Faculty of Power and Aeronautical Engineering
Department of Robotics

Mobile Robot Report

Task 3

Jafar Jafar,
Pratama Aji Nur Rochman

supervised by
dr. Wojciech Szynkiewicz, & Dawid Seredyński

Contents

1 Introduction

The main goal of this task is to learn to control a robot using multiple behaviours governed by a finite state machine (FSM).

Finite State Machines are a useful tool for programming robots. There are many resources you can read to undestand the concept of FSM, e.g. this article

Your task is to generate a trajectory that uses both trajectory generators from the Task 1 and Task 2. The robot should follow a rounded rectangle path that consists of two half circles and two straight lines:

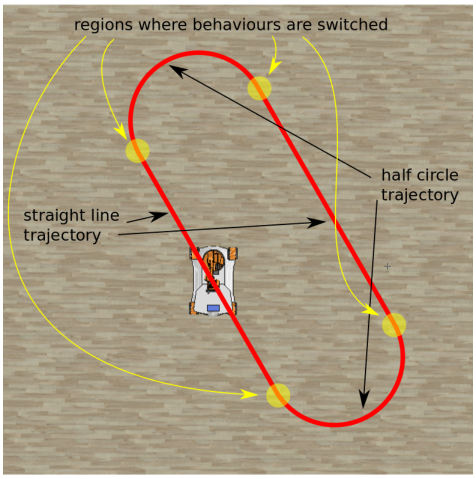

Figure 1: Orientation position

There are three parameters that describe the path: diameter, length and angle:

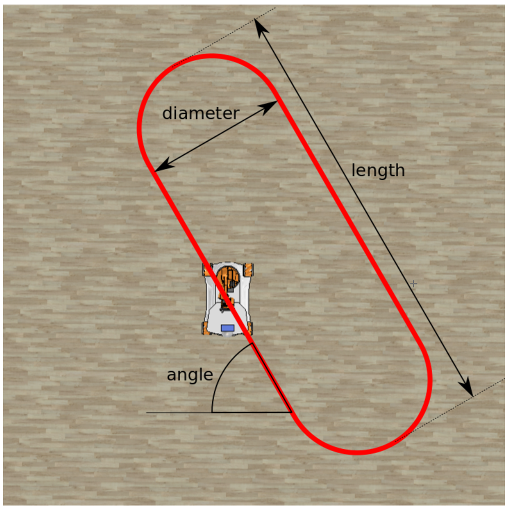

Figure 2: Parameter position

Write a script solution3.m that implements a callback function solution3. The parameters are passed to the control program through the variable-length argument list in the run_simulation function, e.g. run_simulation(@solution3, false, 1, 3, 0.5) where the arguments are:

- solution3 - the name of your control callback function
- false - do not display the sensor data (simulation runs faster)
- 1 - the diameter of the rouded rectangle (this is an example value)
- 3 - the length of the rouded rectangle (this is an example value)
- 0.5 - the angle of the rouded rectangle (in radians, this is an example value)

2 Task requirements

- The name of the control callback function should be solution3
- Both position and orientation must be controlled
- The robot must follow the desired path
- The robot can move infinitelly
- Use the environment file /emor_trs/youbot/vrep_env/exercise01.ttt

3 Solution

3.1 Code

The callback function for this task was declared as:

```
function [forwBackVel, leftRightVel, rotVel, finish] = solution3(pts, contacts, position, orientation, varargin)
% The control loop callback function - the solution for Task 3

    % get the parameters
    if length(varargin) ~= 3,
         error('Wrong number of additional arguments: %d\n', leng(varargin));
    end
    param_diameter = varargin{1};
    param_length = varargin{2};
    param_angle = varargin{3};

    % declare the persistent variable that keeps the state of the Finite
    % State Machine (FSM)
```

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

    % initialize the robot control variables (returned by this function)
    finish = false;
    forwBackVel = 0;
    leftRightVel = 0;
    rotVel = 0;

    % TODO: manage the states of FSM
    persistent init_position;
    persistent init_orientation;
    global circle_center;
    global xyO;
    global fflag;

    %Find the fist X and Y values given Diameter, Length and Diameter
    alpha = param_angle;
    X = -cos(alpha) * (param_length-param_diameter);
    Y = sin(alpha) * (param_length-param_diameter);
    diameter = [0 param_diameter];
    t = pi/2 + alpha;
    Rrr = [cos(t),-sin(t);sin(t),cos(t)];
    xyr = diameter*Rrr;
    X2 = xyr(1);
    Y2 = xyr(2);
    leng = param_length-param_diameter;

    if strcmp(state, 'init'),
        state = 'move_line1';
        init_position = position(1:2);
        init_orientation = orientation(3);
```

