LAB 1 - PART 1

Maintain a fixed distance from the wall by implementing a PD controller

```
clc
clear
close all

addpath('../rc-labs/rc-matlab-lib');

% ConnectToROS();
% Connect to RC
% RC = RCCar();

% Params
KP = .5; % UPDATE THIS
KD = 17; % UPDATE THIS

divisor = 1;

KP = KP / divisor;
KD = KD/ divisor;
desiredDistance = 0.5; % m
duration = 10.0; % s
frequency = 20.0; % Hz
totalTimeSteps = duration * frequency;
speed = 0.5; % m/s

% Variable histories
currentTime_History = nan(1, totalTimeSteps);
distanceError_History = nan(1, totalTimeSteps);
steeringAngle_History = nan(1, totalTimeSteps);

% Keep track of last error and time to calculate rate of change
lastDistanceError = 0;
startTime = datetime("now");
lastTime = startTime;

% Control loop
for k = 1:totalTimeSteps

    % Get the current time
    currentTime = datetime('now');

    % Calculate the time difference from the last function call
    deltaTime = seconds(currentTime - lastTime); % s

    % Obtain the current distance to the right wall
    currentDistance = getDistanceToRightWall(RC); % m

    % Calculate the error in distance
    distanceError = -currentDistance + desiredDistance; % m
```

```
    % Calculate the change in distance error since last iteration

    deltaDistanceError = distanceError - lastDistanceError; % m

    % Rate of change of error with respect to time
    rateOfChangeError = deltaDistanceError/deltaTime; % m/s

    % Calculate the steering angle using PD control law
    steeringAngle = KP * distanceError + KD * rateOfChangeError;

    lastDistanceError = distanceError;

    % Send steering angle command
    RC.setSteeringAngle(steeringAngle);

    % Send speed command
    RC.setSpeed(speed); % constant

    % Add variables to history
    currentTime_History(k) = seconds(currentTime - startTime);
    distanceError_History(k) = distanceError;
    steeringAngle_History(k) = steeringAngle;

     % Display the current information for debugging or monitoring
    fprintf("Distance Error = %.3f m\n", distanceError);
    fprintf("Rate of Change of Error = %.3f m/s\n", rateOfChangeError);
    fprintf("Steering Angle: %.4f rad\n", steeringAngle);

    % Maintain fixed frequency
    pause(1/frequency);

end

% Plot results
figure()

subplot(1, 2, 1)
plot(currentTime_History, distanceError_History)
xlabel("Time (s)")
ylabel("Distance Error (m)")

subplot(1, 2, 2)
plot(currentTime_History, steeringAngle_History)
xlabel("Time (s)")
ylabel("Steering Angle (rad)")

sgtitle("RC Wall Follower Results")


function distance = getDistanceToRightWall(rc)

% Define scan sample range
angle_min = -3*pi/4;
angle_max = -pi/4;
```

```
% Get distances in that range
distances = getDistancesInRange(rc, angle_min, angle_max);

% Distance to the wall is the minimum distance in sample
distance = min(distances);
```

```
File 'getDistanceToLefttWall.m' not found.
```

```
% Retrieves the distance from the LIDAR scan at a specified angle.
function distance = getDistancesInRange(obj, angle_min, angle_max)

% Get the latest LIDAR scan data
scan = obj.getScan();

% Validate if the provided angle lies within the scan's range
if (angle_min < scan.AngleMin) || (angle_max > scan.AngleMax)
    fprintf("Provided scan angle range [%f, %f] is out of the  lidar range
[%f, %f].\n", ...
        angle_min, angle_max, scan.AngleMin, scan.AngleMax);
    distance = -1;  % Return an error code for out-of-range angle
    return;
end

% Create angle range vector
angle_range = angle_min:scan.AngleIncrement:angle_max;

% Calculate the index of the scan data corresponding to the given angle
indices = round((angle_range - scan.AngleMin) / scan.AngleIncrement);

% Fetch and return the distance at the calculated index from the scan ranges
distance = scan.Ranges(indices);

% Change 0 distance (unknown) to nan
distance(distance == 0) = nan;

end
```

Published with MATLAB® R2024a