Homework 6

1 SETUP

a. The state dynamics equation is the method we use to convert our controls into the robot position. Our inputs are the robot's speed V and angular velocity ω. We use these to find the distance and angle traveled using the finite time difference and assume $d = V \times \Delta t$ and $\phi = \omega \times \Delta t$

$$r = \frac{d}{\phi} \tag{1}$$

with the assumption that θ is not equal to 0. Not, we solve for the length ℓ by dividing the arc into two equal triangles with length b and height .

$$\ell = 2 \times r \times \cos\frac{\phi}{2} \tag{2}$$

Finally, we can convert this length into the global coordinates to solve for x, y, and θ of the robot:

$$x = x_0 + \ell \times \cos\left(\frac{\phi}{2} + \theta_0\right) \tag{3}$$

$$y = y_0 + \ell \times \sin\left(\frac{\phi}{2} + \theta_0\right) \tag{4}$$

where θ_0 is the initial robot rotation relative to the inertial frame.

With the case that ϕ is 0:

$$x = x_o + d \times \cos\left(\theta_0\right) \tag{5}$$

$$y = y_0 \tag{6}$$

Finally, the final rotation of the robot, θ_f is

$$\theta_f = \theta_0 + \phi \tag{7}$$

These are the same equations as the program written in HW2 `integradeOdom`.

b. Our measurment function `hFunc` is simply the function `global2Robot` since we want to input the global coordinates of the robot and marker and have the program output the local coordinates of the markers in *robot* frame. This is because the measurement we are given is the location of the markers in robot frame.

c. Since we have no "ground truth" position as the map is unknown, we set the initial position of the robot to be $[x, y, \theta] = [0, 0, 0]$.

2 EKF SLAM

a. The state vector for solving the SLAM problem with EKF is the vector
mu = [X,Y,θ,x1,y1,x2,y2...xn,yn] where xi,yi are the global coordinates of the markers and X,Y,θ are the position and orientation of the robot.

b. We need to linearize the measurement and control equations which end up being the Jacobian H and G respectively in my code.

G is the identity matrix of size N by N where N is the length of mu. Additionally, if phi=0: G(1,3) = -d .* sin(θ);
G(2,3) = d .* cos(θ);

If phi != 0,
G(1,3) = d/phi .* (cos(θ + phi) - cos(θ));
G(2,3) = d/phi .* (sin(θ + phi) - sin(θ));

For H, it is probably easier just to look at my code, but here is my best way to explain it using code, we have the M by N matrix where M is M - 3. the we have a repeating pattern for columns 1 and 2 of [-cos(θ),-sin(θ);sin(θ),-cos(θ)]. For the third column we find:
H(i-3,3) = -sin(θ) .* (markerX - robX) + cos(θ) .* (markerY - robY);
H(i-2,3) = -cos(θ) .* (markerX - robX) - sin(θ) .* (markerY - robY);
Where the marker x and y are the coordinates at i and i+1 as the program continues; All other entries are zeros.

c. I placed the initial state vector at all zeros as we have no way of estimating the markers positions with no information. Additionally we have no global truth pose and thus have to set it's reference frame to be the robot's initial position of [0,0,0].

d. Please see Figures below. I realize there has to be some issue with my code, but I am unable to locate the issue sadly. Overall the trajectory is definitely off what it should be and it is clear that the one wall is extremely unreasonable. I tried a couple checks to see if difference was unreasonable, but they did not fix the issue.

