4 Result

4.1 Experiments Setup

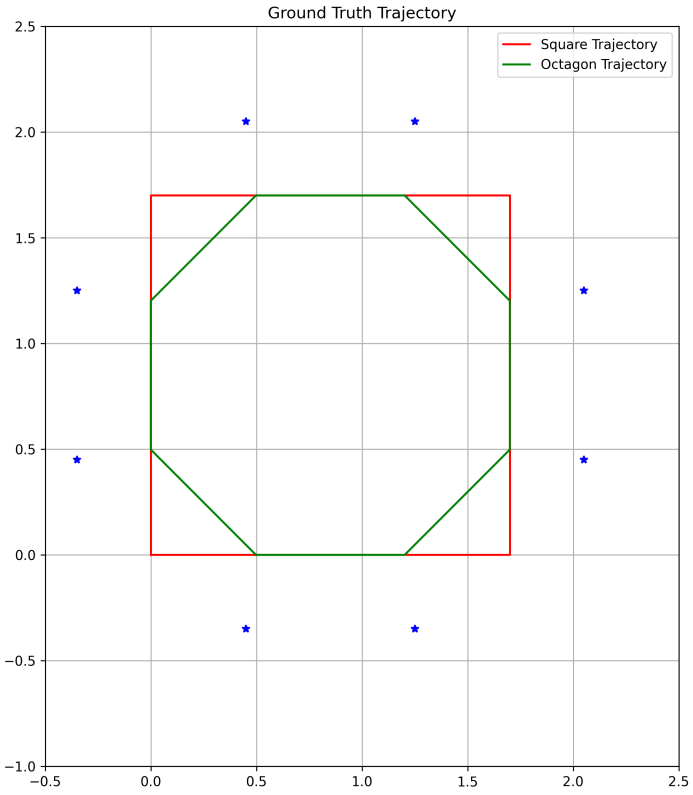
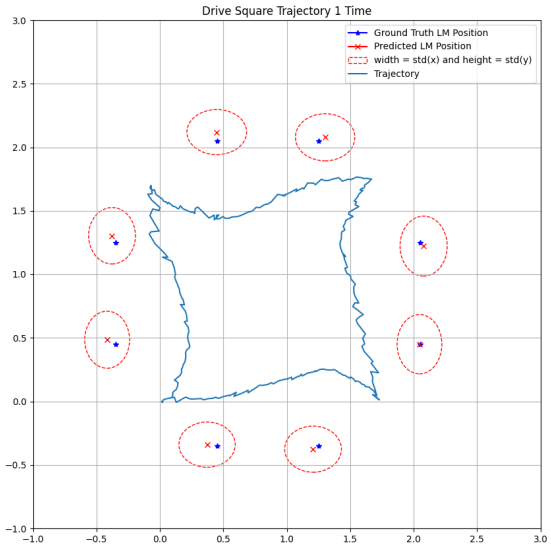


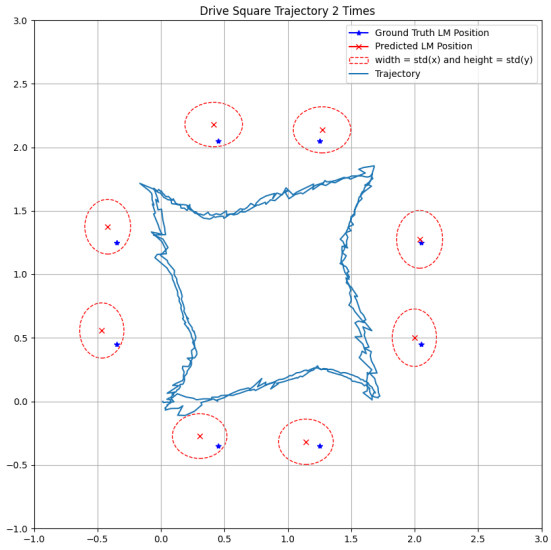
Figure 2: Ground Truth Trajectory

Figure 2 shows the planned square/octagon trajectory. The square trajectory has a side length of 1.7m, and the octagon trajectory is a regular octagon within the square. We designed trajectories this way so the robot can have a broader field of view. This means that the camera can capture at least one landmark for the most of the time.