```
        xy0 = position(1:2);
        xy0(3) = orientation(3);
        fflag = 0;

    elseif strcmp(state, 'move_line1'),
    [vel_local, rVel, fflag]= line(X,Y,alpha,init_position,init_orientation,position,orientation(3),leng);
    forwBackVel = vel_local(2);
    leftRightVel = vel_local(1);
    rotVel = rVel;
    if fflag == 1
        r = [0 (param_diameter/2)];
        d = [0 param_diameter];
        theta = pi + alpha;
        Rr = [cos(theta),-sin(theta);sin(theta),cos(theta)];
        rr = r*Rr + position(1:2);
        dr = d*Rr + position(1:2);
        circle_center =[rr dr];
        xy0 = position(1:2);
        xy0(3) = orientation(3);
        state = 'move_circle1';
    end

    elseif strcmp(state, 'move_circle1'),
        [vel_local,rVel, fflag] = movecircle(circle_center,xy0,alpha, position, (param_diameter/2), orientation(3),xy0(3));
        leftRightVel = vel_local(1);
        forwBackVel = vel_local(2);
        rotVel = rVel;
        if fflag == 1
           xy0 = position(1:2);
           xy0(3) = orientation(3);
           state = 'move_line2';
        end

```

```
    elseif strcmp(state, 'move_line2'),
        [vel_local, rVel,fflag]= line(X2,Y2,(pi+param_angle),xy0,xy0(3),position, orientation(3),leng);
        leftRightVel = vel_local(1);
        forwBackVel = vel_local(2);
        if fflag == 1
           r = [0 (param_diameter/2)];
           d = [0 param_diameter];
           theta =  alpha;
           Rr = [cos(theta),-sin(theta);sin(theta),cos(theta)];
           rr = r*Rr + position(1:2);
           dr = d*Rr + position(1:2);
           circle_center =[rr dr];
           xy0 = position(1:2);
           xy0(3) = orientation(3);
           state = 'move_circle2';
        end

    elseif strcmp(state, 'move_circle2'),
        [vel_local,rVel, fflag] = movecircle(circle_center,xy0,alpha, position, (param_diameter/2), orientation(3),xy0(3));
        leftRightVel = vel_local(1);
        forwBackVel = vel_local(2);
        rotVel = rVel;
        if fflag == 1
           xy0 = position(1:2);
           xy0(3) = orientation(3);
           state = 'move_line1';
        end

    elseif strcmp(state, 'finish'),
        finish = true;
        disp('finished');
    else
    end
```

```
end

function [vel_local, rotVel, fflag] = line(X,Y,fi,init_position,init_orientation,position,orientation,dist)
    fflag = 0;
    fi = (pi/2)-fi;
    forwBackVel = Y;
    leftRightVel = X;
    %Control velocity
    v_c_l = sqrt(forwBackVel^2 + leftRightVel^2);
    max_vel = 1;
       if v_c_l > max_vel
           f = max_vel / v_c_l;
           v_lim = f * [forwBackVel;leftRightVel];
           lrVel = v_lim(1);
           fbVel = v_lim(2);
       else v_c_l < max_vel
           lrVel = forwBackVel;
           fbVel = leftRightVel;
       end
   %Control angular velocity
       fi_error = fi - orientation;
       u_f = 2 * fi_error;
       if u_f > 1
           u_f = 1;
       elseif u_f < 1
           u_f = .8;
       end
   %Frame transformation
       theta = orientation;
       R_G_L = [cos(theta),-sin(theta);sin(theta),cos(theta)];
       R_L_G = R_G_L';
       vel_global = [fbVel; lrVel];
       vel_local = R_L_G * vel_global;
       rotVel=u_f;
```

```
   %Convert fi into 360
       fi_c = orientation -init_orientation;
          if fi_c < -3.12
             fi_c = fi_c + (2*pi);
          elseif fi_c > 3.12
             fi_c = fi_c - (2*pi);
          end

