Algorithmic Robotics
COMP/ELEC/MECH 450/550
Project 1: Getting Familiar with OMPL

The documentation for OMPL can be found at http://ompl.kavrakilab.org.

In this project, you will familiarize yourself with the basics of compiling and running programs that use the Open Motion Planning Library (OMPL). You will run programs that execute basic motion planning queries for a variety of systems, run motion planning queries for 2D and 3D robots in the OMPL.app GUI, and do basic evaluation of sampling-based planners through OMPL's benchmarking interface. Although there is no code to write, it is expected that you install OMPL, inspect, compile and execute demo code, and evaluate interesting planning scenarios using the GUI. Note: the third part of this project may take a while to run. Please start early and plan accordingly.

We will assume in the project documentation that you are using the provided Docker setup. Adapt any statements as necessary to fit how you have installed OMPL.

Project Exercises

0. Launching Docker environment

Install OMPL and OMPL.app. Please refer to the document "Project 0" for how to do this. We highly recommend using the Docker setup for this.

1. Running Example Programs

Compile and execute the example C++ programs in the `Demos` folder.

To do this, create a folder `projects` under the host directory that you mapped to the Docker container in the `docker-compose.yml` file you modified in Project 0. Then simply extract the provided `project1.tar.gz` into `projects`. After extracting the files, launch the Docker container using the following (hopefully, now fimilar) command in the same directory as your `docker-compose.yml` file to launch the webserver (**For older Docker versions** please add the hyphen and type `docker-compose up` instead):

```
docker compose up
```

Alternatively the following if you want to run the docker container in a non-desktop mode:

```
docker compose run --rm --entrypoint bash ompl
```

If you launched the VNC GUI, start a terminal inside the docker container. In the Docker terminal, use `cd` to change directories to the `project1` directory under `/home/ompl/projects` and type `make` to build. If you cannot see the `project1` directory in the container, verify that you've placed the extracted files in the correct directory on the host machine.

Inspect the source code of the programs and the program output in the order below. In this exercise, we want you to get a feel for how problems are constructed using OMPL as well as how difficult certain problems are. Descriptions of the various demo programs can also be found at http://ompl.kavrakilab.org/group__demos.html. To run one of these compiled programs, open a terminal in the `project1` folder, and type `./NameOfProgram`.

First, you should run the following programs for geometric rigid body planning.

1. `RigidBodyPlanning`: Planning for a free-flying rigid body in 3D with no obstacles.
2. `SE2RigidBodyPlanning`: Planning for a free-flying rigid body in 2D. The robot and obstacles are defined by triangle meshes. The environment is the same as `2D/Maze_planar.cfg` in OMPL.app.
3. `SE3RigidBodyPlanning`: Planning for a free-flying rigid body in 3D. The robot and obstacles are defined by triangle meshes. The environment is the same as `3D/cubicles.cfg` in OMPL.app.

Inspect `Demos/RigidBodyPlanning.cpp` to see how to setup and solve a simple motion planning problem.

Next, you should run the following programs with the specified arguments to plan for a car-like system with increasing levels of model fidelity:

1. `GeometricCarPlanning`: Planning for a car-like system in 2D using Reeds-Shepp curves (these will be discussed later in the class) with no obstacles. Run this program with `./GeometricCarPlanning --easyplan` for the empty environment.
2. `RigidBodyPlanningWithControls`: Planning for a car-like system using first-order controls (steering velocity and forward velocity) with no obstacles.
3. `DynamicCarPlanning`: Planning for a car-like system using second-order controls (steering acceleration and forward acceleration) with no obstacles.

2. Running the OMPL.app GUI

Open the OMPL.app GUI program, `ompl_app`. In this exercise, we want you to play around with a variety of environments and planners to get a feel for how each performs.

You will be testing the following planners in a variety of environments:

1. PRM: A roadmap-based planner. The roadmap is built over the entire space simultaneously based on random sampling.
2. RRT: A tree-based planner. The tree is grown from the start configuration based on random sampling.
3. RRT-Connect: A bidirectional tree-based planner. This is a variant of RRT that grows a tree from both the start and goal, and attempts to connect the two trees.
4. KPIECE: A tree-based planner. The tree is grown from the start, but is guided based on what parts of the space have already been explored.

Each of these planners will be discussed further in the class. You are free to try out any of the planners available in `ompl_app`. Note that a few planners are *asymptotically optimal* planners—these typically end with a "star", e.g., RRT*. These planners will take all available time to optimize the path, and will not return immediate even if a path is found. These kinds of planners will be discussed further on in the class.

Load the following configuration files (files with extension `.cfg`) into OMPL.app. The configuration files can be found in `project1/Resources` or `/usr/local/share/ompl/resources`. You can load problem configurations by using the menu: `File/Open Problem Configuration`.

1. `2D/BugTrap_planar.cfg`: A simple planar "bug-trap" environment.
2. `2D/UniqueSolutionMaze.cfg`: A planar maze-like environment with only one valid solution to the end.
3. `3D/Abstract.cfg`: A 3D rigid body must fly through the environment.
4. `3D/Twistycool.cfg`: A "narrow passage" in which a twisted part must navigate.
5. `3D/Home.cfg`: A "piano movers" problem, where a table must be moved in a cluttered apartment.

