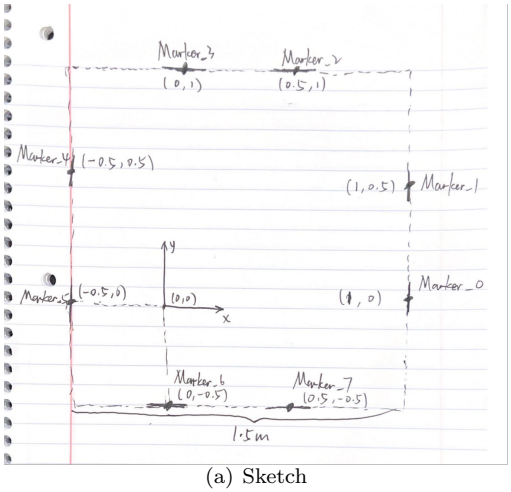
CSE 276A Homework 3 Report

Haoyuan Tian PID: A59024246

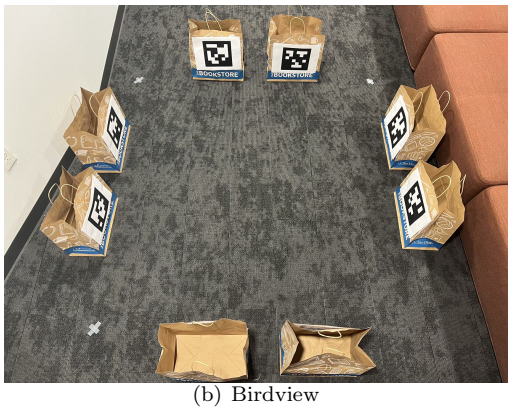
November 15, 2023

1 Environment

The environment is described in sketch and birdview in Fig.1. Due to the space limitation, we can only set up an $1.5m \times 1.5m$ square environment. We use AprilTags with id $0-7$ as the eight landmarks and placed them evenly along the square.



(a) Sketch



(b) Birdview

Figure 1: Environment Overview

2 Algorithm

For this task, we implemented the EKF-SLAM algorithm. The state vector is defined as

$$s_t = [x_r, y_r, \theta_r, x_1, y_1, \ldots, x_n, y_n]^T$$

where x_r, y_r, θ_r denotes the pose of the robot and x_i, y_i denotes the position of landmark i. All elements in s_t are in world coordinate. Typically, the algorithm follows the steps below: (The algorithm largely follows the lecture so some formula in the steps may be omitted).

Step 0: Initialization

At the very beginning, $s_t = [0, 0, 0]^T$ and its covariance matrix Σ is an diagonal matrix with elements on the diagonal being 0.2. This value is large enough to the noise we add since we will multiply the noise vector with speed / distance in the implementation.

Step 1: Prediction

In the prediction step, we predict the state vector $s_{t|t-1}$ from $s_{t-1|t-1}$ by introducing the control signal u_t.

$$s_{t|t-1} = \mathbf{F}s_{t-1|t-1} + \mathbf{G}u_t$$

In our implementation, since we are in a relatively steady system, we define $\mathbf{F}$ as identity matrix of size $len(s) \times len(s)$. In our implementation, the u_t comes directly from the PID controller and is in world coordinate, $u_t = [v_X, v_Y, \omega]^T$, thus the $\mathbf{G}$ is simply 3×3 diagonal matrix with non-zero elements being timestep d_t. Meanwhile, we have to add the system noise Q to $\Sigma_{t-1|t-1}$. In our implementation, the Q_i is related to the speed of the robot and the timestep with the base variance predefined based on the calibration of the robot. (With higher speed or longer timestep we expect to see larger noise).

