Motion planning for a mobile robot

Robotics instructional lab #6

Spring 2022

In this lab, you will plan a path for a mobile robot to move on a plane near obstacles and reach a desired goal.

1 Background

During the lab, you will be given a two-wheel mobile robot seen in Figure 1. The robot will move on a large table between cubic obstacles. A simple camera is positioned on the ceiling observing the table. The camera identifies ArUco[1] markers on the robot, obstacles and reference marker on the table. Each marker has a unique ID number for distinction.

2 Problem

A marker is fixed on the table and represents the *base frame*. The robot is initially positioned at $\mathbf{p}_c \in \mathbb{R}^2$ relative to the base frame. Similarly, we have a set of k obstacles $\mathcal{O} = \{\mathbf{o}_1, \dots \mathbf{o}_k\}$ where $\mathbf{o}_i \in \mathbb{R}^2$ is the position of obstacle i. The robot must move to a goal position $\mathbf{p}_g$ without collisions. It is required to plan a collision-free path from $\mathbf{p}_c$ to $\mathbf{p}_g$ for the robot to follow. Illustration of the problem can be seen in Figure 2.

3 Prerequisites

1. Write a function `planner`($\mathbf{p}_c,\ \mathbf{p}_g,\ \mathcal{O},\ B,\ \delta$) that receives
   - $\mathbf{p}_c$ - Current robot position relative to the base frame.
   - $\mathbf{p}_g$ - Goal position relative to the base frame.
   - $\mathcal{O}$ - Set of obstacle positions represented relative to the base frame.
   - B - Four points in $\mathbb{R}^2$ relative to the base frame describing the vertices of the rectangular bound of the robot. The robot must move only within the rectangle described by these points.
   - δ - Required path resolution.

[1] ArUco markers: `https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html`

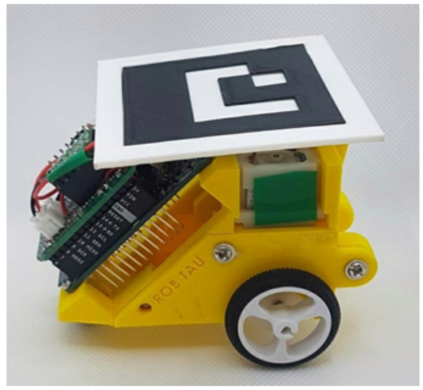

Figure 1: Mobile robot to be used in the lab.

