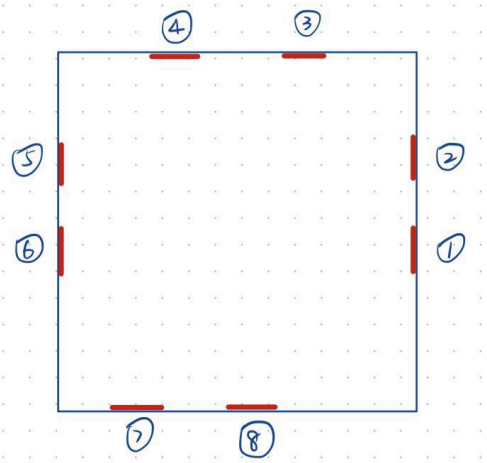
ROBOTICS : 276A HW3 REPORT

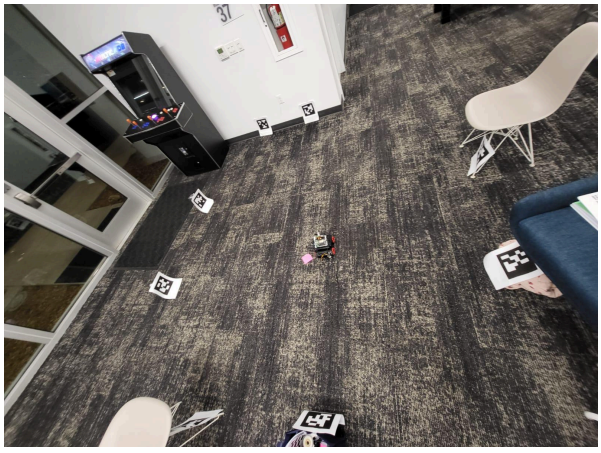
PID1: A69034019 Name: Anandhini Rajendran

PID2: A69033245 Name: Yueqi Wu

Video link for both square and octagon:
https://www.youtube.com/watch?v=htXXl6STyFo

Setting up the environment:





The image on the left is a sketch of our ground-truth map in the top-down view layout. The image on the right is a photo of our landmark settings.

We set the environment in approx 10ft x 10ft area. Let the coordinate of center of the area be (0, 0), then the eight apriltags are at the positions listed in the following table.

AXIS:
X axis : Points towards landmarks 1,2
Y axis: Points towards landmarks 3,4
Origin: starting point of the robot

# apriltag	April tag id	X coordinate	Y coordinate
1	3	2.5	0
2	2	2.5	1
3	4	1	1.5
4	17	-0.3	1.5
5	0	-1.3	0.6
6	16	-1.2	0.2
7	15	-0.3	-1.2
8	1	0.1	-1.2

Algorithm (EKF Based Kalman Filter)

1. Logic

During the driving, we do the following steps every 0.8 seconds. (Do the kalman filter every 0.8 sec and update state).

   a. Twist_callback - predict step
      i. Get the current velocity of the robot (v_x, v_y, omega) from the PID controller. v_x,v_y,omega are scaled up to match the real world - experimented with the scales to get close enough values.
      ii. Multiply the velocity with dt to get the predicted change in the location of the robot (dx, dy, dtheta).
      iii. Use the predicted change in the location to update the robot state in the state vector and the corresponding covariance.
   b. Apriltag_callback - update step
      i. This step is done only if any apriltag is detected. Detailed in the "Handling cases when no April tag is detected" part later.
      ii. For every apriltag in the camera's view, use the robot's current state to convert the tag-to-camera location to tag-to-world location,getting the location of landmarks in the world frame.
      iii. Expand the state vector and covariance vector if any new apriltag is detected.
      iv. Use the landmarks in the view of the robot to update the state vector and covariance vector.

```
# Measurement update
    z_pred = np.array([[math.sqrt(q)], [math.atan2(dy, dx) -  state[2, 0]]])
    S = H @  covariance @ H.T +  R
    K =  covariance @ H.T @ np.linalg.inv(S)
    y = measurement - z_pred
    y[1, 0] =  normalize_angle(y[1, 0])  # Normalize angle difference
    state += K @ y
    covariance = (np.eye(len( state)) - K @ H) @  covariance
```

   c. Publish_estimated_pose
      i. Update the current state in the PID controller using the current state of the robot in the state vector.
      ii. We used the weighted mean of the current state and published state to update the current state owing to error values. E.g. gave more weights to x and lesser to y,w.