Step 2: Updating

In the updating step, we use previously detected landmarks (AprilTags), to correct the prediction $s_{t|t-1}$:

$$s_{t|t} = s_{t|t-1} + \mathbf{K}_i(z_t - \mathbf{H}z)$$

In order to update the orientation of the robot, instead of constructing $z_t = [dx, dy]$ directly from the AprilTag detection, we transform the tf between camera and marker frame to the range-bearing expression following Reference. Specifically, $z_t = [d, \rho]$. d is the distance measured between robot and landmark. To get ρ, we first transform the tf $[dz, -dx]$ in camera coordinate to world coordinate $[dx, dy]$ using θ_r and $\rho = atan2(dy, dx) - \theta_r$. z is also in range-bearing expression which is computed using $[dx, dy] = [x_i - x_r, y_i - y_r]$. $\mathbf{H}$ is calculated as the jacobian of the measurement function. Notice that for each marker previously detected and still can been seen, we update s and Σ one by one to simply the construction of $\mathbf{H}$. When adding measurement noise R_i, we take into consideration the distance d between robot and landmark i, the further the two objetcs are, the larger the measurement noise is.

Step 3: Add new landmarks

For landmarks detected the first time, we won't use them in Step 2 to update s and Σ. Instead, we expand s and Σ to represent the position and the covariance of the landmarks. We maintain two sets and a dictionary in the code to track newly detected and previously detected landmarks and the index of the landmark in s. Specifically, we won't expand s to $2n+3$ if landmark with id n is detected, instead we just expand the length of s by two and use the dict to track the landmark index. (This does little difference for our environment setting, but it should work well with random set of landmarks).

Different type of landmarks

The situation of a landmark going out of the field of view, and a previously detected landmark reappearing has been covered in the above steps. To be short, we use the sets and the dict described in Step 3 to check if the landmark is newly detected and if the landmark is seen at the current frame. For the disappeared landmarks, we will not update its position in Step 2.

3 Results

3.1 Square Motion

For the square motion, we chose a $0.5m \times 0.5m$ square. The reason for the choice is that we would want the robot to be as close as to the landmarks to get a more accurate measurement, but not too close where it can't see the full landmark. Another reason is that since we have a relatively small environment to do the experiment, a larger side length may cause unexpected problems. The trajectory and the estimated map

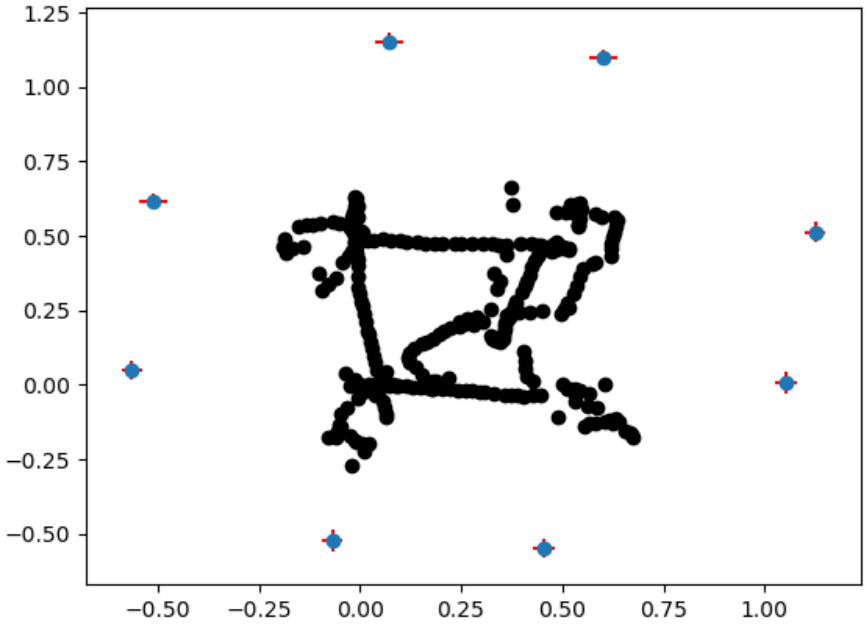


Figure 2: Square Motion

from our square motion is shown in Fig.2. From the result, we can tell that it's hard to make the robot go exactly what we expect it to go in a small environment. However, our estimated map is surprisingly close to the ground-truth map with mean absolute error being (0.0689, 0.0725), which indicates that our algorithm can do SLAM effectively. We tried run the square motion multiple times but the result doesn't improve much given that the error is already low. We witnessed that the car moved more smoothly after 3 laps of square motion, which means the estimated map is helping correct the robot motion more and more.

