Warsaw University of Technology

FACULTY OF
POWER AND AERONAUTICAL ENGINEERING

Mobile Robots Tutorial (Task 5)

Navigation With Bug2 Algorithm

Group Members:

Ahmed Yesuf Nurye (330148)

Taye Tsehaye Alamrew (330292)

April 30, 2023.

Objective

Enable the YouBot navigate to given goal position using Bug2 algorithm while meeting given task requirements.

Assumption:

- Assume maximum linear velocity, u_max, and angular velocity, ω_max, that the robot can achieve to be $5m/s$ and $5rad/sec$ respectively.
- The Proportional gain for the linear velocity control, Kp_linear, is taken to be 2, for the angular velocity, $Kp\_angular$, is 10 and for following goal line we take a separate gain, $Kp\_goal = 20$.

Implementation:

The main steps in the implementation are given below.

Step 1: Finding the goal line, l. Note l is persistent variable and defined only in the initial state.

We can define l by finding its slope and y intercept, $l = [m, b]$, as follows:

$$l(1) = (goal\_position(2) - current\_pos(2)) \, / \, (goal\_position(1) - current\_pos(1))$$

$$l(2) = goal\_position(2) - l(1) * goal\_position(1)$$

Step 2: Processing LIDAR readings and computing goal direction and distance from l.

LIDAR data processing is already done in Task 4, for convenience it is repeated here.

We need to find the shortest distance from the wall and the direction to that distance. To do so using polar coordinate is much more suitable for this problem as we are going to deal with magnitude of a distance and its direction (i.e., angle). Therefore, the LIDAR readings are converted into polar form as follows.

$$[theta, distances] = cart2pol(pts(1,:), pts(2,:))$$

An alternative to this approach is to calculate the Euclidean distance and then its direction.

The front of the robot is rotated by 90^0 we need to take that into consideration when calculating theta.

$$theta = theta + pi/2$$

Then we find contacts of locally minimum distances with minimum prominence of 0.2.

$$desired\_contacts = islocalmin([5 \; distances \; 5], 'MinProminence', 0.2)$$

$$desired\_contacts = and(desired\_contacts(2{:}end - 1), contacts)$$

The maximum distance that can be measured by the LIDAR is almost 5m. That is why we put 5 at the beginning and end of distances array. If $desired\_contacts$ is empty, we move forward else we calculate the minimum distance.

The distance to line l will be used to determine if we are crossing l and used in changing state from wall following to goal depending on d (distance from obstacle to goal) ad is computed as follows:

$$ql(1) = (current\_pos(1) + l(1) * (current\_pos(2) - l(2))) \,/\, (l(1)\char`^2 + 1)$$

$$ql(2) = ql(1) * l(1) + l(2)$$

$$diff\_ql = ql - current\_pos$$

$$dist\_line = norm(diff\_ql)$$

And the goal direction is computed as:

$$diff\_goal = goal\_position - current\_pos$$

$$dist\_goal = norm(diff\_goal)$$

$$goal\_dir = angdiff(atan2(diff\_goal(2), diff\_goal(1)), phi - pi/2)$$

Step 3: Determining conditions for changing state.

- If the distance from the goal is less than $tolerance = 0.1$, then $state = stop$
- If the robot is at a distance which is less than or equal to desired distance from the wall, then $state = wall$, for wall following.

$$if\ any(distances(desired_{contacts}) \leq desired_{distance}) then\ state = wall$$

- If the robot is crossing line l and the distance to the goal is less than d, then $state = goal$

$$[\sim, idx] = min\left(abs\left(angdiff(theta, goal\_dir)\right)\right)$$

$$if\ (distances(idx) > 2 * desired\_distance)\ \&\&\ \left(dist_{goal} < d\right)\ then\ state = goal$$

- One additional condition is that when the robot is trapped. If we are back to where d is measured, then we should indicate an error message.

$$if\ norm(oc - current\_pos) < tolerance\ then\ robot\ is\ trapped$$

Step 3: Calculating motion commands for each state.

For moving along line l towards the goal

Direction to the closest point on the goal line

$$radial = angdiff(atan2(diff\_ql(2), diff\_ql(1)), phi - pi/2)$$

Determine the movement along that line, towards positive or negative.

$$option = sign(angdiff(goal\_dir, radial))$$

Error for proportional control

$$diff\_dist = dist\_line$$

Face the robot towards the goal

$$goal\_phi = -option * pi/2$$

For wall following: This is the same as Task 4. For convenience it is repeated here.

