Faculty of Power and Aeronautical Engineering
Department of Robotics

Mobile Robot Report

Task 6

Jafar Jafar,
Pratama Aji Nur Rochman

supervised by
dr. Wojciech Szynkiewicz, & Dawid Seredyński

Contents

1 Introduction

The goal of this task is to introduce path planning implemented with the Wavefront Planner. This algorithm is capable for finding the optimal route to the goal in the planning phase, in the initial state. For this purpose it analyses the map and generates the path and in an off-line manner. When finished, the robot simply follows the generated path. The detailed description of the Wavefront Planner is in the Lecture 11.

In this task you can run the simulation with the following command in the MATLAB console: run_simulation(@solution6, false, [goal_x, goal_y], map_filename) where solution6 is the control callback function you have to implement, [goal_x, goal_y] is the position of the goal point and map_filename is the name of the map file. It is recommended to plot the state of the Wavefront Planner for debugging and visualisation purposes.

The planner should run in the initial state. The generated collision-less trajectory should be executed using the controller implemented in the (Task 1). The robot should move to the subsequent cells on the path generated by Wavefront.

2 Mapping

he planning algorithm requires a map of the environment. The map files are shipped together with the scene files *.ttt. Each scene file <filename>.ttt has its map file <filename>.png. You can use all scenes provided in the vrep_env directory to experiment and test your solution.

To read a map file use the imread Matlab function, e.g.:

$map = imread('vrep\_env/map2.png')$

The map is a matrix, where each cell corresponds to the occupancy state of a particular place in the scene. The state of a cell is either 0 (occupied) or 255 (free), e.g. map(1,1) == 0 means that the scene section with the lowest x and the lowest y is occupied. The map is indexed in ranges:

for x: from 1 to size_x,

for y: from 1 to size_y,

where [size_y, size_x] = size(map);. As the map corresponds to the scene, please take into account the units conversion, e.g. if the scene spans from -5m to 5m in x direction and from -10m to 10m in y direction, and the size of the map is 100 by 200 cells (size(map)==[200,100]), the units conversions are:

x_map=round(100*((x_world-(-5))/(5-(-5))))

y_map=round(200*((x_world-(-10))/(10-(-10))))

The inverse conversion can be easily calculated. Please note that the indexing of the map is (y,x).

All environments span from -7.5m to 7.5m in both x and y directions, and the size of all maps is 100 by 100.

3 Task requirements

- The robot is supposed to move in diverse environments

4 Solution

4.1 Code

The callback function for this task was declared as:

```
function [forwBackVel, leftRightVel, rotVel, finish] = solution6(pts, contacts, position,
     orientation, varargin)

    if length(varargin) ~= 2
        error('Wrong number of additional arguments: %d\n', length(varargin));
    end

    d = varargin{1};
    goal_x = d(1);
    goal_y = d(2);
    map = varargin{2};

    % State Machine (FSM)
    persistent state;
    if isempty(state)
        % the initial state of the FSM is 'init'
        state = 'init';
    end

    % initialize function return variables
    forwBackVel = 0;
    leftRightVel = 0;
    rotVel = 0;
    finish = 0;

    persistent x_world;
    persistent y_world;
```

```
    persistent curr;
    tolerances = [0.1 0.1];

    % for propotional regulator
    gain_P= [12 12];
    u_max = [5 5];            % max speeds
    u_min = [-5 -5];          % min speeds


    % manage the states of FSM
    if strcmp(state, 'init')

        % read image
        map = strcat('vrep_env/', map);
        pic = strcat(map, '.png');
        mtx = imread(pic);
        map = mtx;

        % get size of matrix
        [size_y, size_x] = size(mtx);

        x_init = position(1);
        y_init = position(2);

        x_init_matrix = round(size_x*((x_init-(-7.5))/(7.5-(-7.5))));
        y_init_matrix = round(size_y*((y_init-(-7.5))/(7.5-(-7.5))));
        x_goal_matrix = round(size_x*((goal_x-(-7.5))/(7.5-(-7.5))));
        y_goal_matrix = round(size_y*((goal_y-(-7.5))/(7.5-(-7.5))));

        % Wavefront Planner
        mtx(mtx < 255) = 1;
        mtx(mtx == 255) = 0;
        mtx(y_goal_matrix, x_goal_matrix) = 2;
        mtx(y_init_matrix, x_init_matrix) = 0;

