Final Project—Task inspection planning

1 General guideline

This final project will cover the topic of task inspection planning. Here we are given a robot manipulator performing a task with its end effector (i.e., the robot's path is provided) and we are required to plan a path for a second manipulator that is charged with tracking the first robot's end effector using on-board sensors. Writeups must be submitted as a PDF. LaTeX is preferred, but other typesetting methods are acceptable. The code must be submitted in a zip archive with proper documentation. Plots generated should be included in the writeup. The project should be submitted in pairs.

Submission date is 15.3.2023, end of day.

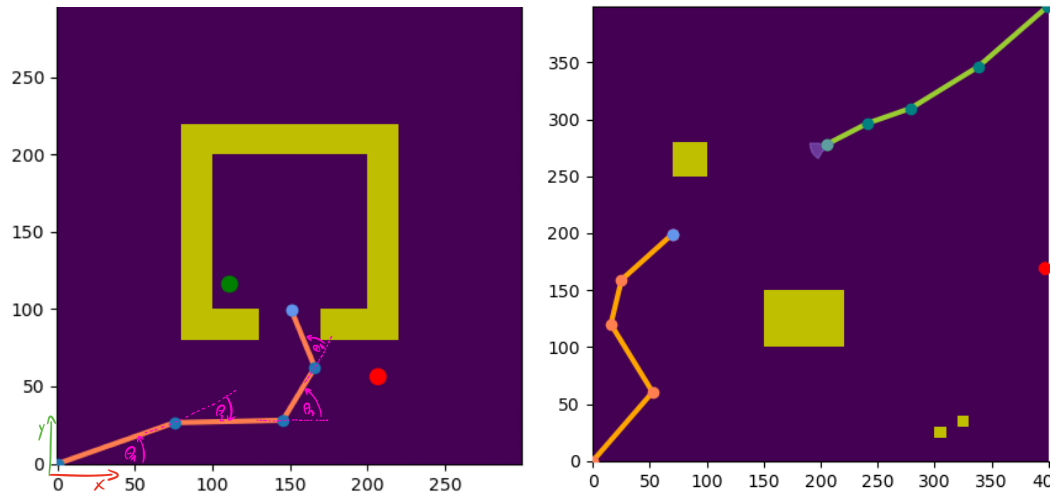
2 Problem description

In the previous assignment, you implemented motion and inspection planning algorithms for a robot manipulator. Here, instead of computing a path to inspect a set of points, your task is to compute a path for an inspecting robot, that will inspect the end effector of a robot performing a task. In other words, your goal is to provide a motion plan that will maximize the amount of time in which the end-effector of the gripping robot is under the field-of-view of the inspecting robot (see Fig. 1b).

Notice that similar to inspection planning, there is no predefined goal here. However, the key difference is that now we need to account for the *time* in which inspection occurs (think how all the algorithms we studied should be altered to account for time...). Similar to the previous exercise, we are dealing with fix-based planar manipulators composed of four links and four joints. We denote the C-space $\mathcal{X} = S^4$ where a *configuration* of a manipulator is defined as a series of four joint angles $q := \langle \theta_1, \ldots, \theta_4 \rangle$.[1]. Similarly, we denote the *state space* $\mathcal{Y} = \mathcal{X} \times \mathbb{R}$ where the last dimension represents the time in when the manipulator is located at a certain configuration. Namely, A *state* s of a manipulator will be in the form $s = \langle q, t \rangle$ with $q \in \mathcal{X}$ denoting a configuration and $t \in \mathbb{R}$ denoting the time. A *trajectory* τ is a *sequence* of configurations, such that each configuration q_i denotes the configuration at timestamp i.

The aforementioned definitions were for a general planar manipulator. Here we will be interested in two manipulators $\mathcal{R}_{\text{task}}$ and $\mathcal{R}_{\text{inspect}}$. The first, $\mathcal{R}_{\text{task}}$, is the robot performing the task using its end effector. This is provided via a trajectory and you have no control of this robot. The second, $\mathcal{R}_{\text{inspect}}$, is the robot performing the inspection using a sensor mounted on its end effector (the specific model is identical to the one used in HW3. Namely, the robot's field of view is bounded by a given angle and range). This is the robot you are tasked with planning a trajectory. More specifically, this trajectory should maximize the time that the end effcetor of $\mathcal{R}_{\text{task}}$ is visible.

[1] Note that joint angles are cyclic which is why $\mathcal{X} = S^4$ and not $\mathcal{X} = \mathbb{R}^4$



(a) (b)

Figure 1: (a) Illustration of joint angles and how forward kinematics works for a robot (either $\mathcal{R}_{\text{task}}$ or $\mathcal{R}_{\text{inspect}}$). (b) Example of task inspection planning in action—$\mathcal{R}_{\text{inspect}}$ is inspecting the end efffector of robot $\mathcal{R}_{\text{task}}$ during motion.

Notice that many of the concepts you developed for HW3 can be re-used here. For example, computing the location of the robot's end effector is done via Forward Kinematics.