First, we find the indices of the local minimum distances then we sort them. After that we select the minimum distance which is basically the distance at the first index of the sorted distances.

$$local\_min\_idxs = find(desired\_contacts)$$

$$[\sim, idxs] = sort(distances(desired\_contacts))$$

$$min\_idx = local\_min\_idxs(idxs(1))$$

Now, we have the index, min_idx, of the minimum distance. We can use it to calculate the error as follows.

$$diff\_dist = distances(min\_idx) - desired\_distance$$

The radial direction can simply be found by indexing theta with min_idx. This is the advantage of using polar coordinates.

$$radial = theta(min\_idx)$$

Step 4: Velocity command calculation.

First let's determine the tangential direction which is perpendicular to radial direction.

$$tangential = radial + option * pi/2$$

Where $option = 1\ or - 1$, and represents the direction of tangential movement.

Weight of radial movement depends on the error from the wall.

$$Pl = Kp\_linear * diff\_dist$$

Limit Pl and then the over all direction of movement can be found as:

$$direction = Pl * radial + tangential$$

Magnitude of velocity (limited), vel, is:

$$vel = max(min(Kp\_goal * dist\_goal, u\_max), -u\_max)$$

The direction is computed with respect to the local frame. Therefore, there is no need to transform it from global to local. However, we need to resolve the velocity into left-right and forward-backward, which is done with the help of $local\_velocity(vel, direction)$ function.

$$u\_local = u\_max * [sin(direction) \; -cos(direction)]$$

$$leftRightVel = u\_local(1)$$

$$forwBackVel = u\_local(2)$$

The proportional controller is already implemented in step 3 while calculating direction.

For the rotational direction, we want to make the radial direction equal to $goal\_theta$.

Therefore, the rotational velocity is proportional to:

