FINAL PROJECT EML 4806 "MODELING AND CONTROL OF ROBOTS"

Line Following Robot

APRIL 8, 2023

I. Introduction

A line following robot is an autonomous robot that follows a line on a track using sensors and control algorithms. These robots have various applications in industries, such as warehouse automation, manufacturing, and transportation.

The scope of line following robots is vast, and their applications continue to expand as technology advances. With the development of new sensors, control algorithms, and hardware components, line following robots can be designed to perform increasingly complex tasks in various industries and applications.

II. Problem Statement

Overall, the importance and scope of line following robots make them a fascinating and promising area of robotics and automation. As technology advances, these robots will continue to play a vital role in various industries and educational settings, shaping the future of automation and robotics

III. Design and Implementation

Components Needed:

- Microcontroller board (Arduino or Raspberry Pi)
- Motor driver board
- DC geared motors
- Line sensor module (QTR-8RC or TCRT5000)
- Power source (battery pack or power supply)
- Chassis (plastic or metal)
- Wheels
- Jumper wires
- Screws and nuts

Assembly Description:

- First, assemble the chassis by attaching the motors to the wheels using screws and nuts.
- Then, attach the line sensor module to the front of the chassis using screws and nuts.
- Connect the motors to the motor driver board using jumper wires. Connect the motor driver board to the microcontroller board using jumper wires.
- Connect the line sensor module to the microcontroller board using jumper wires.
- Connect the power source to the motor driver board and the microcontroller board.
- Write a code to read the sensor data and control the motors. The code should instruct the robot to follow the line by turning left or right based on the sensor data.
- Test the robot by placing it on a line and turning it on.

IV. Results and Analysis

The Matlab code consists of two files, first one is the making_map.m that is generating random maps for robot and follow.m is used by robot to follow the line generated.

a. Making_map.m

The Matlab code for this is added in Appendix A. This Matlab code generates a random road map consisting of multiple curves using Bezier curves. The road map is displayed in a figure and stored in a variable for further use.

The parameters for the road map generation include the number of curves to generate (n_curves), the number of points per curve (n_points), and the range of x and y coordinates (x_min, x_max, y_min, and y_max).

The code initializes an array (map_curves) to store the x and y coordinates of the points on the curves. Then, for each curve, it generates four random control points using the specified range of x and y coordinates. The Bezier curve equation is used to generate points along the curve, which are stored in the map_curves array.

Finally, the road map is displayed in a figure using the plot function, and the map_curves array is stored in a variable named map.

b. Follow.m

The Matlab code for this is added in Appendix B. This MATLAB code simulates a line-following robot moving along a predetermined path on a map generated by making_map.m. The map is defined as a set of points (x, y coordinates) that the robot must follow. The script creates a figure window with a rectangle object that represents the robot and two animated line objects that represent the path the robot has covered and the remaining path.

The script initializes the starting point of the path and adds it to the path array. It then iterates over the map and adds each point to the path array if the distance between the current point and the last point in the path is greater than a threshold value (0.1 in this case).

The script then creates a rectangle object that represents the robot, sets the axes limits to the minimum and maximum x and y coordinates of the map, and creates two animated line objects that represent the covered path and the remaining path. It adds the first point to the covered path and all the points except the first point to the remaining path.

The script then iterates over the remaining path and moves the robot along the path. It updates the position of the robot, adds the current point to the covered path line, removes it from the remaining path line, and updates the title of the remaining path line to show the number of remaining points. The script pauses for 0.1 seconds before moving to the next point in the path.

Overall, this script provides a simple simulation of a line-following robot moving along a predetermined path on a map, and can be used as a starting point for more complex simulations or robotics projects.

V. Evaluation

The line following robot has been successfully doing its job by following the line.

VI. Future work

The development of advanced control algorithms, such as model-based predictive control or reinforcement learning, can enable line following robots to perform more complex tasks and operate in dynamic environments. The integration of environmental sensors, such as temperature, humidity, or gas sensors, can enable line following robots to gather additional information about their surroundings and adapt their behavior accordingly. The integration of multiple sensing modalities, such as vision, infrared, and ultrasonic sensors, can improve the robustness and accuracy of line following robots. The development of natural and intuitive interfaces for human-robot interaction, such as voice or gesture recognition, can enable users to interact with line following robots more effectively and efficiently. The study of swarm robotics, which involves the coordination of multiple robots to accomplish a task, can enable line following robots to work together in teams to achieve complex goals.

VII. Conclusion

In conclusion, line following robots are autonomous robots that can follow a line on a track using sensors and control algorithms. These robots have various applications in industries and education, including automation of repetitive tasks, reduction of labor costs, and promotion of critical thinking and creativity in students. With the development of new sensors, control algorithms, and hardware components, line following robots continue to evolve and expand their scope, making them a fascinating and promising area of robotics and automation.

