ARW – Autonomous Robot Workshop

Lab 2

Localization

INTRODUCTION

The purpose of this lab is to introduce participants to Particle Filter Localization through a Matlab coding exercise that builds on the simulation created in Lab 1. By the end of this lab, students should understand how particle sets represent a robot's belief state, and how they can be updated with real time measurements. This lab can be done in pairs. **NOTE:** This lab will likely take longer than 1 hour, and should be completed for homework to be ready for day 2 of the workshop.

Codebase

The Matlab code base

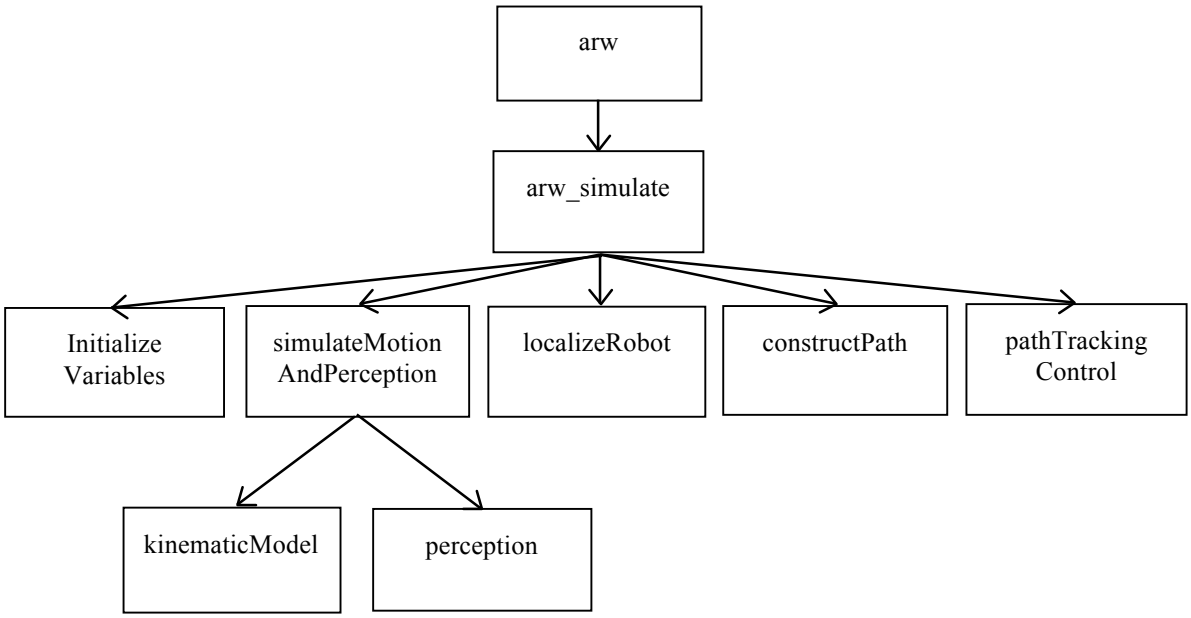

Figure 1: The ARW Matlab codebase.

For Lab 2, we will be only programming in the files named arw_simulate.m and localizeRobot.m. In arw_simulate.m, the function arw_simualate() will be modified to make sure the localizeRobot() function will be called at every time step of the control loop. In localizeRobot.m, there will be two main functions to be modified: localizeRobot() and getWeight(). Within these two functions, (and with a helper function), your Particle Filter will be implemented. These modifications are described in the next section.

3. CONTROL LOOP MODIFICATIONS

Open the file named arw_simulator.m. It should look similar to (but not exactly like) the code shown in Fig. 2. First, note that the first function call to initializeVariables() initializes the variable P. This will be your particle set. Second, note the major difference between this lab and the previous: in Lab 2 we call the localizeRobot() function.

As in Lab 1, many of the other function calls in the control loop are commented. No motion planning is functioning, so we still comment out the call to constructPath(). Instead of setting the control vector (which corresponds to right and left motor speeds) using the pathTrackingControl(), we will hardcode it for the time being.