$$diff\_theta = angdiff(radial, goal\_theta)$$

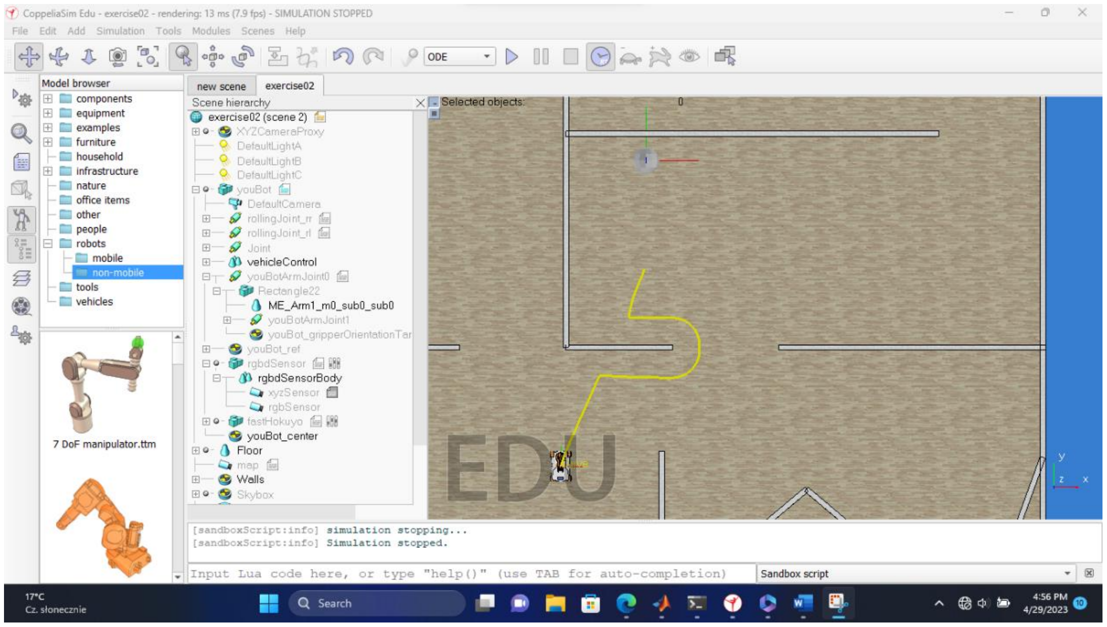
$$Pr = Kp\_angular * diff\_theta$$

Limit the output of proportional controller

$$rotVel = max(min(Pr, w\_max), -w\_max)$$

Simulation result

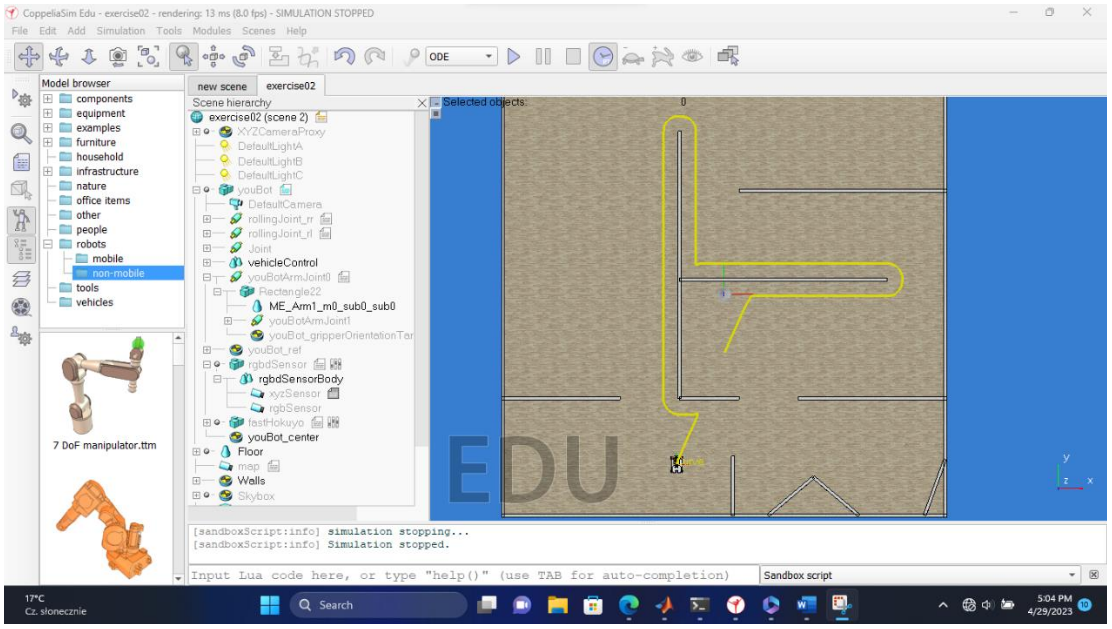
- run_simulation(@solution5, false, [0 -2]). This is result for a goal that can be reached. The goal coordinate of course is given with respect to the global frame.



```
[sandboxScript:info] simulation stopping...
[sandboxScript:info] Simulation stopped.
```

```
Input Lua code here, or type "help()" (use TAB for auto-completion)
```

(a)



```
[sandboxScript:info] simulation stopping...
[sandboxScript:info] Simulation stopped.
```

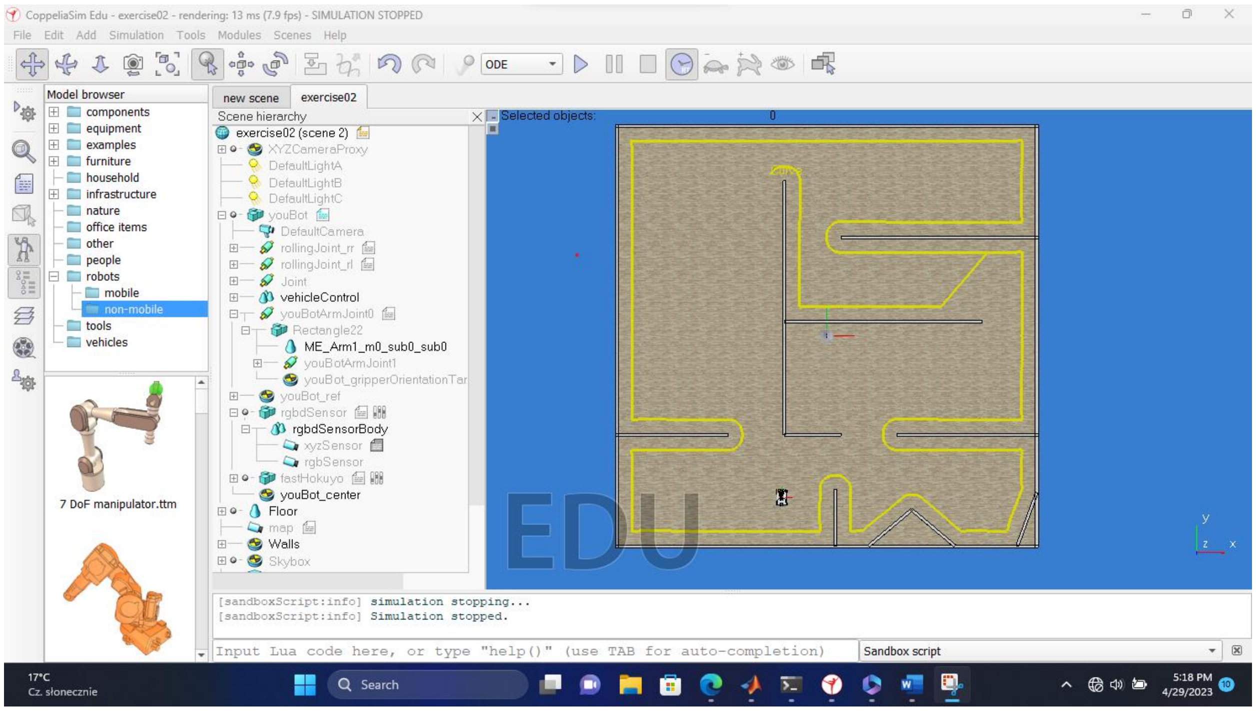
```
Input Lua code here, or type "help()" (use TAB for auto-completion)
```