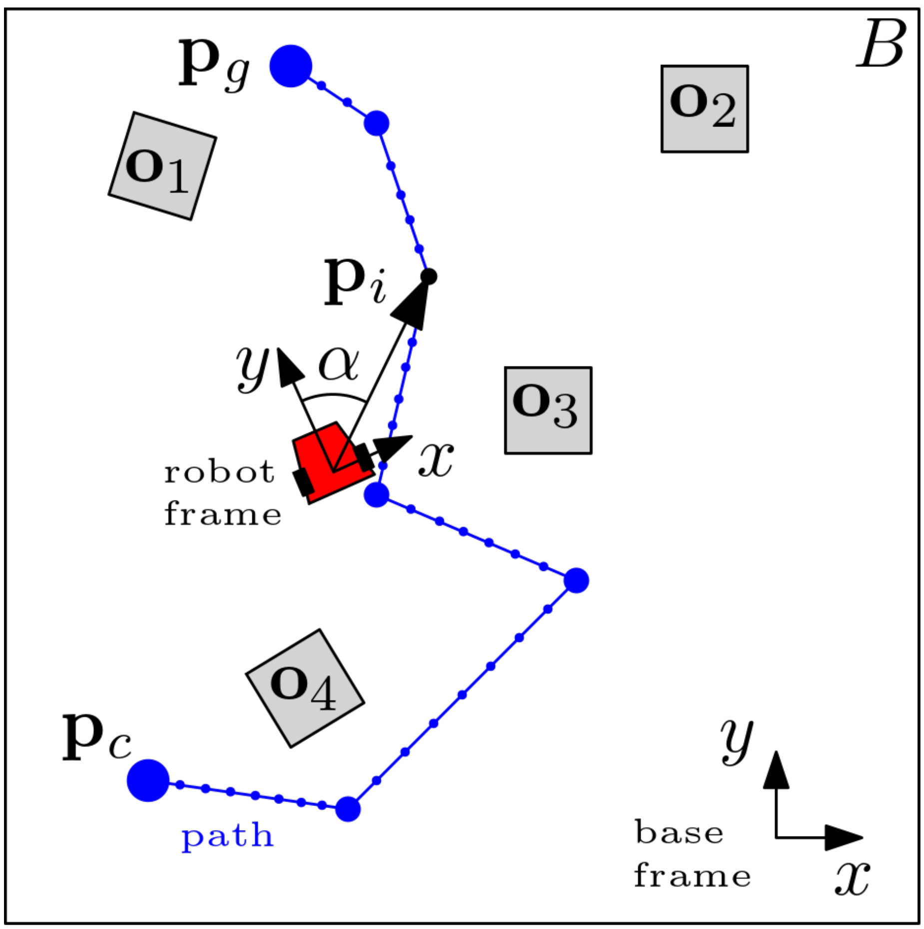

Figure 2: Sketch of the robot setup.

The function will output a path represented by a sequence of positions in the form $\{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_N\}$ such that $\mathbf{p}_1 = \mathbf{p}_c$, $\mathbf{p}_N = \mathbf{p}_g$ and $\|\mathbf{p}_i - \mathbf{p}_{i+1}\| \leq \delta$ for any $i = 1, \ldots, N-1$. Some remarks:

(a) All points in the path should be represented with respect to the base frame.

(b) Although not mandatory, it is recommended to plan the path using the Rapidly-exploring Random Tree (RRT)[2]. However, using potential functions is **not** allowed.

(c) It is recommended to visualize the output of the function on a 2D plot for verification and debugging.

(d) Bonus: Plan the shortest path to the goal.

2. For the controller to correctly track the path, we must compute the steering angle towards the next intermediate point on the path.
   Write a function `steering_angle(`A^{cam}_{robot}, A^{cam}_{base}, $\mathbf{p}_i^{(cam)}$`)` that receives

   - A^{cam}_{robot} - Homogeneous transformation matrix mapping pose from the robot coordinate frame to the camera frame.
   - A^{cam}_{base} - Homogeneous transformation matrix mapping pose from the base coordinate frame to the camera frame.
   - $\mathbf{p}_i^{(base)}$ - Position of a point on the path represented in base coordinate frame.

   Matrices A^{cam}_{robot} and A^{cam}_{base} are extracted by the camera. The function will return steering angle α for the robot to turn and position of a point in the path represented in robot coordinate frame. Note that the angle is relative to the heading vector (y-axis in the robot frame).
   Hint: Represent point $\mathbf{p}_i$ in the robot frame. Input and output example:

```
#Input:
A_cam_robot = np.array([[-0.1614, -0.6982, 0.6975, 0.412],
                        [0.9769, -0.0127, 0.2133, 0.218],
                        [-0.1400, 0.7158, 0.6841, 0.797],
                        [0., 0., 0., 1.]])
A_cam_base = np.array([[-0.7537 , 0.6208, 0.2157, 0.112],
                       [-0.1910, -0.5292, 0.8246, 0.801],
                       [0.6261, 0.5783, 0.5230, 0.797],
                       [0., 0., 0., 1.]])
p_i_base = np.array([0.5, 1.1, 0.]) # Notice that
# the z index is irrelevant, so it should always
# be set to 0 (or another random value) for future
# calculations.

# Output:
p_i_car = np.array([-0.22, 0.676]) # returns only x,y  coordinates.
alpha = -18 # Angle in degrees
```

Prerequisites are a mandatory in order to carry out the lab .

[2] RRT explained:
`http://lavalle.pl/rrt/`
`https://bit.ly/3Mvx4Ma`
`https://bit.ly/37K17AJ`

4 Lab instructions

1. Update file **lab_06.py** with your functions.
2. While observing the output of the camera, move the robot across the table to find the four points in B having the largest bounding rectangle visible by the camera.
3. Repeat twice: randomly position the robot and obstacles within the bound. Plan and roll-out a path to a chosen goal position. Record the rolled-out path.
4. Ask the instructor to re-position the robot and obstacles, and define a goal. Plan and roll-out a path. Record the rolled-out path.

5 Report requirements

The lab report should include the following:

1. Explain the operation of your motion planner.
2. Describe the process of the lab.
3. Include the following:
   - A table summarizing all results.
   - For each path, show a plot with the planned and real (recorded during the experiment) paths along with the positions of the obstacles.
   - Compute the mean tracking error along each path.
   - Explain the reasons for the errors and how could they be improved.
4. If the bonus was solved, explain how the shortest path was found.
5. Provide a summary for your results.