VIII. References

- https://www.mathworks.com/help/supportpkg/arduino/ref/arduino-robot-line-follower-application.html

- "Design of line following robot on Matlab", International Journal of Emerging Technologies and Innovative Research (www.jetir.org | UGC and issn Approved), ISSN:2349-5162, Vol.8, Issue 5, page no. ppe819-e821, May-2021,

- https://in.mathworks.com/matlabcentral/fileexchange/66586-mobile-robotics-simulationtoolbox

IX. Appendices

Appendix A

Matlab Code for making Map

```
% Set parameters
n_curves = 5; % Number of curves in the map
n_points = 100; % Number of points per curve
x_min = -10; % Minimum x-coordinate of the map
x_max = 10; % Maximum x-coordinate of the map
y_min = -10; % Minimum y-coordinate of the map
y_max = 10; % Maximum y-coordinate of the map

% Generate random curves
map_curves = zeros(n_curves*n_points, 2); % Initialize array for curve points
for i = 1:n_curves
    % Generate random control points for the current curve
    if i == 1
        % For the first curve, start at a random point
        x1 = rand*(x_max - x_min) + x_min;
        y1 = rand*(y_max - y_min) + y_min;
        x2 = x1 + rand*(x_max - x_min)/2;
        y2 = y1 + rand*(y_max - y_min)/2;
    else
        % For subsequent curves, start at the end of the previous curve
        x1 = map_curves((i-2)*n_points+n_points, 1);
        y1 = map_curves((i-2)*n_points+n_points, 2);
        x2 = x1 + rand*(x_max - x_min)/2;
        y2 = y1 + rand*(y_max - y_min)/2;
    end
    x3 = x2 + rand*(x_max - x_min)/2;
    y3 = y2 + rand*(y_max - y_min)/2;
    x4 = x3 + rand*(x_max - x_min)/2;
    y4 = y3 + rand*(y_max - y_min)/2;
    % Generate points on the current curve using Bezier curve
    t = linspace(0, 1, n_points);
    px = (1-t).^3*x1 + 3*(1-t).^2.*t*x2 + 3*(1-t).*t.^2*x3 + t.^3*x4;
    py = (1-t).^3*y1 + 3*(1-t).^2.*t*y2 + 3*(1-t).*t.^2*y3 + t.^3*y4;
    % Store points in array
    map_curves((i-1)*n_points+1:i*n_points,:) = [px' py'];
end

% Display map
figure;
plot(map_curves(:,1), map_curves(:,2), 'b');
xlim([-10 90]);
ylim([-10 90]);
title('Random Road Map');

% Store map in a variable
map = map_curves;
```

Appendix B

Matlab code to follow Line

```
% Initialize the starting point of the path
start = [map(1,1), map(1,2)];

% Initialize the path with the starting point
path = start;

% Iterate over the map and add the next point in the path
for i = 2:size(map,1)
    % Get the coordinates of the current point
    current = [map(i,1), map(i,2)];

    % Calculate the distance between the current point and the last point in the path
    distance = norm(current - path(end,:));

    % If the distance is greater than a threshold (e.g. 0.1), add the current point
to the path
    if distance > 0.1
        path = [path; current];
    end
end

% Create a figure window
fig = figure;

% Set the title of the figure
title('Line Following Robot Path');

% Set the axes limits to the minimum and maximum x and y coordinates of the map
xlim([min(map(:,1)) max(map(:,1))]);
ylim([min(map(:,2)) max(map(:,2))]);

% Create a rectangle object that represents the robot
robotWidth = 1; % Set the width of the robot
robotHeight = 1; % Set the height of the robot
robot = rectangle('Position', [start(1)-robotWidth/2, start(2)-robotHeight/2,
robotWidth, robotHeight], ...
                  'FaceColor', 'yellow');

% Create an animated line object that represents the path robot has covered
coveredPathLine = animatedline('Color', 'b', 'LineWidth', 2, 'DisplayName', 'Covered
Path');

% Create an animated line object that represents the remaining path
remainingPathLine = animatedline('Color', 'r', 'LineWidth', 2, 'DisplayName',
'Remaining Path');

% Add a legend to the figure
legend('Location', 'northwest');

% Add the first point to the covered path line
```

```
addpoints(coveredPathLine, path(1,1), path(1,2));

% Add all the points except the first point to the remaining path line
addpoints(remainingPathLine, path(2:end,1), path(2:end,2));

% Iterate over the remaining path and move the robot
for i = 2:size(path,1)

    % Update the position of the robot
    set(robot, 'Position', [path(i,1)-robotWidth/2, path(i,2)-robotHeight/2,
robotWidth, robotHeight]);

    drawnow;

    % Add the current point to the covered path line and remove it from the remaining
path line
    addpoints(coveredPathLine, path(i,1), path(i,2));
    clearpoints(remainingPathLine);
    addpoints(remainingPathLine, path(i+1:end,1), path(i+1:end,2));

    % Set the title of the remaining path line
    title(sprintf('Remaining Path: %d points', size(path(i+1:end,:),1)));
    xlabel('Line Following Robot');
    % Pause for 0.1 seconds
    pause(0.1);
end
```