```

```
        % where we have 1 we thicken the wall
        neighbor = round(0.5/15 * 100);
        [a, b] = ind2sub(size(mtx), find(mtx == 1));

        for k = 1: length(a)
            j = a(k);
            i = b(k);
            for m = (j - neighbor): (j + neighbor)
                for n = (i - neighbor): (i + neighbor)
                    if (m > 0 && n > 0 && m <= size_y && n <= size_x)
                        if (mtx(m, n) == 2 || (x_init_matrix == n && y_init_matrix == m)
                            || mtx(m, n) == 1)
                        else
                            mtx(m,n) = 1;
                        end
                    end
                end
            end
        end

        f = 2;
        while mtx(y_init_matrix, x_init_matrix) == 0

            [a, b] = ind2sub(size(mtx), find(mtx == f));

            for k = 1: length(a)
                j = a(k);
                i = b(k);
                if mtx(j, i) == 1
                    continue
                end
                for m = (j - 1): (j + 1)
                    if (m > 0 && m <= size_y)
                        if (mtx(m, i) == 1 || mtx(m, i) == f || mtx(m, i) == f - 1)
                        else
```

```
                        mtx(m, i) = f + 1;
                    end
                end
            end
            for n = (i - 1): (i + 1)
                if (n > 0 && n <= size_x)
                    if (mtx(j, n) == 1 || mtx(j, n) == f || mtx(j, n) == f - 1)
                    else
                        mtx(j, n) = f + 1;
                    end
                end
            end
        end
        f = f + 1;
    end

    % Wavefront Planner - Phase 2
    goal_matrix = [y_goal_matrix, x_goal_matrix];
    solution = [];

    current = [y_init_matrix, x_init_matrix];
    while current(1) ~= goal_matrix(1) || current(2) ~= goal_matrix(2)
        j = current(1);
        i = current(2);
        min_value = mtx(current(1), current(2));
        min_index = current;

        for m = (j + 1): -1: (j - 1)
            for n = (i + 1): -1: (i - 1)
                if (m > 0 && n > 0 && m <= size_y && n <= size_x)
                    if mtx(m, n) > 1
                        if (mtx(m, n) < min_value)
                            min_value = mtx(m, n);
                            min_index = [m, n];
                        end
```

```
                        end
                    end
                end
            end
            solution = [solution; min_index];
            current = min_index;
        end

        % plotting
        figure;
        imagesc(map);
        hold on;
        plot(solution(:, 2), solution(:, 1), 'r-*', 'linewidth', 1.5);
        set(gca, 'ydir', 'normal');

        % units conversion from IMAGE to WORLD
        x_world = solution(:, 2) .* (7.5-(-7.5))/size_x + (-7.5);
        y_world = solution(:, 1) .* (7.5-(-7.5))/size_y + (-7.5);

        curr = 1;
        state = 'move';

    elseif strcmp(state, 'move')

        dest = [x_world(curr), y_world(curr)];
        u = zeros(2, 1);
        for i = 1: 2
            measured = position(i);
            errors = dest(i) - measured;
            u(i) = gain_P(i) * errors;
            if u(i) > u_max(i)
                u(i) = u_max(i);
            elseif u(i) < u_min(i)
                u(i) = u_min(i);
            end
```

```
        end

        % changing global velocities to local
        phi = orientation(3);
        speed_x = cos(phi) * u(1) + sin(phi) * u(2);
        speed_y = -sin(phi) * u(1) + cos(phi) * u(2);

        % setting speeds
        forwBackVel = speed_y;
        leftRightVel = speed_x;

        % criteria to advance current division
        if curr < length(x_world)
            if abs(position(1) - dest(1)) <= 0.15 && abs(position(2) - dest(2)) <= 0.15
                curr = curr + 1;
            end
        end

        % check if goal position reached
        if abs(position(1) - goal_x) <= tolerances(1) && abs(position(2) - goal_y) <=
            tolerances(2)
            fprintf('changing FSM state to %s\n', state);
            finish = 1;
        end
    end
end
```

