Table of Contents


```
clc; clear all; close all; format compact;

% Exam 2 Simulation Mobile Robotics
% Jack Vranicar 11/25/24
```

Robot Characteristics

```
Robot.x = 0;
Robot.y = 0;
Robot.phi = pi/4;
Robot.radius = 1;
Robot.width = .125;
Robot.length = Robot.width/2;
Robot.velocity = 1;

Wheel.radius = Robot.length/4;
Wheel.wheel_width = Wheel.radius;
```

Obstacle creation

```
obst_radius = .1;

obstPosns = [1 1.5; 3 1; 3 2];
map = [0 5 0 5];

figure;
grid on

Obstacle = makeObstacles(obstPosns, obst_radius, [3.5 2 pi/4], map);
nobst = size(obstPosns, 1);

for i = 1:nobst
    hold on
    fill(Obstacle.(strcat("O", num2str(i))).xCoords, Obstacle.(strcat("O",
num2str(i))).yCoords, 'r')
    axis equal
    axis(map)
end

fill(Obstacle.OU.xCoords, Obstacle.OU.yCoords, "g")
numsegments = size(Obstacle.OU.xCoords, 2) - 1;
```

```
for i = 1:numsegments

    segment_x(i) = mean(Obstacle.OU.xCoords(i:i+1));
    segment_y(i) = mean(Obstacle.OU.yCoords(i:i+1));
    segment_phi(i) = atan2(Obstacle.OU.yCoords(i+1) -
Obstacle.OU.yCoords(i), Obstacle.OU.xCoords(i+1) - Obstacle.OU.xCoords(i));

end
```

