Warsaw University of Technology

FACULTY OF
POWER AND AERONAUTICAL ENGINEERING

Mobile Robots Tutorial (Task 6)

Path planning with wavefront planner

Group Members:

Ahmed Yesuf Nurye (330148)

Taye Tsehaye Alamrew (330292)

May 14, 2023.

Objective

The main objective of this task is to use a wavefront planner to generate a collision free path to a goal pose and execute this path while meeting given task requirements.

Assumption:

- Assume maximum linear velocity, u_max, and angular velocity, ω_max, that the robot can achieve to be $10m/s$ and $10rad/sec$ respectively.
- The Proportional gain for the linear velocity control, Kp_linear, is taken to be 20, for the angular velocity, $Kp\_angular$, is 10.

Implementation:

The main steps in the implementation are given below.

Step 1: Planning

1.1. Reading the map and transforming the *start and goal positions* from environment (real) to map coordinate.

Transforming from real to map is done using:

```
function on_map = env2map(on_env, size_map, size_env)
```

1.2. Use wavefront algorithm to get a new map containing wave connecting the start pos and goal pos on the map.

```
function map = wavefront(start_on_map, goal_on_map, map, debug)
```

Detailed explanation of wavefront implementation.

- Define cost of travel between adjacent cells. Cost4 can also be used.

```
cost8 = [sqrt(2) 1 sqrt(2);
         1       0       1;
         sqrt(2) 1 sqrt(1)];
```

- Enlarge obstacles so that the robot will have enough clearance from the wall. 7 is chosen with trial and error; however, if the size of the robot is known `env2map()`can be used.

```
map = imerode(map, ones(7));
```

- Set wall as NaN and floor as Inf

```
map(map ~= 0) = Inf;
map(map == 0) = NaN;
```

- Set the cost of start location to 0 and make a copy of the map.

```
map(start_on_map(1), start_on_map(2)) = 0;
map_copy = map;
```

- Use for loop to iterate over the cells and set their cost.

$$neighbor\_rows = max(row - 1, 1){:}min(row + 1, size\_map(1));$$

$$neighbor\_cols = max(col - 1, 1){:}min(col + 1, size\_map(2));$$

$$cost = map\_copy(row, col) + cost8(neighbor\_rows - row + 2, neighbor\_cols - col + 2);$$

Take minimum one

$$map(neighbor\_rows, neighbor\_cols) = min(map(neighbor\_rows, neighbor\_cols), cost, \quad 'includenan');$$

- Exit the loop when the cost of the goal location is calculated

```
if ~isinf(map(goal_on_map(1), goal_on_map(2)))
    break;
end
```

1.3 Backpropagate path from the goal to the start position

```
function path = backpropagate(goal_on_map, map, debug)
```

- Start by finding the row and column index of the goal cell and work our way out to the start point.

```
row = goal_on_map(1);
col = goal_on_map(2);
```

- Initialize the path and use loop until the start pos in not reached

```
path = [];
while map(row, col) ~= 0
```

- Find neighbor indices of the current cell

```
neighbor_rows = max(row - 1, 1):min(row + 1, numrows(map));
neighbor_cols = max(col - 1, 1):min(col + 1, numcols(map));
```

- Find cell with lower cost

```
[~, idx] = min(map(neighbor_rows, neighbor_cols), [], 'all', 'linear');
[row_idx, col_idx] = ind2sub([3 3], idx);
```

- Update row and column to move to next cell

```
row = row + row_idx - 2;
col = col + col_idx -2;
```

- If we encounter a local minimum that is not the goal, raise an error

```
if isinf(map(row, col))
    error('Could not reach the goal')
end
```

- Repeat this until start pos is reached.

Now the planning is done we can move to the second stage which is executing the path.

2. Executing the planned path.

For this we have used the algorithm for task 1. We also controlled the orientation of the robot so that it faces the direction of motion.

2.1 If there are multiple points on the path, we move through them at max speed.

$$diff\_pos = path(end,:) - pos;$$
$$norm\_diff = norm(diff\_pos);$$
$$vel = u\_max;$$

Upon reaching a certain cell it will be removed from the path and the motion to the next cell will be executed. Reaching a cell is determined using the through tolerance.