2. Variables
   a. State vector : [x,y,w, lx_0,ly_0,lx_1,ly_1…lx_n,ly_n]. The state vector contains the state of the robot and coordinates of landmarks in the world frame. We initialize the state to be (0, 0, 0). We don't have any information about the landmarks in advance, including the number of landmarks. We keep adding landmarks to the state as we run through the waypoints using thresholding to obtain the right number of landmarks.
   b. Measurement vector z: is obtained from april tag detection vector : (z,x) the transformed to world frame for further calculations.
   c. F: system matrix: is identity matrix of size: len(state)*len(state).
   d. G: control matrix: G is a 3 *3 diagonal matrix with non-zero elements as dt.

e. Predict step using F and G.
The v_x,v_y,omega comes from pid controller(scaled up to match the real world)
dx = (v_x * math.cos(theta) - v_y * math.sin(theta)) * dt
dy = (v_x * math.sin(theta) + v_y * math.cos(theta)) * dt
dtheta = omega * dt
# Update the robot's state
state[0, 0] += dx
state[1, 0] += dy
state[2, 0] += dtheta
state[2,0] = normalize_angle(state[2,0])

f. H: measurement matrix
H = np.zeros((2, len(state)))
H[0, 0] = -dx / math.sqrt(q)
H[0, 1] = -dy / math.sqrt(q)
H[1, 0] = dy / q
H[1, 1] = -dx / q
H[0, idx] = dx / math.sqrt(q)
H[1, idx] = -dy / q

g. Covariance initialization
We initialize the covariance of the state to be 0.1 on diag and 0 off diag. When landmarks are added to the state, we initialize 1000 on diag and 0 off diag. For the covariance initialization when adding landmarks, we also tried values 100,500,800,50. The square trajectory sometimes benefitted from being initialized to 100 but the octagon gives very bad accuracy, so we use 1000 for both in the end.

h. System noise and measurement noise: (Q&R)
   i. We tried various combinations of Q, and R to arrive at a good estimate. The logic was to use higher values for R in the y coordinate, owing to the bad y estimate from the April tag.
   ii. We also noticed that increasing just one coordinate error makes the algorithm unstable and gives bad values. Hence we tried scaling the whole Q, R after estimating a good ratio between the X, and Y coordinates.
   iii. The graphs for multiple runs and varied Q,R, covariance initializations can be found in this document: https://docs.google.com/document/d/1nPTDsfUdxOX5swU_ll0JcWFpqadgQ4E_qtZVQY0cnXU/edit?tab=t.0

3. Handling Landmarks:
When a landmark is detected, we calculate its Euclidean norm with every landmark in the state vector to check if it is already present in the state vector. If the Euclidean norm is less than a threshold, then we consider this landmark as detected for the first time. Otherwise, we consider this landmark to be previously detected.

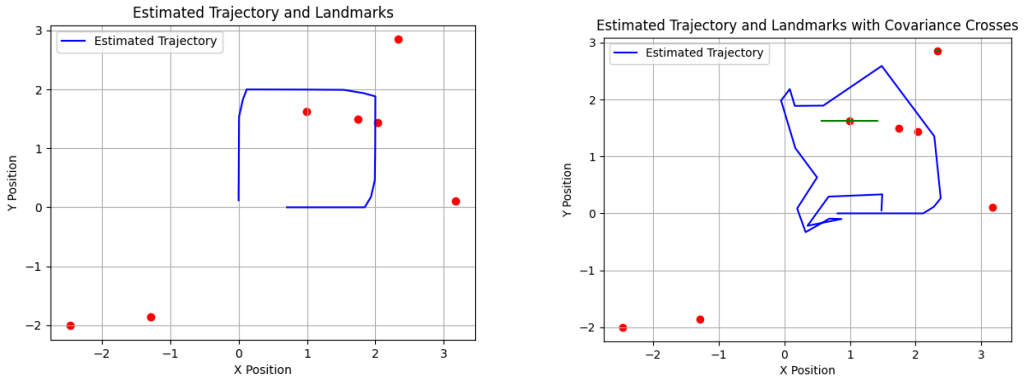
   a. Choosing threshold: We use 1 for octagon and 1.6 for square.
   We experimented with 0.5, 1, 1.6, 2, and 2.5. If the threshold is too small, the tolerance for error in detected location of the apriltags is too small, then one apriltag will be counted for multiple times. If the threshold is too big, two apriltags nearby will be considered as one. So we chose 1 for octagon and 1.6 for square to get correct landmarks.

b. When a landmark is detected for the first time, we add the detected location of the landmark to the state vector and extend the covariance. Then we use the location of this landmark to update the state vector and covariance.
c. If the landmark is previously detected, we fetch its location from the state vector, and update the state vector and covariance. We use the get_landmark_index function to get the index if the landmark is already present in the update step.
d. For disappearing landmarks i.e. going out of field of view we don't do the update.