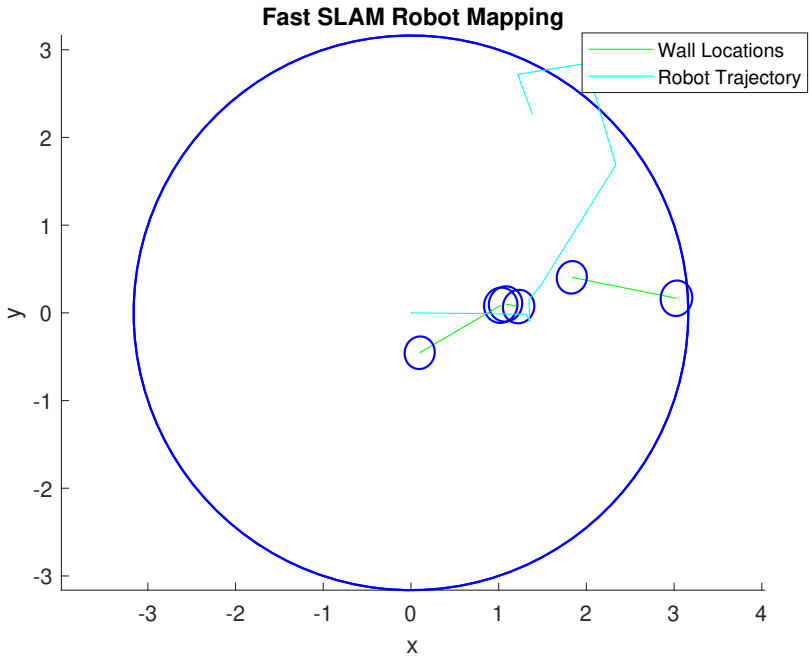

Figure 1: EKF SLAM with 1/3 the Data

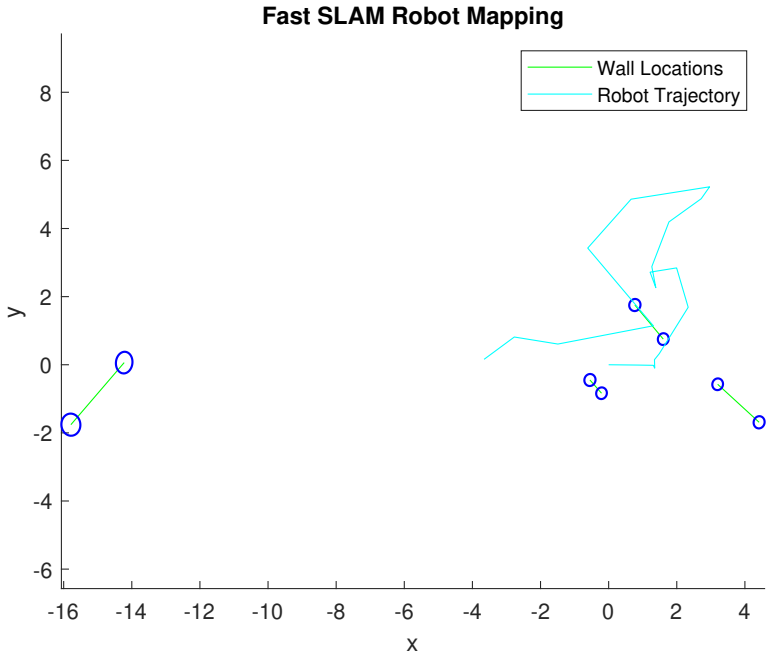

Figure 2: EKF SLAM with 2/3 the Data

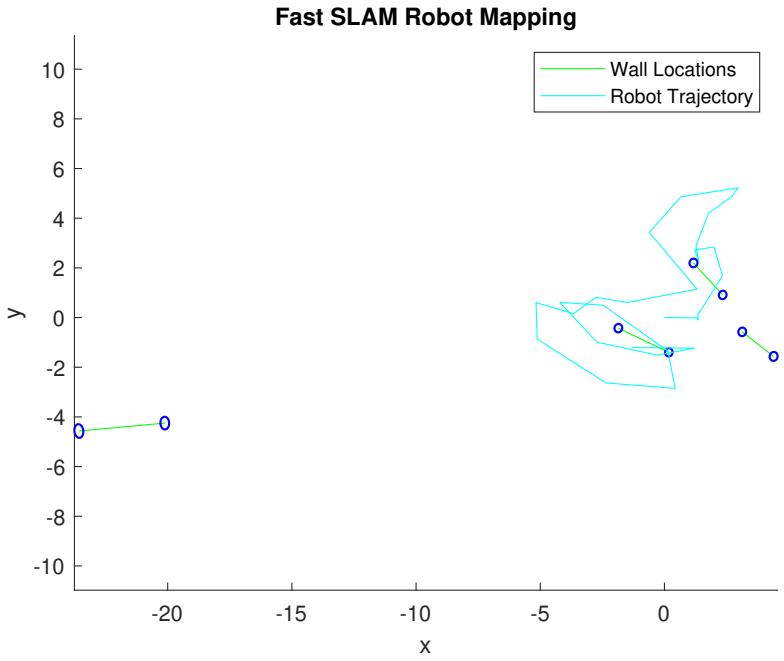

Figure 3: EKF SLAM at Final Step

3 Fast SLAM

a. I choose a particle set size of N=500 to get a large number of particles while keeping computation time to less than two seconds. I created a structure for the particle set which is detailed as follows:

Struct:**PSET**

- x - x location of robot
- y - y location of robot
- θ - θ of robot
- weight - probability of particle being correct
- length - number of markers (based on input matrix
- **markers** - Struct of length `Pset.length`
  - seen - seen =1 if seen before, 0 if not
  - mu - estimated global location of marker if seen
  - sig - covariance matrix of marker location if seen
  - weight - weight of marker if seen

To access the particle set I use syntax pset(i).marker(j).mu to access the i'th particle and the estimated position of marker j.

I initialize all of the particles as follows: $[x, y, \theta] = [0, 0, 0]$, weight = 1/N (N particles), length = number of markers, seen = 0, mu = [], sig = [], weight = [].

b. Completed using Matlab :)

c. I choose the robot position to be the highest weighted particle at each time step and output the array to the main function. Below are figures of three different time steps during the motion of the robot. Overall I am extremely happy with how this program turned out! As seen below, the wall positions in the intermediate steps are not quite correct as the robot passes through the estimated wall, but at the final step the program is able to correct this.