2.2 If there is only one point on the path, i.e., the goal position, we move to it in a regulated manner.

```
if norm_diff < goal_tol
        Pl = Kp_linear * norm_diff;
        vel = max(min(Pl, u_max), -u_max);
```

Or if the goal is reached then we stop the robot and perform a clean exit.

```
        leftRightVel = 0;
        forwBackVel = 0;
        rotVel = 0;
        finish = true;
        return;
```

Now that we have the magnitude of the velocity, we can determine the velocity by multiplying it with the direction.

$$direction = diff\_pos/norm\_diff;$$

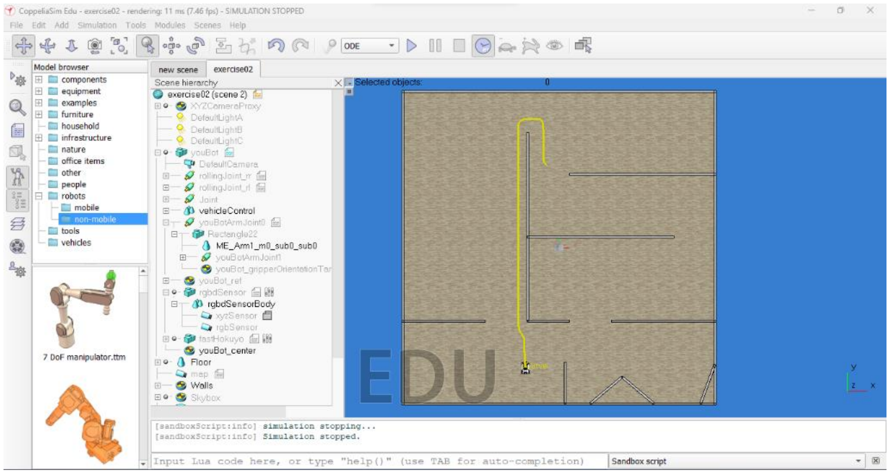
$$u\_global = vel * direction;$$
$$u\_local = global2local(u\_global, phi);$$
$$leftRightVel = u\_local(1);$$
$$forwBackVel = u\_local(2);$$

Rotational velocity control. We want the robot to face the direction of movement.

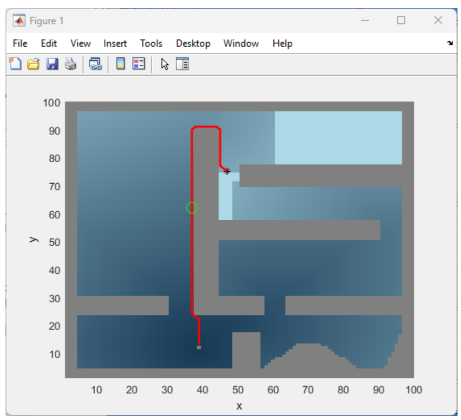
$$diff\_phi = angdiff(atan2(diff\_pos(2), diff\_pos(1)), phi - pi/2);$$
$$Pr = Kp\_angular * diff\_phi;$$
$$rotVel = max(min(Pr, w\_max), -w\_max);$$

Simulation result

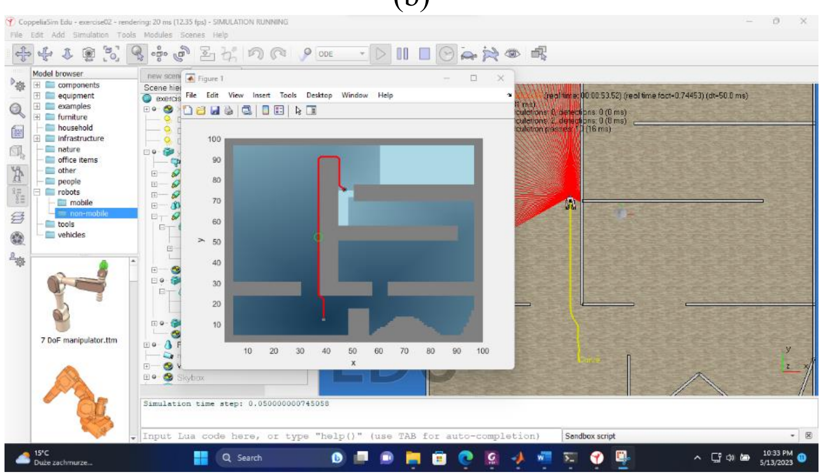
- `run_simulation(@solution6, false, [-0.5, 3.8], "map.png")`