Go to Goal

```
y_goal = 4.5;
x_goal = 4;
plot(x_goal, y_goal, 'bo')
goalThreshold = .1;
dt = .05;
state = 1;

obstTolTowards = .5;
obstTolAway = .75;
wallToleranceTowards = .2;
wallToleranceAway = .2;

while getEuclideanDist([Robot.x Robot.y], [x_goal y_goal]) > goalThreshold
    closestDistance = inf;

    distToGoal = getEuclideanDist([Robot.x Robot.y], [x_goal y_goal]);

    distToObstacle1 = getEuclideanDist([Robot.x Robot.y], [obstPosns(1, 1),
obstPosns(1, 2)]);
    distToObstacle2 = getEuclideanDist([Robot.x Robot.y], [obstPosns(2, 1),
obstPosns(2, 2)]);
    distToObstacle3 = getEuclideanDist([Robot.x Robot.y], [obstPosns(3, 1),
obstPosns(3, 2)]);

    phiObst1 = atan2(obstPosns(1,2) - Robot.y, obstPosns(1,1) - Robot.x);
    phiObst2 = atan2(obstPosns(2,2) - Robot.y, obstPosns(2,1) - Robot.x);
    phiObst3 = atan2(obstPosns(3,2) - Robot.y, obstPosns(3,1) - Robot.x);

    % Finding out which segment of the wall the robot is the closest to
    for z = 1:numsegments
        curDistance = getEuclideanDist([Robot.x Robot.y], [segment_x(z)
segment_y(z)]);
        if curDistance < closestDistance
            closestDistance = curDistance;
            closestSegmentIndex = z;
        end
    end


phi_goal = atan((y_goal - Robot.y) / (x_goal - Robot.x));

    switch state
```

```
        case 1 % Go to goal case

            phi_d = phi_goal;

            if (distToObstacle1 < obstTolTowards)
                state = 2;
                phi_d = phiObst1 + pi;
            elseif (distToObstacle2 < obstTolTowards)
                state = 2;
                phi_d = phiObst2 + pi;
            elseif (distToObstacle3 < obstTolTowards)
                state = 2;
                phi_d = phiObst3 + pi;
            elseif (closestDistance < wallToleranceTowards)
                state = 3;
            end

        case 2 % Obstacle avoidance

            if distToObstacle1 > obstTolAway
                state = 1;
            end

        case 3 % Wall Following

            mu_CW = segment_phi(closestSegmentIndex);
            mu_CCW = -segment_phi(closestSegmentIndex);

            if abs(phi_d) - abs(mu_CW) <= pi
                phi_d = mu_CW;
            else
                phi_d = mu_CCW;
            end

            if (closestDistance > wallToleranceAway) && (abs(phi_goal -
phi_d) <= pi)
                state = 1;
            end


    end

    x_dot = Robot.velocity * cos(phi_d);
    y_dot = Robot.velocity * sin(phi_d);
    Robot.phi = phi_d;
    Robot.y = y_dot * dt + Robot.y;
    Robot.x = x_dot * dt + Robot.x;


    [h1, h2, h3] = drawRobot(Robot, Wheel);
    pause(dt);
    delete(h1);
    delete(h2);
```

```
    delete(h3);

end


function [h1, h2, h3] = drawRobot(Box, Wheel)

% All for main body
length = Box.length; %y-direction
width = Box.width; % x-direction

y_box = [-length/2 -length/2 length/2 length/2 -length/2];
x_box = [width/2 -width/2 -width/2 width/2 width/2];

x = Box.x;
y = Box.y;
phi = Box.phi;

rot_matrix = [cos(phi), -sin(phi); sin(phi), cos(phi)];
box_rotated = rot_matrix * [x_box; y_box];

box_translated_rotated = [box_rotated(1,:) + x; box_rotated(2,:) + y];


% Wheels
radius = Wheel.radius;
wheel_width = Wheel.wheel_width; %y direction

% Left wheel
x_left_wheel = [(-width/2), (-width/2), (-width/2 + 2*radius), (-width/2 +
2*radius), (-width/2)];
y_left_wheel = [(length/2 + wheel_width), (length/2), (length/2), (length/2
+ wheel_width), (length/2 + wheel_width)];

left_wheel_rotated = rot_matrix * [x_left_wheel; y_left_wheel];

left_wheel_rotated_translated = [left_wheel_rotated(1,:) + x;
left_wheel_rotated(2,:) + y];

% Right Wheel

x_right_wheel = [(-width/2), (-width/2), (-width/2 + 2*radius), (-width/2 +
2*radius), (-width/2)];
y_right_wheel = [(-length/2 - wheel_width), (-length/2), (-length/2), (-
length/2 - wheel_width), (-length/2 - wheel_width)];

right_wheel_rotated = rot_matrix * [x_right_wheel; y_right_wheel];

right_wheel_rotated_translated = [right_wheel_rotated(1,:) + x;
right_wheel_rotated(2,:) + y];

%Plotting
h1 = fill(right_wheel_rotated_translated(1,:),
right_wheel_rotated_translated(2,:), 'k');
```

```
h2 = fill(left_wheel_rotated_translated(1,:),
left_wheel_rotated_translated(2,:), 'k');
h3 = fill(box_translated_rotated(1,:), box_translated_rotated(2,:), 'r');


end



function [dist] = getEuclideanDist(startPoint, endPoint)

dist = sqrt( (startPoint(2) - endPoint(2) )^2 + (startPoint(1) -
endPoint(1) )^2 );
end



function [Obstacle] = makeObstacles(obst_posns, rad, u_posn, limits)


%% Making circular shapes

drawThetas = linspace(0, 2*pi, 500);


for i = 1:size(obst_posns, 1)

    % Radius
    r =  rad;

    x_circle = r*cos(drawThetas);
    y_circle = r * sin(drawThetas);

    % Position
    x_pos = obst_posns(i, 1);
    y_pos = obst_posns(i, 2);

    % Check X Pos
    Obstacle.(strcat("O", num2str(i))).xCoords = x_circle + x_pos;
    Obstacle.(strcat("O", num2str(i))).yCoords = y_circle + y_pos;

end

%% Making the U shape

u_x_vals = [zeros(1, 4), linspace(0, .8, 4), ones(1,4), linspace(1, .2, 4),
zeros(1,4), linspace(0, 1, 4), 1.25*ones(1,5), linspace(1.25, 0.25, 4)];
u_y_vals = [linspace(0, .2, 4), .25*ones(1,4), linspace(.25, 1.25, 4),
1.5*ones(1,4), linspace(1.5, 1.7, 4), 1.75*ones(1,5), linspace(1.4, .28, 4),
zeros(1,4)];

theta = u_posn(3);
rotMatrix = [cos(theta), -sin(theta); sin(theta), cos(theta)];
coords = rotMatrix * [u_x_vals; u_y_vals];
```

```
Obstacle.OU.xCoords = coords(1, :) + u_posn(1);
Obstacle.OU.yCoords = coords(2,:) + u_posn(2);

end
```

Published with MATLAB® R2024a