Planning Algorithms in AI
PS2: Sampling-based Planning

Gonzalo Ferrer
Skoltech

November 18, 2022

This problem set has two tasks and is individual work. You are encouraged to talk at the conceptual level with other students, discuss on the equations and even on results, but you may not show/share/copy any non-trivial code.

Submission Instructions

Your assignment must be submitted by 11:59pm on **November 30, Wednesday**. You are to upload your assignment directly to Canvas as **two** attachments:

1. A `.tar.gz` or `.zip` file *containing a directory* named after your uniquename with the structure shown below.

   ```
   yourname_ps2.zip:
   yourname_ps2/all_provided_code_file.py
   yourname_ps2/data.pickle
   yourname_ps2/any_your_code_file.py (or .ipynb)
   yourname_ps2/solve_4R.mp4 (or .gif)
   ```

2. A PDF with the written portion of your write-up. Scanned versions of hand-written documents, converted to PDFs, are perfectly acceptable (reduced size). No other formats (e.g., .doc) are acceptable. Your PDF file should adhere to the following naming convention: `yourname_ps2.pdf`.

Homework received after 11:59pm is considered late and will be penalized as per the course policy. The ultimate timestamp authority is the one assigned to your upload by Canvas. No exceptions to this policy will be made.

Task 1: Visualization (20 points)

For this task, you will write a Python script or notebook, please make sure all your code is well structured, understandable and reproducible.

The Workspace is a continuous 2D area: $\mathcal{W} = \mathbb{R} \times \mathbb{R}$ with obstacles. The target object is the 2D manipulator-like mechanism which consist of 4 links, with "root" joint at (0, 0). The configuration of the manipulator is expressed as $\mathbf{q} = [\theta_1, \theta_2, \theta_3, \theta_4]$, where $\theta_i \in (-180; 180]$ is the angle of the link relative to its joint in degrees. Both angles and coordinates on the plane are continuous variables. Note that intersection of the links is allowed (imagine that all links are in different planes).

All the required data can be obtained from `PS2_data.pickle`. See `main.py` to check how to load the data. The pickle file consist of the dictionary with the following fields:

- `start_state` and `goal_state` are the `[4,]` arrays with the starting configuration and the target configuration to reach
- `obstacles` is the `[6, 3]` array of the circular obstacles in format `[x, y, radius]`
- `collision_threshold` is the float number that defines minimal allowed distance to the obstacle

To see how display the manipulator in desired state, check `environment.py` file.

A. (10 pts) Visualize the manipulator in the start state and target state. Comment on your thoughts about comparison the discretized orientation space from PS1 vs continuous orientation space in current problem set.

B. (10 pts) Visualize the manipulator in 4 random orientations that include both colliding and non-colliding configurations. Check what does the `ManipulatorEnv.check_collision` function returns for those configurations. Comment on your observations.

Task 2: RRT (80 points)

For this task, you will implement the RRT algorithm to solve the path planning for the 4R manipulator. Follow the material explained in class

A. (40 pts) You need to implement the RRT algorithm for agent in continuous domain. The starting configuration of the agent is `(0, 0, 0, 0)` and the goal configuration is `(-180.0, -60.0, 72.0, -60.0)`. For searching nearest pose use L_1 `distance` between two configuration vectors:

$$distance = \|\mathbf{q_2} - \mathbf{q_1}\|_1 \tag{1}$$

You may want to use function `angle_difference` and `angle_linspace` from `angle_utils.py`. As a maximum allowed rotation for each joint we suggest you to use 10 degrees.

Save the result of calculated plan in `solve_4R.mp4` using `plotting_result` function from PS1(make the necessary changes to the function) or `solve_4R.gif`. If you want to add more smoothness to the video, you can add additional intermediate configurations, but it's optional.

You may want to use the function `ManipulatorEnv.check_collision`, in order to check if the the manipulator is in collision at any point towards the new configuration.

Hint: you may want to use the collision check function over a small sequence of configurations, connecting your current one with the new configuration candiate q_{new}

B. (10 pts) Comment on how many states have been visited? What is the final trajectory size? Can you comment on the optimality of the plan? You can also collect some observations and statistics across multiple runs.

C. (15 pts) Try to change weight of rotation in calculation of distance between two agent positions. We suggest you to build a distance function based on weighted sum of the angle distances. Comment on the results.

D. (15 pts) Try to change step size used for RRT branches. Comment on the results.