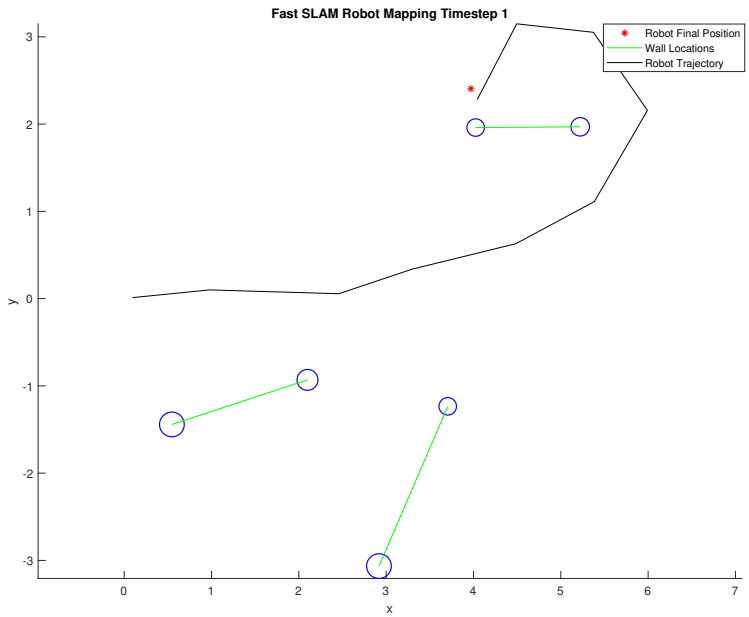

Figure 4: Fast Slam with 1/3 the Data

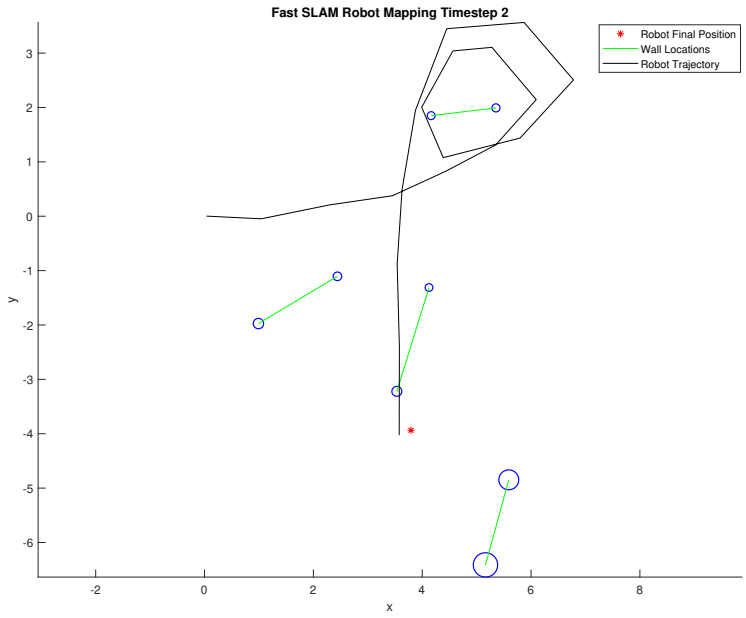

Figure 5: Fast Slam with 2/3 the Data

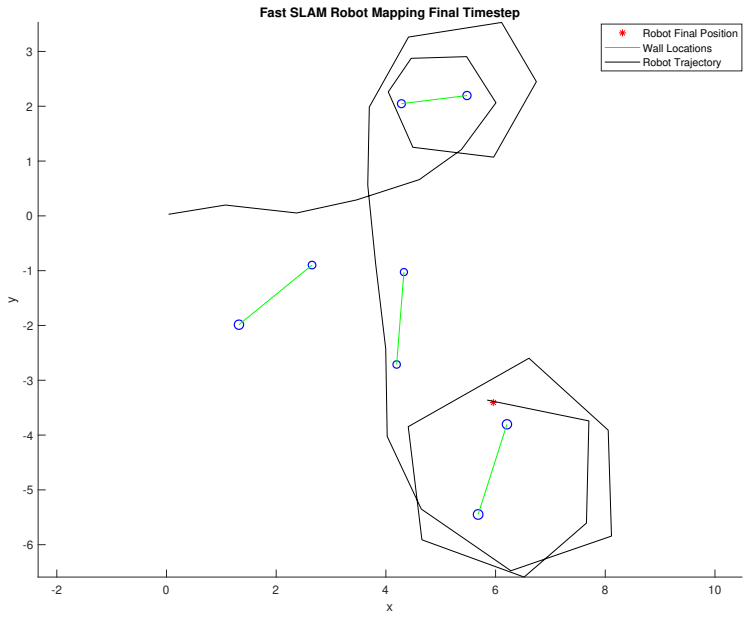

Figure 6: Fast Slam Final Plot

4 Test Function

Please see matlab for my test function