ARW – Autonomous Robot Workshop
Lab 1

Simulation

1. INTRODUCTION
The purpose of this lab is to introduce participants to the ARW Matlab codebase and code up the simulation. By the end of the lab, students should understand the codebase architecture, and how to program the robot to conduct future lab experiments. This lab can be done in pairs.

2. CODEBASE
Download the codebase stubs from the website. Inside you will find 12 Matlab files. Most of these files are shown in Fig. 1. Each block corresponds to a single file, although several supporting functions can be found in some of the files. Once participants modify the code as described in this document, participants can run the simulator by simply typing arw at the Matlab prompt. This arw.m script will call the arw_simulator function which is the heart of the code. It houses a control loop that iteratively calls all the other functions.

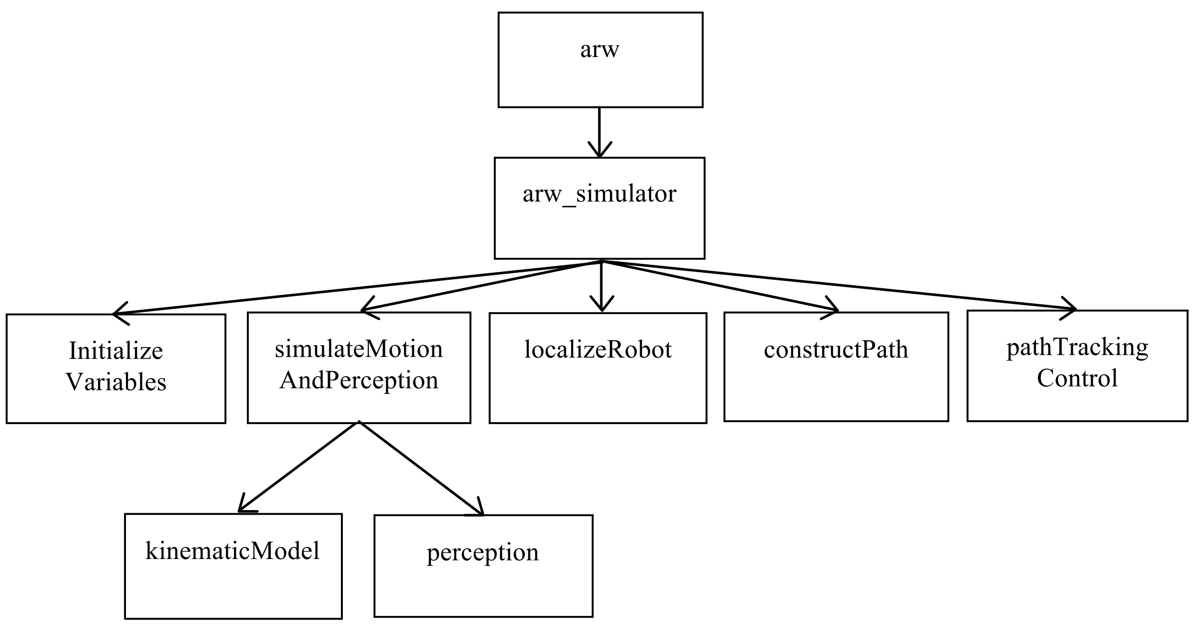

Figure 1: Codebase Architecture

For this lab 1, you will only need to modify two files: arw_simulator.m and kinematicModel.m. Modifying the control loop in arw_simulator() will allow you to run the simulator without calling all the non-working functions that you will complete in following labs. Modifying the kinematicModel.m file will allow the code to simulate the differential drive kinematics from lecture, and visualize them on the screen.

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 your variables, and if you want to change them, you must open the corresponding file, although it is not necessary for Lab 1. Second, note the while loop which acts as the robot's control loop. This loop will run until either a) the pathTrackingControl() function returns the output that it has completed tracking the path, or b) a preset number of iterations has passed.

Within this control loop, you are most interested in the call to the function simulateMotionAndPerception(). This function will subsequently call kinematicModel() which you will be modifying later.

Many of the other function calls in the control loop are commented out for Lab 1. Since your localization hasn't been coded yet, we assume the estimated state X_est can be set to the actual state X_real for Lab 1. No motion planning is functioning, so we 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. The remainder of arw_simulator() remains the same.

At this point, modify your arw_simulator() function as shown below. As we progress through the labs, these modifications will be undone and you will be forced to understand how all the functions work together.

```
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 = X_real; %[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 1.

4. KINEMATICS MODIFICATIONS

Open the file named kinematicModel.m. It should look similar to (but not exactly like) the code shown in Fig. 3. In this case much of the code is missing. Given the input values of the previous state X_tm1 (which is supposed to represent $X_{t-1} = [\ x_{t-1}\ y_{t-1}\ \theta_{t-1}\ ]$ in units of meters and radians), and the recent encoder measurements $\Delta\varphi_R$ and $\Delta\varphi_L$ in units of pulses (where one encoder pulse equates to 1/4096 of a revolution), the function must calculate the updated state $X_t = [\ x_t\ y_t\ \theta_t\ ]$.

Use the lecture notes to fill in the kinematic equations.

```
function [X_t] = kinematicModel(X_tm1, deltaEncoderR, deltaEncoderL)

    % Some useful parameters
    robotRadius = 0.2;
    wheelRadius = 0.1;
    pulsesPerRevolution = 4096;
    pulsesToMeters = 2*pi*wheelRadius/pulsesPerRevolution;


    % Calcualate Distance travelled by each wheel based on wheel angular
    % velocity in pulses/s
    % wheelDistanceR = ...
    % wheelDistanceL = ...

    % Calculate the distance travelled by robot center deltaS, and the
    % angle rotated about center deltaTheta
    % deltaS = ...
    % deltaTheta = ...

    % Update States
    % theta_t = ...
    % x_t = ...
    % y_t = ...




    % Set State Vector
    X_t = [ x_t y_t theta_t];
end
```

Figure 3: The kinematicModel.m file to be modified in Lab 1.

5. ROBOT SIMULATIONS 1

If you implemented your kinematics correctly, your circular robot should be driving around the screen. Play with the hard coded control input vector U. Make sure the robot doesn't break any kinematic constraints. Ask if you are unsure. Note, the kinematics must be working for the rest of your labs to function properly.