(a)



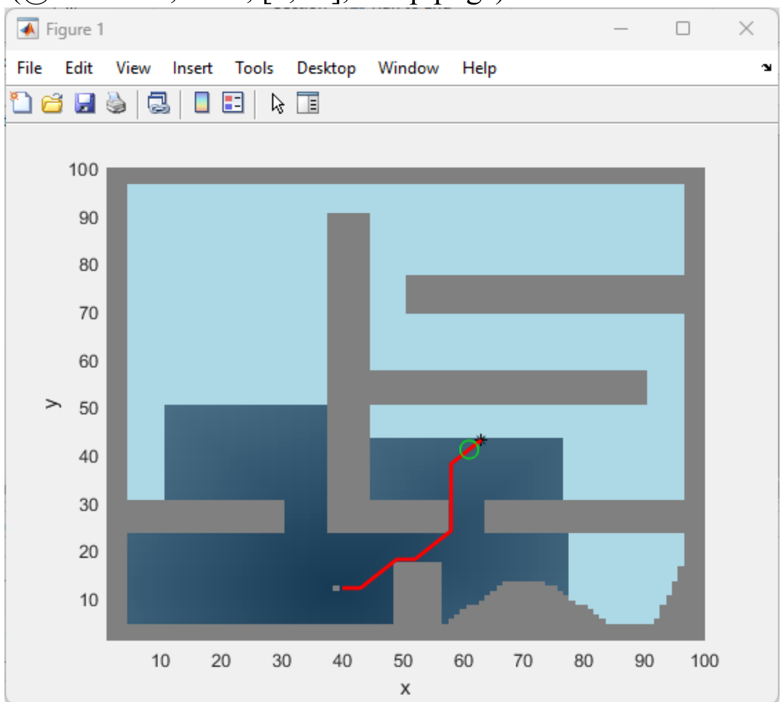
(b)



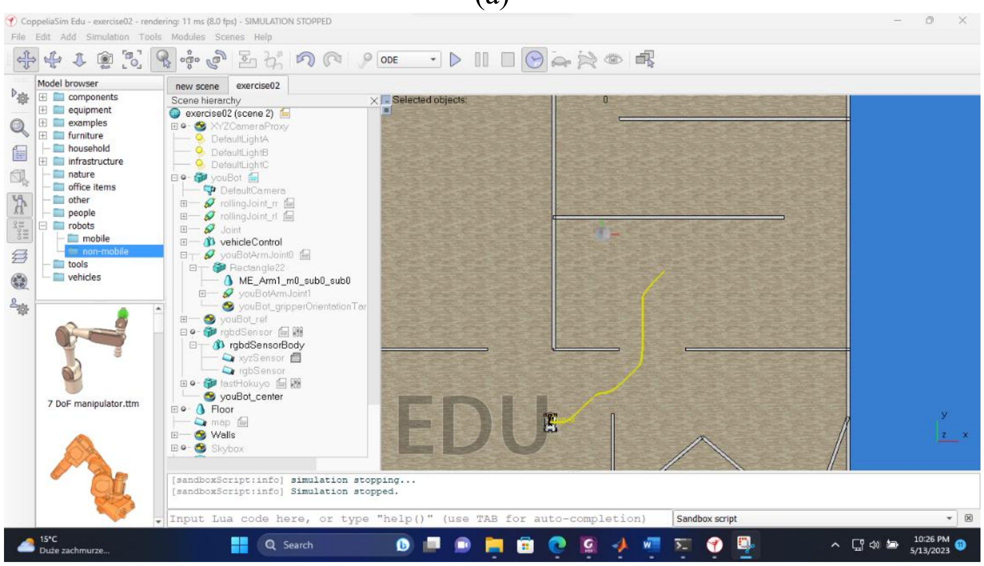
(c)

Figure 1. (a) Robot motion (b) debug plot of wave front and robot motion (c) both.

- run_simulation(@solution6, false, [2, -1], "map.png")



(a)



(b)

Figure. (a) debug plot and (b) robot motion.