Motion Planning RBE 550

Project 1: Throwing Points on a Disk

DUE: Thursday September 5 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
        DiskSampler.h
        DiskSampler.cpp
        DiskPlanning.cpp
        visualize.py
        build
            naive_samples.graphml
            correct_samples.graphml
```

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

Goal of This Project

This project aims to assess your understanding of fundamental concepts required for the course. It will also introduce you to key operations in OMPL, specifically sampling and collision checking for a simple point robot.

Theoretical Questions (40 points)

1. **(15 points)** Consider $\mathcal{A}$, a unit disc centered at the origin in the workspace $\mathcal{W} = \mathbb{R}^2$. Suppose $\mathcal{A}$ is described by the algebraic primitive $H = (x,y) \mid x^2 + y^2 \leq 1$. Demonstrate that rotating this primitive about the origin does not alter its representation. To prove this, show that any point within the rotated primitive H' is also within H, and vice versa.

2. **(25 points)** Let S be a square object parameterized by (x, y, θ, s), where (x, y) denotes its position in the X-Y plane, θ represents its rotation relative to its center, and s is the length of its sides. Given a random point $p_r = (x_r, y_r)$, provide a pseudocode algorithm to determine if p_r is inside the square. The function should return `False` if the point is inside or on the edge of the square, and `True` otherwise.

Ensure to account for all edge cases in your solution.

Programming Component (60 points)

In this component, you will:

1. Implement a sampler that generates uniform random samples on a disk with a radius of 10.
2. Develop a simple collision checker, similar to the pseudocode from the previous exercise.

You will be provided with the following files:

1. `CMakeLists.txt` – This file helps create a Makefile for easier building. Use it as you did in the previous project. You do not need to modify this file, but you are welcome to review and understand it.
2. `DiskSampler.h` – Header file for DiskSampler. This file includes the sampling and collision checking functions. You should not need to modify this file, but you may review it to understand its contents.
3. `DiskSampler.cpp` – Contains the implementation of sampling and collision checking functions. You will need to modify this file.
4. `DiskPlanning.cpp` – Sets up some planning routines and performs sampling and collision checking. You do not need to modify this file, but you are encouraged to review it to understand its function.
5. `visualize.py` – Helps visualize the results of your implementation. You do not need to modify this file unless you are not using Docker and the generated files are in a different location.

First, update your environment. If you are using docker:

```
# pull again the updated image
docker pull cchamzas/rbe550-ompl:initial-commit
```

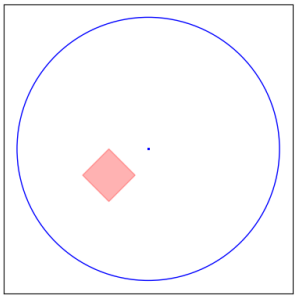
If you have your local setup

```
#Install networkx, and matplotlib
pip install networkx matplotlib
```

Now to build the project

```
# Build the project.
cd code
# Build the project.
mkdir -p build && cd build
cmake ../ && make
# Run the executable.
./diskPlanning
#Run the visualize script
cd ../
python3 visualize.py
```

If everything is set up correctly, you should see two images named `naive_samples.png` and `correct_samples.png` generated, as shown below: You are now ready to begin the coding part of the assignment



1. **(40 points)** Implement a sampler that generates uniformly distributed points on a disk in $\mathbb{R}^2$ space using OMPL. Complete the following tasks:

   - Implement the `sampleNaive(ob::State *state)` function in `DiskSampler.cpp`. Use the following process to perform naive sampling on a disk with a radius of 10:
     (a) Sample random polar coordinates $r \sim [0, 10]$ and $\theta \sim [0, 2\pi)$.
     (b) Convert the polar coordinates to Cartesian coordinates (x, y) in $\mathbb{R}^2$.

     Evaluate whether these points are uniformly distributed on the disk. Explain why they may not be uniformly distributed and include the `naive_samples.png` figure in your report.
   - Implement the `sampleCorrect(ob::State *state)` function in `DiskSampler.cpp`. Use a correct sampling process to generate uniformly random points on the disk. (Uniform here means samples drawn from a uniform distribution over the area of the disk, not a grid pattern). Many methods can achieve this, but the most elegant solution involves a single code change. Describe what you changed and why, and include the `correct_samples.png` figure in your report.

2. **(20 points)** Implement a collision checker for a point and a translated and rotated square using the algorithm proposed in Theoretical Question 2. Complete the following task:

- Implement the `isStateValid(ob::State *state)` function in `DiskSampler.cpp`. This function should check for collisions with a square obstacle of edge size $2 * \sqrt{2}$, located at (-3, -2) and rotated by $\pi/4$. Use the `visualize.py` script to verify the correctness of your implementation. Include the final figure produced by the visualization script in your report.

Pro Tips

1. You can use `M_PI` as the value of π in C++.
2. For sampling random values, use the `rng_` object, which is available in the `DiskSampling.h` file.
3. Refer to the OMPL documentation available here to understand the basics of OMPL. There is a search bar in the top right corner.