3.2 Octagon trajectory

For the octagon motion, we chose a side length of $\frac{1}{1+\sqrt{2}}m$. The reason is similar as above. We want the robot to try to get close to the landmarks but not too close. And the $1.5m \times 1.5m$ environment makes larger length choice impossible. The trajectory and the estimated map from our octagon motion is shown in Fig.3. The result is frustrating which means using this kind of small side length is not reasonable, but there were no time for better calibrating the noise and the measurement. We believe the reason for the failure lies in that the robot won't see a landmark for a continuous time since it's always rotating to the next point as shown in the trajectory, which makes the measurement unreliable. The mean absolute error for our octagon error is (0.107, 0.153). We've tried runing the motion for mutiple times and the results improved by approximately 20%. We suppose that a well-developed octagon motion would perform better than the square motion since it would have more time to update landmarks such as $Marker_3$ in our environment.

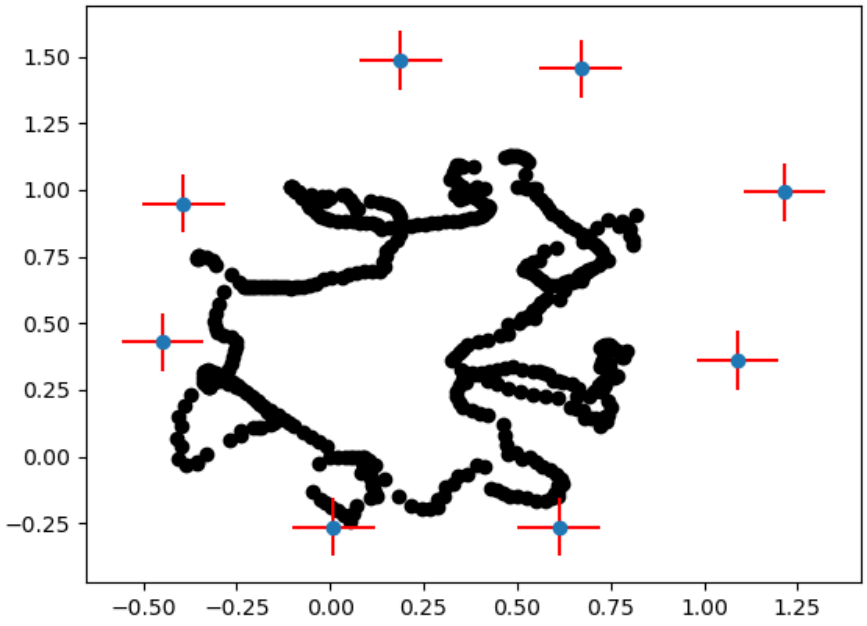


Figure 3: Octagon Motion

3.3 Discussion

Video Link: LINK
Although we've achieved low error between the estimated map and the ground-truth map, there remains several problems to deal with:

1. The trajectory and performance of the robot varies for the same setting, making the SLAM process unpredictable and somewhat unreliable (The robot will sometimes 'stuck' in a particular position for a little while which can be seen from the path it took in the video).

2. The calculation of the measurement vector z_t relies on θ_r, which is an estimated value before updating. This cause larger uncertainty for correcting the orientation of the robot.

3. During experiment, we found the performance of the robot is vulnerable to the R_i and Q_i, a slight change in the two noise measurements will cause a huge difference. And the initialzation of Σ is also important.

4. The space really limited the performance of the robot as we assume that going for a small distance using SLAM is harder than going a longer distance.

Overall, we believe we have successfully implemented the EKF-SLAM, however, we lack time to do proper parameter tuning and experiment. We'll try to improve the performance in the future.