4. Handling cases when no April tag is detected:
We use a flag variable to mark if there is any landmark detected. If there is any landmark detected, we do apriltag_callback. Otherwise, we do just the predict step and the robot continues to move to the next waypoint.

Results

1. square motion
   a. Plot of the trajectory estimated by the algorithm along with the map of landmarks

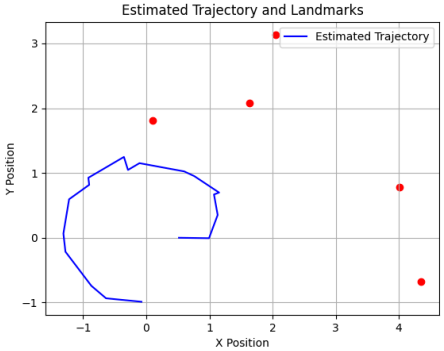


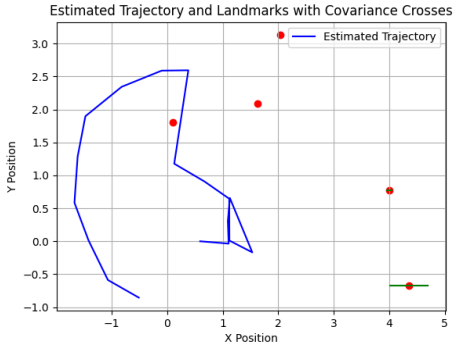
In the left plot, the trajectory generated doesn't use the state from the kalman filter to update the current state in the PID controller i.e. if we don't do the kalman the robot will move like this. The second plot is when we do the update and plot trajectory based on the published state.
Most of the covariances are too small hence not plotted in the graph. Please find the values below:
State: [[1.0487015], [-0.31588752], [0.08678586], [0.99207092], [1.63040955], [2.33453788], [2.8488358], [2.03008265], [1.44203491], [1.74985695], [1.50001144], [3.17674592], [0.10325159], [-1.28260458], [-1.85302937], [-2.46601758], [-2.00941113]]
Covariances:
[0.2096 -0.0076 -0.0055 0.1785 0 0.1795 0 0.1873 0 0.1806 0 0.195 0 0.1898 0 0.1898 0]
[-0.0076 0.0321 0.0192 -0.0007 0 0.0017 0 0.0043 0 -0.0012 0 -0.0028 0 0.0022 0 0.0025 0]
[-0.0055 0.0192 0.0431 -0.0006 0 -0.0006 0 0.0015 0 -0.0018 0 -0.0001 0 0.0089 0 0.01 0]

[0.1785 -0.0007 -0.0006 0.1821 0 0.1737 0 0.1763 0 0.1763 0 0.1783 0 0.1772 0 0.1771 0]
[0 0 0 0 1000 0 0 0 0 0 0 0 0 0 0 0]
[0.1795 0.0017 -0.0006 0.1737 0 0.2278 0 0.1789 0 0.1777 0 0.1782 0 0.1758 0 0.1756 0]
[0 0 0 0 0 1000 0 0 0 0 0 0 0 0 0 0]
[0.1873 0.0043 0.0015 0.1763 0 0.1789 0 0.2277 0 0.1853 0 0.1847 0 0.1819 0 0.1814 0]
[0 0 0 0 0 0 1000 0 0 0 0 0 0 0 0 0]
[0.1806 -0.0012 -0.0018 0.1763 0 0.1777 0 0.1853 0 0.2765 0 0.1809 0 0.1763 0 0.1759 0]
[0 0 0 0 0 0 0 1000 0 0 0 0 0 0 0 0]
[0.195 -0.0028 -0.0001 0.1783 0 0.1782 0 0.1847 0 0.1809 0 0.2031 0 0.193 0 0.1932 0]
[0 0 0 0 0 0 0 0 1000 0 0 0 0 0 0 0]
[0.1898 0.0022 0.0089 0.1772 0 0.1758 0 0.1819 0 0.1763 0 0.193 0 0.3973 0 0.2308 0]
[0 0 0 0 0 0 0 0 0 1000 0 0 0 0 0 0]
[0.1898 0.0025 0.01 0.1771 0 0.1756 0 0.1814 0 0.1759 0 0.1932 0 0.2308 0 0.5351 0]
[0 0 0 0 0 0 0 0 0 0 0 1000 0 0 0 0]

b. Average error of landmarks
Mean Absolute Error (MAE): 1.2839

2. 8-point motion
   a. Plot of the trajectory estimated by the algorithm along with the map of landmarks





In the left plot, the trajectory generated doesn't use the state from the kalman filter to update the current state in the PID controller i.e. if we don't do the kalman the robot will move like this. The second plot is when we do the update and plot trajectory based on the published state.