5 Result

In this task you can run the simulation with the following command in the MATLAB console: run_simulation(@solution6, false, [goal_x, goal_y], 'exercise02')

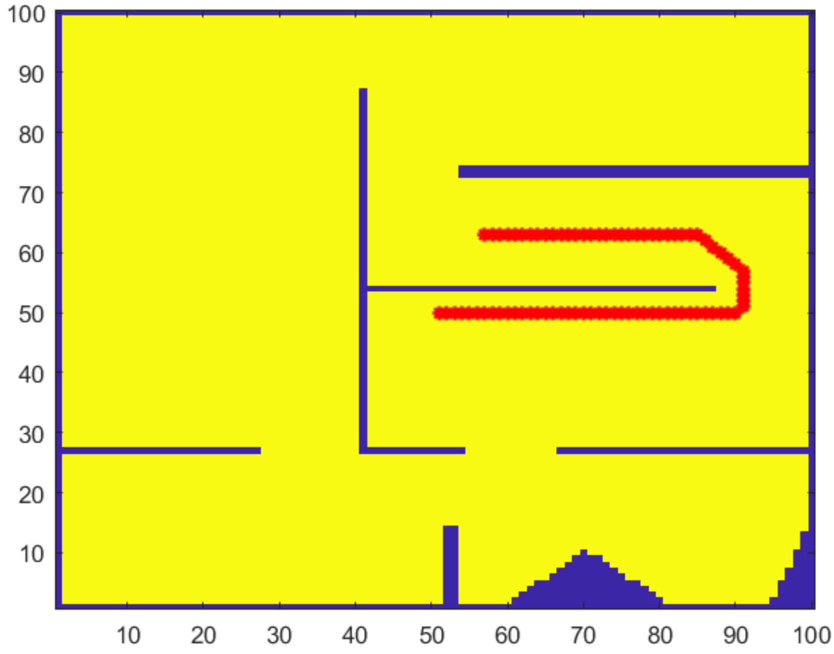

Figure 1: Plotting

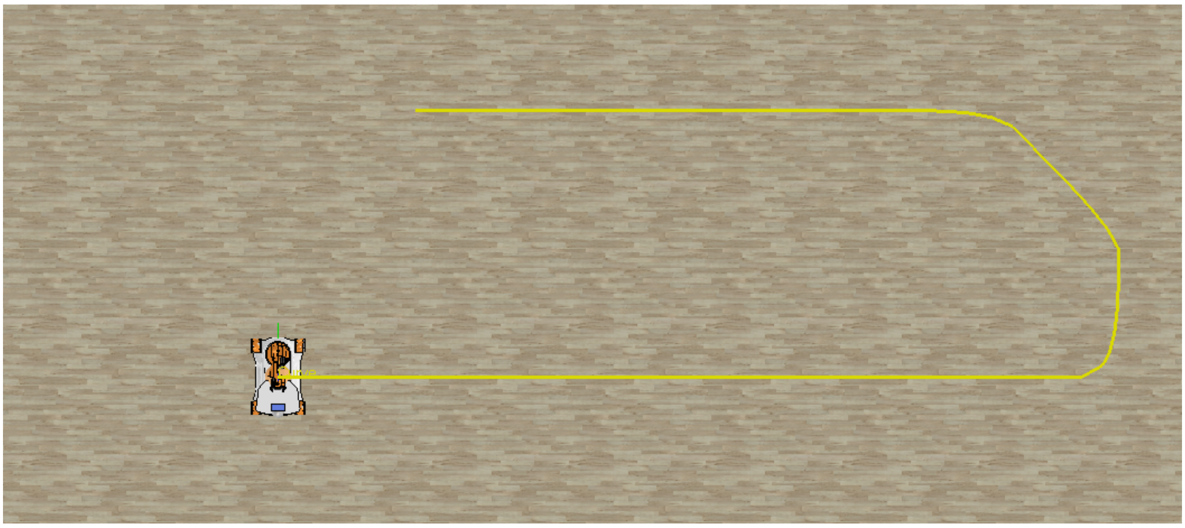

Figure 2: goal_x = 1, goal_y = 2