Institute for Aerospace Studies
UNIVERSITY OF TORONTO

AER1516: Robot Motion Planning

Assignment #2: Planning in Practice

Winter 2022

Assignment Overview

This assignment is designed to provide some experience with sampling-based motion planning. The assignment is due on **Monday, March 28, 2022** by **11:59 p.m.**. The number of points available are shown next to each portion of the assignment description below; there are **50 points** in total. You will submit Python code and a PDF file (in a `.tar` archive).

Motion Planning for a Dubins Vehicle

The rapidly-exploring random trees (RRT) algorithms is a widely-used algorithm for sample-based robot path planning, in part because it is able to take into account *kinodynamic constraints*. These are constraints on velocity and acceleration, for example. RRTs are also able to handle *nonholonomic constrains*, that is, constraints that involve the possible paths taken (parallel parking is such an example: a car cannot move sideways instantaneously). RRT* is a variant of RRT that produces paths that are provably asymptotically optimal, i.e., paths that converge almost surely to the optimum as the number of samples increases.

A popular class of kinodynamic paths is known as Dubins paths. A Dubins path is the shortest curve that connects two points in the two-dimensional Euclidean plane with a constraint on the curvature of the path, and with prescribed initial and final tangent vectors to the path. An assumption is made that the vehicle can only travel forward. This is a very useful model for many robotics applications. You can read more about Dubins paths here. An example of such a path is shown in Figure 1.

Your Assignment Tasks

As part of the assignment, your tasks are to:

1. Implement an RRT-based planner for a Dubins-type vehicle in a 2D planar world with circular obstacles. The RRT planning function should accept an RRT problem setup object that contains: a map (2D size of the world), a list of circular obstacles, a starting pose, and a goal pose (as well as plenty of helper functions for dubins path generation). (**20 points**)

   The function should produce a list of nodes along a valid path starting from the initial state and reaching the goal state. Use the function template in the file called `rrt_planner.py`. Instructions have been provided in this file. The above-mentioned problem setup object as well as a simple test have been implemented in the helper file called `dubins_path_problem.py`, while other helpful Dubins-type path generation functions have been provided in `dubins_path_planning.py`.

   Hint: A purely random exploration strategy is not fast enough when finding the goal. If you already know the goal pose, a "smarter" random exploration strategy can be employed to find a faster solution. Consider what you now know about metrics as well.

2. Implement (as a different function) an RRT*-based planner for the Dubins-type vehicle. The RRT* planning function should accept the same problem setup as the function above, and produce a list of nodes along a valid paths starting from the initial state and reaching the goal state. You may employ the same set of helper functions (and tests). (**25 points**)

   Use the function template in file called `rrt_star_planner.py`. Instructions have been provided in this file. The RRT*-based planner should produce shorter paths than the RRT-based planner.

3. Generate a bar chart plot that compares the performance of RRT against RRT* for 1,000 random instances of a reasonably difficult Dubins planning problem. Your comparison metric (bar height) should be the average path length of the RRT solution versus the average path length of the RRT* solution. (**5 points**)

   Your goal is to *highlight* the gains that can be achieved by using RRT*. As such, you are free to design an obstacle-strewn environment that favours RRT* (as you see fit). For this portion of the assignment, you should submit a PDF document (called `comparison.pdf`) that contains a plot of the environment you have built and the bar chart. **Please also include** a note that indicates which seed value you used for the random number generator in Python. You should keep the `max_iter` parameter set to its default value of 10,000.

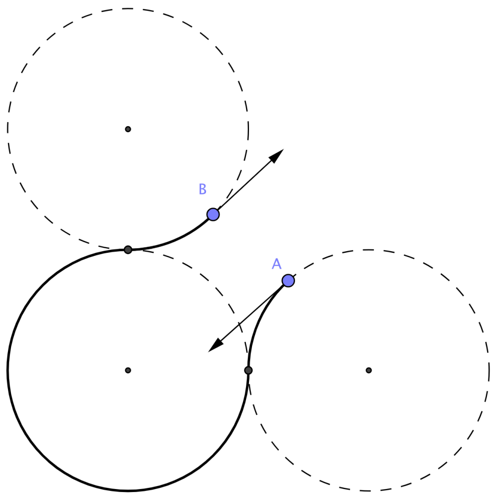

Figure 1: Example Dubins LRL path.

Note that major chunks of code have already been provided—for example `rrt_dubins_problem.py` already contains a propagation function and a collision checking function. You are encouraged to review this code to understand the 'big picture' of how everything works. The code is well documented for your understanding.

Submission and Grading

You *must* write your own code using Python 3.7 and NumPy 1.17.0 or later. If your code does not run, we will be unable to test it. To submit your code, put it in a `.tar` file with this command (from a directory containing the two requisite `.py` files and the PDF file):

```
tar cvf handin.tar rrt_planner.py rrt_star_planner.py comparison.pdf
```

You may then upload your `.tar` file through Quercus. Note that **only the above three files should be included in the archive**. These are the only three files we will examine for grading.

We will run a variety of automated (and manual) tests to check the operation of your code, and it must function properly for you to receive full marks. Each test will provide an obstacle map, a start pose, and a goal pose. Your functions must return a kinematically feasible set of nodes from the start pose to the final goal pose, with no collisions. The tests will check whether nodes on your paths are correct (we will fix the seed of the pseudo-random number generator to ensure that all results will be the same between runs). Keep in mind that there are (reasonable) time limits on the tests (particularly for RRT), which implies it is essential to reach the goal pose fairly fast.

Grading criteria also include: correctness and succinctness of the implementation of your functions, proper overall program operation, and code commenting. Code that is not properly commented or that looks like 'spaghetti' may result in an overall deduction of up to 10%.