ARW – Autonomous Robot Workshop

Lab 4

Path Tracking

1. INTRODUCTION

The purpose of this lab is to introduce participants to a straightforward point tracking control system that drives the robot to a desired point, despite the robot's nonholomic constraints. By the end of the lab, students should close the entire navigation control loop, in which the robot kinematics are simulated (Lab 1), the robot localizes itself with respect to the map (Lab 2), the robot constructs a collision-free path to its goal state (Lab 3), and uses the point tracking controller implemented in this lab (4) to track the path.

2. CODE BASE

The arw_simulator.m file should be calling all functions in the code base, (none will be commented out at this point). See Fig. 2.

The majority of the code to be modified in this lab will be located in the pathTrackingControl.m file. Within pathTrackingControl(), the function pointTrackingControl() is called to set the control vector U. You must modify pointTrackingControl().

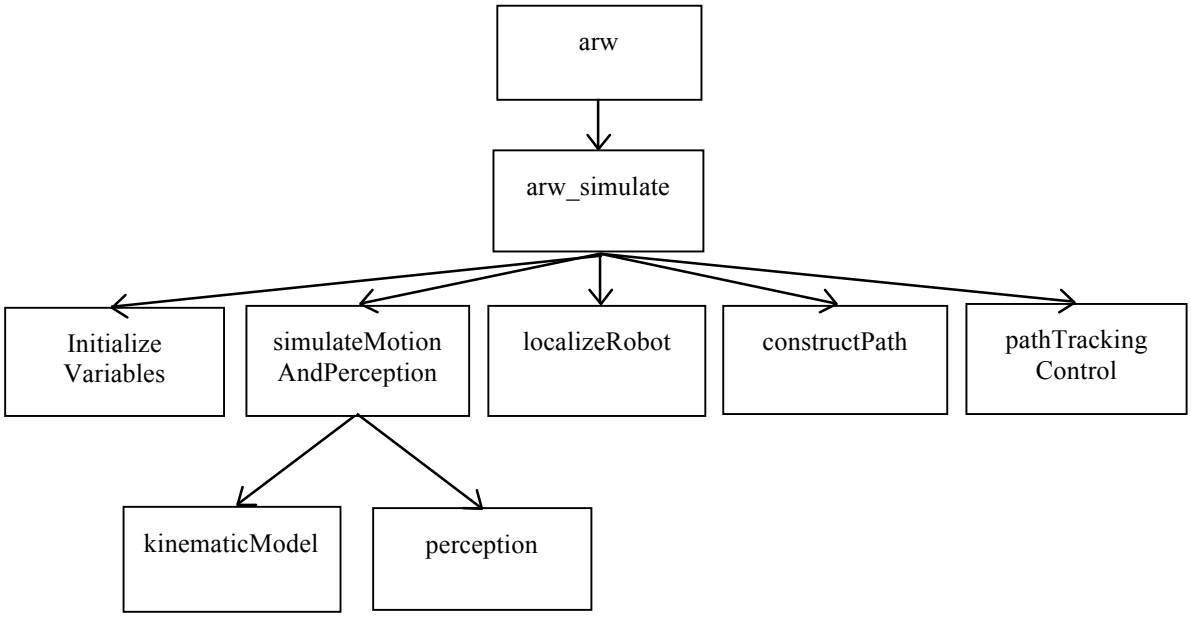

Figure 1: Codebase Architecture

```
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, 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 4.

3. MODIFYING POINTTRACKINGCONTROL

The point tracking control system inputs the desired position to track (X_toTrack) and the current estimated position (X_est) along with the tolerance within which the desired state must be tracked (trackingThresholds). Note that trackingThresholds is a 1x2 vector giving position and angle thresholds respectively. The function outputs the wheel rotational velocities (pulses/sec).

To calculate appropriate wheel velocities, first determine the current value of the distance error variable (rho) that must be driven to zero. Check if rho is in tolerance, and if so a rotate on the spot controller should be invoked. To accomplish this, reset rho to be zero, set alpha to be zero, and set beta to be the angle tracking error, i.e. the angular difference between the estimated theta of the robot and the desired theta.

If rho is not in tolerance, calculate the angular error variables alpha and beta. Using rho, alpha, beta, calculate v and w. Something we haven't covered in lecture, is a method to convert v and w to individual wheel velocity. Here is some code to accomplish this:

```
desiredWheelSpeedR = (v + w*robotWidth)*(2*pi)/(2*pi*wheelRadius);
desiredWheelSpeedL = (v - w*robotWidth)*(2*pi)/(2*pi*wheelRadius);
```

Details about derivations of this can be found in the HMC E160 lecture 2,3 notes:
http://www.hmc.edu/lair/E160/lectures.html

Before setting U to be these wheel speeds, we must first make sure they don't break any physical speed limits. This speed limiting is coded for you already at the bottom of the pointTrackingControl() function.

```
function U = pointTrackingControl(X_toTrack, X_est, trackingThresholds)

    % Some useful variables
    K_rho = 1.0;
    K_alpha = 2.0;
    K_beta = -0.99;
    robotRadius = 0.2;
    robotWidth = robotRadius*2;
    wheelRadius = 0.1;

    % Some helpful calculations
    deltaX = X_toTrack(1) - X_est(1);
    deltaY = X_toTrack(2) - X_est(2);
    ang = atan2(deltaY,deltaX);

    % Calucate rho, the distance to goal
    % rho = ...

    % Check to see if simply rotation on the spot, if rho is small
    % if ...
        % rho = ...
        % alpha = ...
        % beta = ...


    % if rho is still too big, invoke the point tracking theory
    % else ...

        % Calculate alpha and beta
        % alpha = ...
        % beta = ...


    % Calculate v, w
    % v = ...
    % w = ....


    % Calculate desired wheelSpeed in pulses / second
    %desiredWheelSpeedR = ...
    %desiredWheelSpeedL = ...


    % Check that max velocity in rad/s isn't reached
    currentMaxVel = max(abs(desiredWheelSpeedL), abs(desiredWheelSpeedR));
    maxWheelSpeed = 2*pi;
    if  currentMaxVel > maxWheelSpeed
        desiredWheelSpeedR = desiredWheelSpeedR * maxWheelSpeed / currentMaxVel;
        desiredWheelSpeedL = desiredWheelSpeedL * maxWheelSpeed / currentMaxVel;
    end

    % Set the control vector U
    U = [desiredWheelSpeedR desiredWheelSpeedL];

end
```

Figure 3: The pointTrackingControl.m file for use with Lab 4.

Once you have completed your code, test it out. Make sure the robot follows the red paths constructed by your motion planner from Lab 4.