Motion Planning RBE 550

Project 3: Boxing with a Chain

DUE: Monday October 7 at 5:00 pm.

All written components should be typeset using LaTeX. A template is provided on Overleaf. However, you are free to use your own format as long as it is in LaTeX.

Submit two files **a)** your code as a zip file **b)** the written report as a pdf file. If you work in pairs, only **one** of you should submit the assignment, and write as a comment the name of your partner in Canvas. Please follow this formatting structure and naming:

```
project_YOUR_NAME.pdf
project_YOUR_NAME.zip
    code
        CMakeLists.txt
        plan.cpp
        visualise.py
        clear_env.txt
        clear.gif
        clear.png
        narrow_env.txt
        narrow.gif
        narrow.png
        build
            clear_path.txt
            narrow_path.txt
            clear_benchmark.db
            narrow_benchmark.db
```

Present your work and your work only. You must *explain* all of your answers. Answers without explanation will be given no credit.

Theoretical Questions (40 points)

1. **(10 points)** Recall the visibility graph method. Similar to the PRM, the visibility graph also captures the continuous space using a graph structure. Compare these two methods. For each method, provide at least one scenario in which it would work well while the other would not. Justify your answer.

2. **(10 points)** For each of the three manipulators shown in Figure 1, determine the topology and dimension of the manipulator's configuration space.

3. **(10 points)** Answer the following questions about asymptotically optimal planners:

    (a) **(5 points)** What is the core idea behind RRT*? That is, explain what modifications are done to

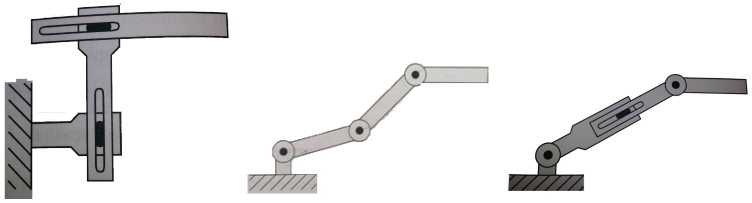

Figure 1: From left to right: a manipulator with two prismatic joints, a manipulator with three revolute joints, and a manipulator with two revolute joints and a prismatic joint.

RRT in order to make it asymptotically optimal. What methods are changed, and what changes are they?

(b) **(5 points)** How does Informed RRT* improve upon RRT*. Is Informed RRT* strictly better than RRT* ? e.g., are there any cases where RRT* can find a better solution given the same sequence of samples?

4. **(10 points)** Suppose five polyhedral bodies float freely in a 3D world. They are each capable of rotating and translating. If these are treated as "one" composite robot, what is the topology of the resulting configuration space (assume that the bodies are not attached to each other)? What is the dimension of the composite configuration space?

Programming Component (60 points+20 bonus points)

Now that you are familiar with OMPL, your task is to implement several advanced features such as creating custom configuration spaces (OMPL StateSpaces), benchmarking asymptotically optimal planners, and exploring different narrow passage sampling distributions. Additionally, you will implement a custom optimization objective. Although these might seem like a lot we provide you with lot of guidance and demos that have already implemented similar functionalities, so make sure you review all the provided information carefully.

Project Exercises

1. **(20 points)** Complete the following functions in `plan.cpp`:

   (a) **(10 points)** `createChainBoxSpace`: Implements the state space for the *chainbox*. Take the following into account:

   - The *chainbox* robot has a square base where each side is 1 and a 4 link chain each link size 1, with the first joint at the center of the box. See included gifs for a visualization.
   - The box center of the robot must remain within a [5,-5], boundary at all times.
   - For guidance on custom state spaces, refer to this tutorial.
   - In your report provide the topological space of this robot.

   (b) **(10 points)** `setupCollisionChecker`: Implements the collision checking for the *chainbox* robot. Take the following into account:

   - The robot base center must remain within [5,-5] at all times. The example gifs violate this, on purpose to ensure you provide your own.
   - The chain should not self-intersect with itself or the box (except for the first link).

- You can heavily leverage and modify `KinematicChain.h` to achieve this collision checking. One approach is to implement everything by modifying `KinematicChain.h`. Then Stablover overthe `setupCollisionChecker` will be only 1 line.

2. **(20 points)** Complete the following functions in `plan.cpp`:

   (a) **(10 points)** `planScenario1`: Solve the narrow passage problem in Scenario 1.
   - The environment obstacles, start, and goal are already provided to you in makeScenario1.
   - Choose the most efficient planner from `rrt`, `prm`, or `rrtconnect`. to solve this problem. In your report, explain which planner you selected and why you believe it is the fastest for this scenario.
   - Use the visualizer to plot the solution, and submit the generated files as `narrow.txt` and `narrow.gif`.

   (b) **(10 points)** `benchScenario1`: Benchmark different PRM sampling strategies for Scenario1:
   - Benchmark (Uniform, Gaussian, Bridge, and Obstacle)-based sampling for scenario1.
   - The example in this benchmarking demo could help.
   - In your report, explain which sampling strategy performed the best and why. Include supporting figures from PlannerArena.

3. **(20 points)** Complete the following functions in `plan.cpp`:

   (a) **(10 points)** `planScenario1`: Solve Scenario2 by finding a path with maximum workspace clearance.
   - The environment obstacles, start, and goal are already provided to you in makeScenario1.
   - Implement a clearance optimization function. This tutorial could be helpful optimal planning tutorial.
   - Calculating the true c-space clearance is impractical in most situations. Simply approximating a workspace clearance by calculating the distance from the box center to the obstacle corners. will suffice for this homework.
   - Choose an asymptotically optimal planner to solve the problem and sumbit your solutions as `clear.txt` and `clear.gif` from the visualizer.

   (b) `benchScenario2`: Benchmark different AO planners

      i. Benchmark `rrt*`, `prm*`, and `rrt#` using your the custom clearance objective. In your report, identify the most effective planner and explain the results with figures from PlannerArena. Include the benchmarking figures from plannerarena in your submission.

4. **Bonus (10 points)**: Implement the true workspace clearance function, which calculates the minimum distance between the entire `chainbox` robot and the obstacles. Explain how you implemented this in your report. Submit the solution as `optimal_clear.gif` and `optimal_clear_path.txt`

5. **(Competition points)**: **Clearance Challenge!** The top three `optimal_clear_path.txt` submissions with the highest total maximum clearance will receive 10, 7.5, and 5 points, respectively.

   - You can use the provided `clearance` executable inside the build folder to help you check what the clearance cost of your path is. This executable assumes an optimal_clearance_path.txt file exists in the same location. The provided path should provide a clearance of inf.
   - If you want to replicate the same cost make sure you use weights of 1 when you define your subspaces.

Your code must compile, run, and solve the problem correctly. You are allowed to add new functions but you are not allowed to use external libraries, or modify the names of the provided functions. Correctness of the implementation is paramount, but succinct, well-organized, well-written, and well-documented code is also taken into consideration.

Protips

- You can use the ompl_benchmark_statistics.py script to concatenate multiple log files into one db file.