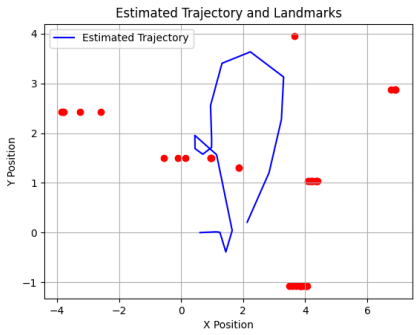
Most of the covariances are too small hence not plotted in the graph. Please find the values below:

State: [[-0.51196781], [-0.85530622], [-1.35962586], [4.35595641], [-0.67346633], [4.00711227], [0.77902347], [1.63209416], [2.08627069], [0.10411131], [1.80626829], [2.04369176], [3.12904192]]

Covariance:

[[0.9726, -0.0748, 0.2485, 0.1184, 0, 0.1268, 0, 0.1487, 0, 0.1595, 0, 0.1394, 0],
 [-0.0748, 0.2606, -0.0227, 0.0056, 0, -0.0012, 0, -0.0003, 0, 0.0733, 0, 0.0967, 0],
 [0.2485, -0.0227, 0.1260, -0.0039, 0, 0.0017, 0, 0.0042, 0, -0.0412, 0, -0.0562, 0],

[0.1184, 0.0056, -0.0039, 0.1329, 0, 0.1143, 0, 0.1158, 0, 0.1306, 0, 0.1322, 0],
[0, 0, 0, 0, 1000, 0, 0, 0, 0, 0, 0, 0, 0],
[0.1268, -0.0012, 0.0017, 0.1143, 0, 0.1599, 0, 0.1294, 0, 0.1214, 0, 0.1206, 0],
[0, 0, 0, 0, 0, 0, 1000, 0, 0, 0, 0, 0, 0],
[0.1487, -0.0003, 0.0042, 0.1158, 0, 0.1294, 0, 0.2203, 0, 0.1359, 0, 0.1338, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 1000, 0, 0, 0, 0],
[0.1595, 0.0733, -0.0412, 0.1306, 0, 0.1214, 0, 0.1359, 0, 0.3493, 0, 0.3058, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1000, 0, 0],
[0.1394, 0.0967, -0.0562, 0.1322, 0, 0.1206, 0, 0.1338, 0, 0.3058, 0, 0.3722, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1000]]



This graphs is when you run with a lesser threshold for judging whether a landmark is previously detected or not. There are only 5 landmarks in the above graph due to high threshold. We did some experiments and couldn't find the exact threshold which shows 8 landmarks,

b. Average error of landmarks
Mean Absolute Error (MAE): 1.43
The average error of 8-point motion is larger than the square motion because the robot has more rotation and movements which leads to bigger error in estimating the state.

Comment on the results

1. Choice of side length
For the square motion, we chose a side length of 2 meters. For the 8-point motion, we chose a side length of 0.75 meters. We made the choice based on following considerations:
    a. If we set the side length too small, there is less time for the SLAM and detecting landmarks. Also, if the robot drives far away from the April tags, the accuracy of detection decreases.
    b. If we set the side length too big, the landmark is very close to the robot and the April tags are not detected. Also, there is an issue of not seeing both the tags and just focusing on one.
    c. Hence, we took medium-sized side lengths, driving through almost 70 per cent of the grid to give proper accuracy.
2. Map after driving multiple times
    a. Driving multiple times gives different results but mostly similar trajectories and close enough accuracy for a square and for octagon it improves better.

    b. Changing the starting point of the robot affects the accuracy a lot, implying that the detections of the April tag affect the prediction accuracy.
3. Comparison of two motions/maps
   The square motion of the robot is more accurate compared to the 8-point motion, since there are fewer rotations. So the estimation of the robot state is more accurate in square motion, which results in a better trajectory of the robot in the map and more accurate locations of the landmarks. The difficulty in the 8-point motion also leads to the situation where the robot can't detect all 8 landmarks sometimes, so some landmarks are missing.
4. Improvements and discussion
    a. Changes in starting location, distances affect SLAM's accuracy a lot which makes it unstable and unreliable.
    b. The performance of SLAM varies a lot depending on choice for Q,R and covariance initialization.
    c. We can not do SLAM for a very small area since it depends on april tag detections and time, as seen by comparing accuracies of square and octagon trajectory.