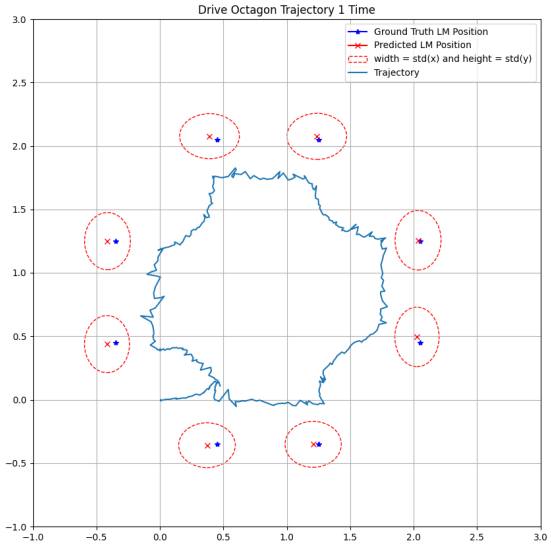
4.2 Diagram



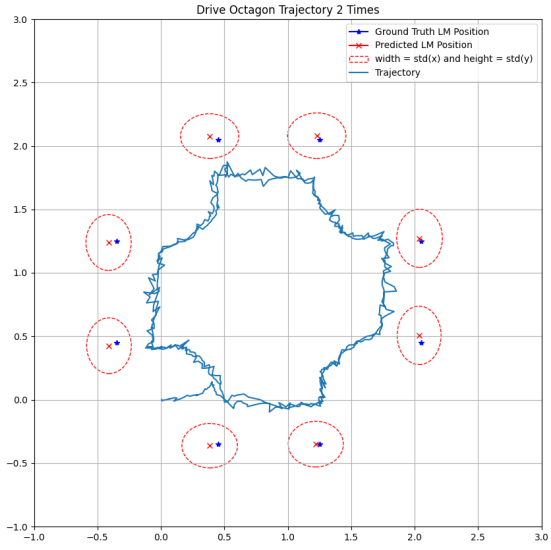
(a) Drive Square Trajectory 1 Time



(b) Drive Square Trajectory 2 Times



(c) Drive Octagon Trajectory 1 Time



(d) Drive Octagon Trajectory 2 Times

Figure 3: Estimated Landmarks and Trajectory

Figure 3 shows the estimated landmarks' positions and trajectory for driving square/octagon trajectory 1/2 times. The blue star represents the ground truth landmark position. The red $\times$ represents estimated landmark position. The red ellipse represents the standard deviation of the landmark position. The horizontal semi-axis represents $std(x)$ and the vertical semi-axis represents $std(y)$. The blue line represents the actual trajectory.

4.3 Numerical Results

Landmark ID	0	1	2	3	4	5	7	8	Average
Error	0.0785	0.0499	0.0036	0.0405	0.0598	0.0694	0.0598	0.0745	0.0545
$std(x)$	0.4434	0.4469	0.3531	0.3717	0.4682	0.4705	0.3693	0.3541	0.4096
$std(y)$	0.3566	0.3634	0.4669	0.4717	0.3713	0.3553	0.444	0.4488	0.4097

Table 1: Numerical Results of Driving Square Trajectory 1 Time

Landmark ID	0	1	2	3	4	5	7	8	Average
Error	0.1635	0.1114	0.0725	0.0271	0.0893	0.1332	0.1424	0.1581	0.1122
$std(x)$	0.4299	0.4314	0.3487	0.3637	0.4525	0.4535	0.361	0.3493	0.3987
$std(y)$	0.352	0.3576	0.4517	0.4537	0.3626	0.3502	0.4303	0.4343	0.3990

Table 2: Numerical Results of Driving Square Trajectory 2 Times

Landmark ID	0	1	2	3	4	5	7	8	Average
Error	0.0790	0.0418	0.0503	0.0162	0.0282	0.0646	0.0631	0.0681	0.0514
$std(x)$	0.4458	0.4435	0.3488	0.3623	0.4695	0.4705	0.3616	0.3553	0.4072
$std(y)$	0.3535	0.3625	0.4698	0.4691	0.3632	0.3549	0.4506	0.4498	0.4092

Table 3: Numerical Results of Driving Octagon Trajectory 1 Time

Landmark ID	0	1	2	3	4	5	7	8	Average
Error	0.0666	0.0288	0.0582	0.0249	0.0372	0.0711	0.0631	0.0640	0.0517
$std(x)$	0.4379	0.4346	0.3462	0.3603	0.4597	0.4593	0.3567	0.3523	0.4009
$std(y)$	0.3502	0.3601	0.4604	0.4593	0.3591	0.3521	0.4405	0.4385	0.4025

Table 4: Numerical Results of Driving Octagon Trajectory 2 Times

Table 1, 2, 3, 4 shows the numerical results of 4 experiment settings. The units are all meters.

4.4 Comments

From the above numerical results, we drew the following conclusions:

- When driving 2 different types of trajectories for the same amount of times, the octagon trajectory can always produce a smaller average error. When driving square/octagon trajectory 1 time, octagon trajectory can reduce the average error by 0.0031m. When driving square/octagon trajectory 2 times, octagon trajectory can reduce the average error by 0.0605m.
- When driving the square trajectory 2 times, the average error increases 0.0577m compared with driving square trajectory 1 time. When driving the octagon trajectory 2 times, the average error is almost the same as driving the octagon trajectory 1 time. Therefore, we think the octagon trajectory is more robust than the square trajectory.

5 Reference

- Probabilistic Robotics book, section 7.6.1 for EKF-SLAM algorithm.
- EKF-SLAM simulation from PythonRobotics.