3 Code Overview

The starter code is written in Python and depends on numpy, matplotlib, imageio and shapely, as seen in the previous assignment. We recommend you to work with virtual environments for python. If any of the packages are missing in your python environment, please use the relevant command:

```
pip install numpy
pip install matplotlib
pip install imageio
pip install Shapely
```

You are provided with the following files:

- **run.py** - Contains the main function. Note the command-line arguments that you can provide for map file and coverage.
- **Robot.py** - A file containing management classes for both robots. `Robot` represents a general manipulator robot, `GrippingRobot` represents the robot with the given path, and `InspectingRobot` represents the robot that you are in charge to operate.

- **MapEnvironment.py** - Environment management class, containing all functions related to the environment itself, including visualization. Here, `compute_inspected_timestamps_for_edge` is a function that is provided to you to handle inspection. The function receives configurations and timestamps describing an edge, and returns the timestamps in which robot $\mathcal{R}_{\text{task}}$ was inspected from robot $\mathcal{R}_{\text{inspect}}$'s viewpoint.
- **map_plan_p1.json and map_plan_p2.json** - JSON files with details describing maps. Contains map dimensions, (workspace) obstacles, start configuration (in the C-space!), and the name of the file with the path of robot $\mathcal{R}_{\text{task}}$. Notice that if you want to debug your code, you can remove obstacles to ease on your algorithm, but make sure to leave them for the report.
- **plan_mp_p1.txt and plan_mp_p2.txt** - Files containing paths for each of the provided JSON files. No need to link them, the code will link the relevant file according to the given JSON file. Notice that each row represents a single timestamp (e.g., first row is $t = 0$, second row is $t = 1$, and etc.).
- **TaskInspectionPlanner.py** - planner for the inspecting robot. Logic to be filled in by you.
- **utils.py** - Utility functions provided.

The following are two examples of how to run the code:

```
python run.py –map map_plan_p1.json –coverage 0.3
python run.py –map map_plan_p2.json –coverage 0.5
```

In your work you are allowed to add more files or change the structure of the code, but notice that if you want to participate in the competition (detailed below), you need to make sure that your code works under the given requirements.

4 Example

Together with this file and the code you are also provided with a visualization of a solution produced by the a highly-simplistic solver. Here, the orange robot $\mathcal{R}_{\text{task}}$ is the one performing the task using its end effector while the green robot $\mathcal{R}_{\text{inspect}}$ is the one performing the inspection using a sensor mounted on its end effector (visualized by the purple triangle). This simplistic planner produced a coverage of 30%—you are expected much better results!

5 Report

Your task is to fill the empty `plan` function (`InspectionPlanner.py` file) to provide a planner that will return (in the following order):

- `plan` - A numpy array with dimensions $[N, 4]$ containing the path of the inspector robot ($\mathcal{R}_{\text{inspect}}$), where N is the number of steps provided by your planner (without interpolation, this will be provided in the visualization).
- `plan_timestamps` - A numpy array with dimensions $[N]$ containing the timestamps of the plan, where N is the number of steps.
- `path_coverage` - A scalar value denoting the coverage of the path.
- `path_cost` - A scalar value denoting the cost of the path (in C-space).
- `computation_time` - The time it took to compute the path (in seconds).

Notice that, in addition to newly-developed code, you can use any material provided, as well as any material from previous exercises.

Metrics. Here we care about three different (and often conflicting metrics) (i) computation time, (ii) coverage and (iii) length of the inspecting path of $\mathcal{R}_{\text{inspect}}$.

Report. The coverage as a function of the computation time as well as the path length as a function of the coverage. Each plot should contain results averaged over ten different runs. In addition, choose one of the maps and attach three representative visualizations created by `visualize_plan`. One for an initial plan (fast planning times, coverage of 0.3), an intermediate plan (medium planning times, coverage of 0.45) and a final plan (longest planning times, coverage of 0.6).

Discuss. your solution to this problem and explain how did you tackle the need to track a certain end-effector position. Detail the changes you were required to do in order to compute the inspecting plan. Notice that for higher coverage the search may take a few minutes, so think about how you can improve your search! **Hint:** In motion planning we biased the sampling towards a goal vertex. If there is no goal, how can we bias the sampling to improve coverage? Discuss the complexity of the approach as well as the tradeoffs you faced. Initial approaches that did not work but from which you gained insights should be described as well (together with the insights of course).

6 Competition (Bonus to final grade—up to 3 points)

In this project, you will also have the chance to compete in a task-inspection competition. You are encouraged to reduce your computation time and work on the efficiency of your algorithm to output a path that is as-short-as-possible while having high coverage of $\mathcal{R}_{\text{task}}$'s end-effector. If you want to improve given modules, such as collision detection or the sensor function, you are allowed to, but make

sure to keep your favorite values as default. Notice that you will be tested on JSON files that are not provided with the code.

If you want to participate, all you need to do is to **notify** it in your project report, and **make sure** that your code supports the given execution format:

```
python run.py –map map_file.json –coverage 0.5
```

Notice that we want to limit the coverage in the competition so make sure your planner knows to return a path once coverage requirements are met. Make sure you **do not** change the stats saving procedure (the function `write_plan_stats`)! Otherwise, we will not be able to evaluate your results.

The extra points will be credited to your final grade. First, second and third places will receive 3,2 and 1 additional points, respectively.

Good luck!