You should also visualize the planner graph (the search structure built while planning) by changing the field in the "Show:" drop-down at the bottom of the application. If you want to play around more with various motion planning problems, please try out any of the problem configurations available in the resources folder (`/usr/local/share/ompl/resources`), or to try and set up your own!

Documentation for using the GUI can be found at http://ompl.kavrakilab.org/gui.html. Note that you must load a problem configuration before using the GUI. You can also use the webapp version of the OMPL.app GUI program, available at http://omplapp.kavrakilab.org/.

3. Benchmarking Motion Planners

Benchmarking sampling-based planners is critical to understanding how they perform against any metric. Sampling-based planners are by nature random, and thus can have a wide range of performance purely based on luck of the draw. In this exercise, you will understand how to benchmark OMPL planners and understand their output.

Run the provided benchmarking program `./Benchmarking` with each of the following options:

1. `./Benchmarking 0`: Benchmarks the described planners 50 times each in the "home" environment.
2. `./Benchmarking 1`: Benchmarks the described planners 50 times each in the "twistycool" environment.
3. `./Benchmarking 2`: Benchmarks the described planners 50 times each in the "abstract" environment.

These may take a while to run. Please plan accordingly. After completing the benchmark, a set of `*.log` files will be created in the directory. These contain the output of the benchmarking runs.

We will be using Planner Arena (http://plannerarena.org/), a website for interactive visualization of benchmarking data gathered from OMPL benchmarking. First, run the script "`ompl_benchmark_statistics.py *.log`" in order to generate a database of planning results, `benchmark.db`. On Planner Arena, switch to the

"Change Database" tab, and upload your generated database. You should now be visualizing the results of your benchmarking.

Play around with the visualization, and view the various attributes of the planners. In particular, you will have generated results for the planners you previously used:

1. PRM: With the default settings.
2. RRT: With a range of 5, 25, and 50.
3. RRT-Connect: With a range of 5, 25, and 50.
4. KPIECE: With the default settings.

Before you can make conclusions from your benchmarking data, you must verify that the data you have is *meaningful*. If a planner fails to solve a problem, values in the benchmarking results can range from informative to meaningless, as it is the product of an incomplete run of a planner. To verify that your benchmarking results have solved the problem, you should inspect the following two properties:

1. First, you should observe the "solved" attribute. A "1" corresponds to successfully solving the planning problem. A "0" corresponds to failing to solve the problem. Your planners should have mostly (more than 90%) successes.
2. Next, you should observe the "approximate solution" attribute. A "1" corresponds to only approximately solving the problem, but not successfully reaching the goal. A "0" corresponds to exactly solving the problem, reaching the goal. Your planners should mostly (more than 90%) have exact solutions, indicating they all solved the problem.

If your benchmarking results contain mostly failures, you will need to increase the allowed planning time of the benchmark by editing the `Demos/Benchmarking.cpp` file, and rerunning the benchmark.

Finally, observe how each of these planners performs on the following metrics:

1. "time": how much time was spent solving the problem?
2. "graph states": how many states (roughly how much memory) are in the planner's search graph?
3. "solution length": how long is the solution path?

Deliverables

This project must be completed individually. Submissions are **due Friday September 1st at 5:00pm** via Canvas. Submit a report, no more than 3 pages in PDF format, that addresses the following:

1. **(5 points)** Which method(s) did you use to install OMPL and OMPL.app? How difficult was the setup process?
2. **(10 points)** Describe the perceived difficulty of each of the six demo programs you ran in Exercise 1. Please clearly state on what metric are you evaluating difficulty? According to your metric, which demos seemed easier and/or harder to solve than others?
3. **(15 points)** Describe the perceived difficulty of the specified problem configurations you ran in Exercise 2. Did any configurations seem easier or harder to solve than others? On what metrics?
4. **(15 points)** Out of the specified planners you ran in Exercise 2, did any seem to perform better than the others? In what problem configurations did the planner perform better? On what metric are you evaluating planner performance? Elaborate.
5. **(10 points)** You might have noticed that in OMPL.app each planner has a set of parameters. For PRM, there is "max nearest neighbors." For RRT, RRT-Connect, and KPIECE, there is "range" and "goal bias." Try varying the values of these parameters:
   (a) "max nearest neighbors": from 8 to 100
   (b) "range": from 5 to 50. Note that if 0, OMPL attempts to guess what a "good" value for the range should be.
   (c) "goal bias": from 0.01 to 0.95

   Did you encounter any cases where changing a planner's parameters improved performance on a particular configuration in exercise 2? Explain.
6. **(40 points)** Observe the benchmarking output for the three problems from Exercise 3 on Planner Arena. Which planners perform best in which environments? Explain why, and by what metrics. Attach plots to your report as figures that support your claims. Claims made without supporting evidence will be penalized.
7. **(5 points)** Rate the difficulty of each exercise on a scale of 1–10 (1 being trivial, 10 being impossible). Give an estimate of how many hours you spent on each exercise, and detail what was the hardest part of the assignment.