(b)

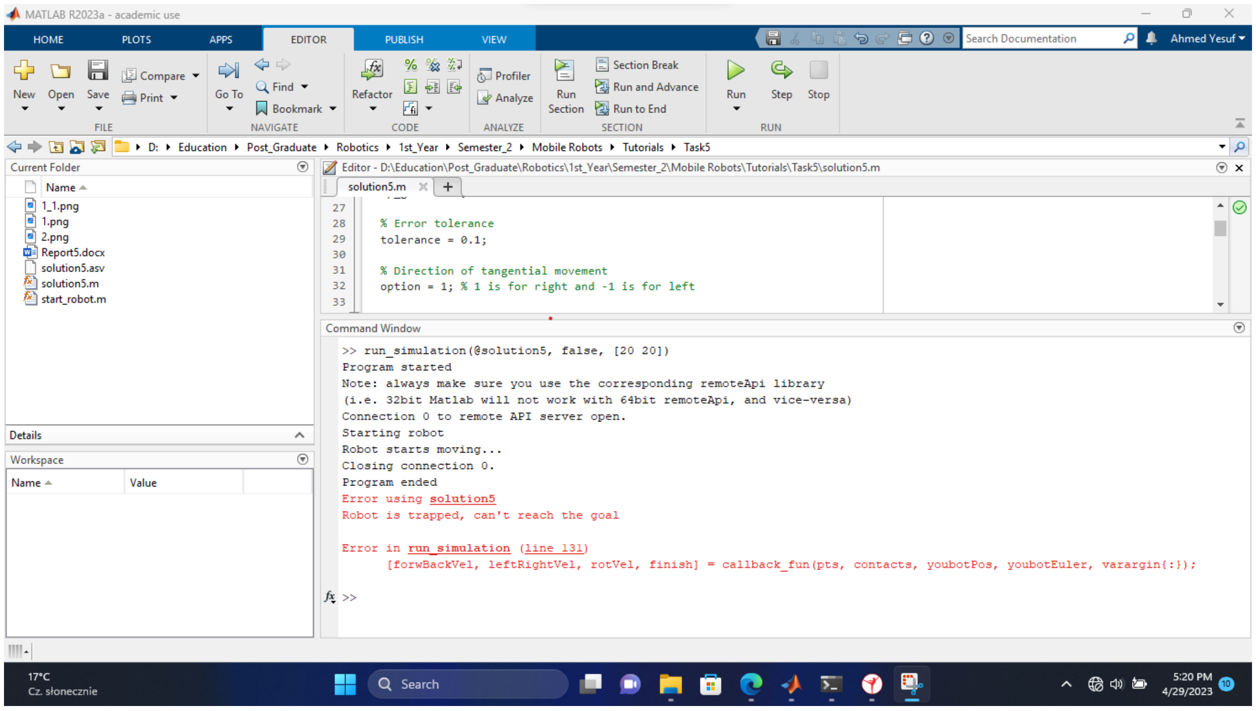
Figure 1: Result for a goal that can be reach. (a) option= -1 and (b) option=1 respectively.

Now, let’s consider a goal which is outside of the environment something like [20, 20].

- run_simulation(@solution5, false, [20 20])


CoppeliaSim Edu - exercise02 - rendering: 13 ms (7.9 fps) - SIMULATION STOPPED
Model browser
Scene hierarchy
Selected objects: 0

```
[sandboxScript:info] simulation stopping...
[sandboxScript:info] Simulation stopped.
```


MATLAB R2023a - academic use
Editor - D:\Education\Post_Graduate\Robotics\1st_Year\Semester_2\Mobile Robots\Tutorials\Task5\solution5.m
Command Window

```
% Error tolerance
tolerance = 0.1;

% Direction of tangential movement
option = 1; % 1 is for right and -1 is for left
```

```
>> run_simulation(@solution5, false, [20 20])
Program started
Note: always make sure you use the corresponding remoteApi library
(i.e. 32bit Matlab will not work with 64bit remoteApi, and vice-versa)
Connection 0 to remote API server open.
Starting robot
Robot starts moving...
Closing connection 0.
Program ended
Error using solution5
Robot is trapped, can't reach the goal

Error in run_simulation (line 131)
      [forwBackVel, leftRightVel, rotVel, finish] = callback_fun(pts, contacts, youbotPos, youbotEuler, varargin{:});
```

Figure 2. Result for a goal that can’t be reached.