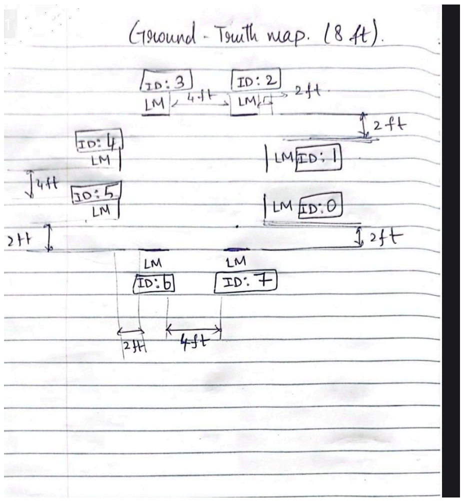
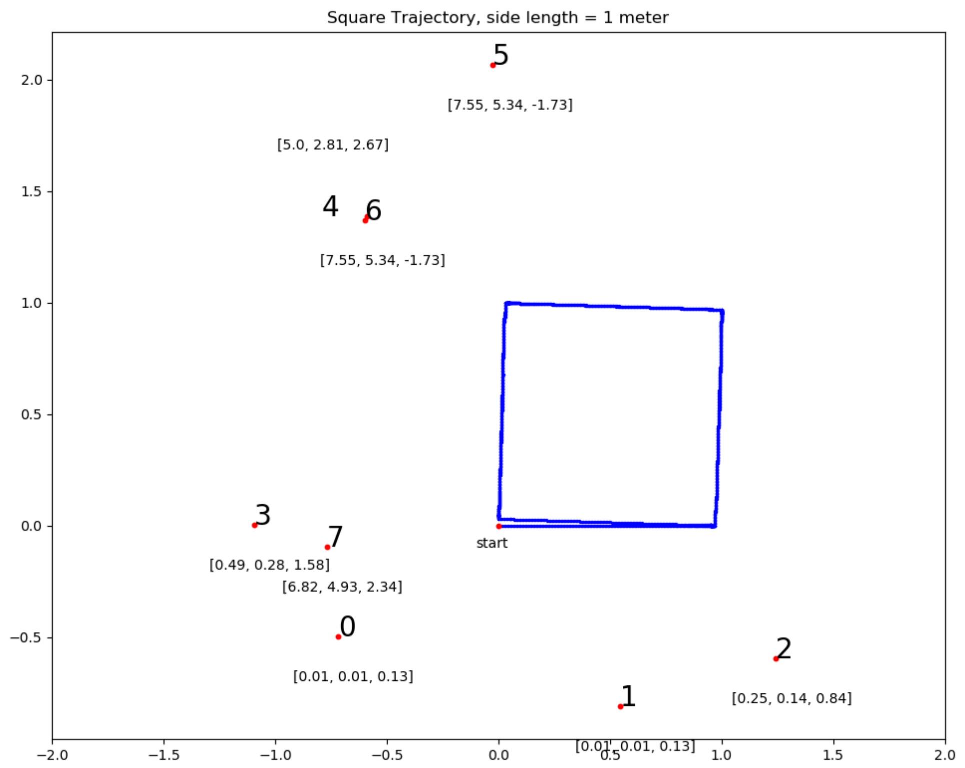
CS276A - Introduction to Robotics
Homework 3

Raghul Shreeram and Rami Altai

1) Ground Truth Map:

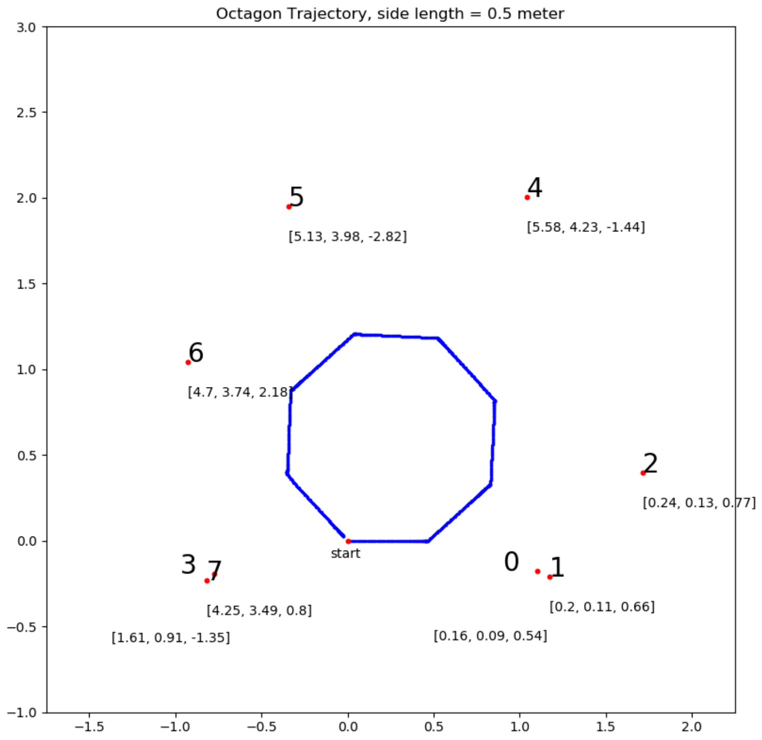


5) ii)The results obtained with the square motion:



Average error of landmarks: [1.154875, 0.5304] (x, y)

iii) Results obtained with 8-point motion:



Average error of landmarks: [0.541, 0.389] (x,y)

The error does seem to significantly decrease with multiple runs, as does the covariances. The error of the octagon is also significantly better. We theorize this is due to the many different angles we see each marker at.

Note: Unfortunately, we have a bug in our code that we weren't able to resolve where specifically the measured orientation value in our state changes our covariances and causes some of them to increase over time.

We have tested our Kalman filter on straight motions and marker locations are much more precise and covariance decreases over time with new sensor estimations, as expected.

6)

i) State Vector (x): Our state vector contains the robot's position and landmark positions in world coordinates. For both the robot and landmarks, we save the state as a 3D vector [x, y, theta], with theta being the angle around the z axis.

   Measurement vector (z): The measurement vector is constructed from the AprilTag detections, and is the distance from the robot to tag in the robot coordinate frame. The measurement is also a 3D vector [x, y, theta].

ii) System Matrix (F): The system matrix F is an identity matrix because when the robot doesn't move, the world coordinate positions of the robot and tags (our state) should not change. An identity matrix means the state at time t should be the same as a state at time t-1

   Control Matrix (G): The control matrix G is a diagonal matrix for the first three rows with elements [delta_t, delta_t, delta_t], with delta_t being our update timestep in our PID controller. As our control vector u represents our velocities [v_x, v_y, omega], G*u will provide the estimated position change using our motion model on the Kalman filter prediction step. As our control vector is not related to our waypoints, the matrix does not provide an update to waypoint state values, and the rest of the rows are 0's.

   H matrix: This matrix is a composition of two transforms. The first calculates the relative distance and orientation between the robot and the detected tags (and only detected tags) in the world coordinate frame. This simply involves equations of the form (tag estimation - robot

position){world frame} = tag relative distance {world frame}.
The second transform is a rotation to get the same relative values in the robot coordinate frame. This allows us to calculate the innovation (z - Hs) in the same coordinate frame.

iii) Initial State (x0|0): The initial position is set as [0,0,0] with respect to the world frame.

Covariance (Σ0|0): To initialize the covariance, we only set the diagonal, as we expect the covariance between many of our state variables (robot vs tag) to be negligible, and it is also hard to measure. For the diagonal values pertaining to the robot, we set them all to 0 initially as we define the robot's initial position as [0, 0, 0] w.r.t the world frame. Therefore, its initial location is perfectly precise. For the tag diagonals, we decided to multiply our error Q values by a scalar such that our initial variances were quite high ([2.5, 2.5, 100 degrees]). This was because as the first detection, the pose estimation could be quite wrong and so we have little confidence in its initial value.

iv) System Noise (Q) and Measurement Noise (R):
For both Q and R, we made the assumption that they were diagonal matrices. This was because there should be practically no relation between many of the errors, e.g. the error of the robot's x location v.s tag 1's rotation, and these errors are negligible and hard to measure.

To find the system noise (Q), we used our previous experience with HW1 and HW2 to estimate the error components for x, y, and orientation.

To determine the measurement noise (R), we placed the robot and tag at various distances and angles from each other, and found the mean error between each component of the april tag module's predicted position v.s the actual position using measuring tape.

v) Handling Landmarks

a) New landmark Detection: When a new landmark is detected for the first time, we add its estimated position to our state vector (keeping the state vector values in order of tag number) as well as expand the covariance matrix, initializing the diagonal values that pertain to the new tag.

b) Landmark Out of Field of View: If a landmark goes out of the field of view and is no longer detected, it will not contribute to the measurement vector during the update step, despite still being included in the state vector and covariance matrix. The Kalman filter handles this gracefully by adjusting the size of the H and R matrices based on the available detections at that time step. If there are no landmark detections at all, the Kalman filter will only perform a predict step, and no update step.

c) Previously Detected Landmark Reappearing: If a landmark reappears after going out of the field of view, it will again contribute to the measurement vector during the update step, and the H and R matrices will be resized accordingly. As the state vector always contains the tag states in order of tag number, the relevant indices to include in the H matrix is always known as we obtain tag number upon detection.

Video Links:

1) Square Trajectory: https://youtu.be/jQaoZjon0Ro
2) Octagon trajectory: https://youtu.be/BzJJ7DH41hw