At this point, modify your arw_simulator() function as shown below in Fig. 2.

```
function [X] = arw_simulator(X_0, X_des)

    % Initialize variables
    [ X_real, X_est, U, pathTracked, timeStep, M, P, T, e, nodes, deltaT ]
                                            = initializeVariables( X_0 );

    % Loop over time, until pathtracking is complete
    maxTimeStep = 100;
    while pathTracked == false && timeStep <= maxTimeStep

        % Simulate the actual robot to get the real state, measured
        % odometry O, and range measurements Z
        [ X_real, O, Z ] = simulateMotionAndPerception( X_real, U, M, deltaT );

        % Localize the robot, i.e. estimate the current state X
        [X_est, P] = localizeRobot(X_est, O, Z, M, P);

        % Construct a new path if it is the first iteration
        %[T, nodes] = constructPath(X_0, X_des, M, T, nodes);

        % Determine the control inputs U to track the path T
        U=[1200 50*timeStep]; T=[0 0 0 1];%[U, T, pathTracked] = pathTrackingControl(X_est, T);

        % Save states and calculate the error e
        X(timeStep,1:3) = X_real;
        e(timeStep) = sqrt((X_real(1)-X_est(1))^2 + (X_real(2)-X_est(2))^2);

        % Plot the output
        plotState(X_real, e, X, Z, M, P, nodes, T);
        timeStep = timeStep + 1;
    end
end
```

Figure 2: Modified arw_simulator.m file for use with Lab 2.

4. LOCALIZEROBOT MODIFICATIONS

Open the file named localizeRobot.m. It should look like the code shown in Fig. 3. In this case much of the code is missing. Given the input values of the previously estimated state X_est_tm1 (which is supposed to represent $\hat{X}_{t-1} = [\hat{x}_{t-1} \hat{y}_{t-1} \hat{\theta}_{t-1}]$ in units of meters and radians), and the recent encoder measurements O = [$\Delta\varphi_R$ $\Delta\varphi_L$] in units of pulses (where one encoder pulse equates to 1/4096 of a revolution), the recent n bearing/range measurements Z = [γ_1 r_1; … ; γ_n r_n], the map M, and the particle set P = [x_1 y_1 θ_1 w_1; … ; x_m y_m θ_m w_m] of m particles, the function must calculate the updated state $\hat{X}_t = [\hat{x}_t \hat{y}_t \hat{\theta}_t]$ based on the updated particle set P.

Use the lecture notes to fill in the core steps of the particle filter, (see Fig. 3). For the *prediction step*, make sure to loop over all particles, using the number of particles equal to size(P,1). The randn() function may be useful here, as well as the kinematic model you wrote in the previous lab. This step should also call the getWeight() function for every particle.

For the *correction step*, you can leverage the pre-existing function called resample(). This function implements the approximate approach to weighted sampling described in lecture.

To calculate the estimated state, use the pre-existing function called getMeanState(). Try to understand how this function works, especially how angle theta is estimated.

```
function [X_est_t, P] = localizeRobot(X_est_tm1, O, Z, M, P)

    % Tseful variables
    sigma_O = 0.5*max(abs(O));
    X_pred_i = zeros(size(X_est_tm1,1),size(X_est_tm1,2));

    % Prediction Step
    % X_pred_i = ...



    % Correction Step
    % P = ...




    % Calculate State
    % X_est_t = ...


end
```

Figure 3: The localizeRobot.m file to be modified in Lab 2.

4. GETWEIGHT MODIFICATIONS

Now scroll to the function called getWeight() as shown in Fig. 4. In this case, you will implement the weight calculation for a single particle. This weight should represent how close the expected range measurements from the particle state X represent the actual range measurements Z. The map M is used with X to calculate the expected range measurements by calling the function getClosestRangeToWalls(). This function, already coded for you, finds the range to the closest wall from a sensor position x, y facing in a direction $theta$. You can find the function in its own file called getClosestRangeToWalls.m.

```
function [w] = getWeight(X, Z, M)

    % A useful parameter
    var_z=0.1;

    % Set the weight
    %w = ...

end
```

Figure 4: The getWeight.m file to be modified in Lab 2.

5. CUSTOMIZATION OF PARAMETERS

To obtain nominal performance, you may leave the parameter values as the default values, (see Fig. 5). However, if you want to solve the difficult unknown start problem, you will need to set the value of unknownStart to be true in the the initializeVariables() function. You will likely need to increase the number of particles to around 250. Unfortunately this will slow your code down considerably.

```
function [ X_real, X_est, U, pathTracked, timeStep, M, P, T, e, nodes, deltaT ] =
initializeVariables( X_0 )
    X_real = X_0;
    X_est = X_0;
    U = [0 0];
    pathTracked = false;
    timeStep=1;
    M = setMap();
    deltaT = 0.1;
    numParticles = 50;
    unknownStart = false;
    [P] = initializeParticles(X_0, M, numParticles, unknownStart);
    T=[];
    e=[];
    nodes=[];
end
```

Figure 5: The initializeVariables.m file to be modified in Lab 2.

6. LOCALIZATION TIME

Test your localization algorithm with different numbers of particles. Watch the plot below the workspace image to track your localization error. See how long it takes to converge (e.g. reduce the error) as a function of the number of particles you use.