   %End condition #1 translation finish first
   len = sqrt((position(1) - init_position(1))^2 + (position(2) - init_position(2))^2);
       if len > dist
           vel_local = [0 0];
          if fi_c >  fi
             fflag = 1;
          end
       end
   %End condition #2 rotation finish first
        if fi_c > fi
           rotVel=0;
           if len > dist
              fflag = 1;
           end
        end
end

function [vel_local,rotVel, fflag] = movecircle(center_circle,xy,alp ,position,dest_r,orientation,init_orientation)
   fflag = 0;
   center_circle = center_circle(1:2);
   P_v_r = 2;
   Max_v_r = 1;
   Max_v_t = 1;
   center_circle_position = center_circle - position(1:2);
   u = sqrt(center_circle_position(1)^2 + center_circle_position(2)^2);
```

```
    dir = center_circle_position / u;
    R_err = u - dest_r;
    pos = R_err * P_v_r;
    v_r = max(min(pos, Max_v_r), -Max_v_r);
    v_t = Max_v_t;
    vel_global = (v_r * dir) + (v_t * dir * [0 -1; 1 0]);
    theta = orientation;
    R = [cos(theta), -sin(theta); sin(theta), cos(theta)];
    vel_local = vel_global * R;
    rotVel = 1;
    beta = atan(center_circle_position(2)/center_circle_position(1));
    alpha = atan((center_circle(2)-xy(2))/(center_circle(1)-xy(1)));
    angle = (beta-alpha);

    fi = orientation - init_orientation;
    if fi < 0
        fi = fi + (2*pi);
    end

    if fi > (pi)
        rotVel = 0;
    end

    if angle < 0
        angle = angle + pi;
    end

    if angle > 3.1241
       fflag=1;
    end
end
```

We know that are 2 kind of trajectories, straight line trajectory and half circle trajectories, from the previous task we have a function that follows the linear trajectory given the desire X,Y point. In this case we can calculate this desire X,Y points. From the previous image we can observe that this X,Y points are given by:

$$X = -\cos(\alpha) * h$$

$$Y = \sin(\alpha) * h$$

On line 117, the function is doing the same as the task 1. given desired points (X,Y) the function calculate the forwBackVel and leftRightVel values. The function controls the linear velocity (limited at maximum 1) and the rotation velocity (limited to at 1). To finalize this function we 2 conditions the fist one is that the distance traveled on X,Y is proper, and the second is the rotation. The variable 'dist' is used to finalize stop the linear velocity, we know that we need to travel 'dist' distance (param_length − param_diameter) so we need to fullfill this condition to finish:

$$dis > \sqrt{current\_position - initial\_position}$$

Once the distance traveled by the robot is equal to the desired, and the robot is rotated to the desired angle, one 'flag' called fflag is set to 1, the one will finalize the current state and will move the FSM to the next state, the finalization of the current state include calculate two new points, knowing that the next state is half circle trajectory we need to calculate the (X,Y) location of the circle.

On line 175, the function as task 2 rotates around given (X,Y) and is counting the degrees between the current position and the original position once this value is bigger than 179.5 (a value really close to 180) the flag 'fflag' is set to 1 finalizing the current state. The finalization of this state calculate the (X,Y) location of the next state and saves the current position and orientation to be used as initial position and orientation of the next state

4 Result

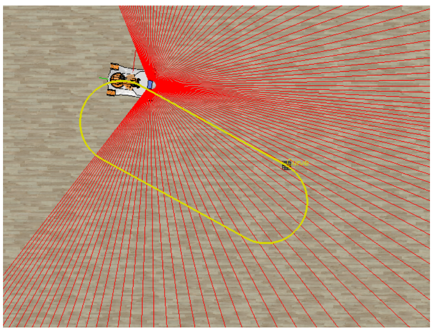

Figure 3: Diameter = 1, Length = 3, Angle = 0.5

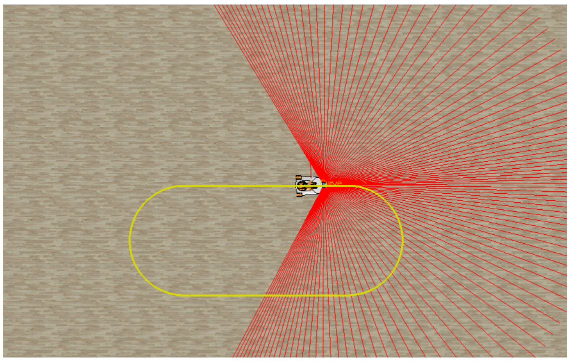

Figure 4: Diameter = 1, Length = 3, Angle = 0