Semester 2 2017
COMP3702/7702 ARTIFICIAL INTELLIGENCE
ASSIGNMENT 2: Coordinating Robots to Clean the Walls of New Buildings

Note:

- This assignment consists of two parts: Programming and report.
- You can do this assignment in a group of at most 3 students. This means you can also do the assignment individually.
- For those who choose to work in a group:
  - All students in the group must be enrolled in the same course code, i.e., all COMP3702 students or all COMP7702 students.
  - Please register your group name in https://goo.gl/d21epq before **11.59pm on Tuesday, 8 September 2017**. If you have not registered your group by the said time, you will need to work on the assignment individually.
  - All group members are expected to work in both programming and report. In the report, you are required to write the role of each team member.
- Submission Instruction:
  - Your program should compile using ant or make/cmake from command prompt, and generate an executable named a2-[courseCode]-[ID] that can be run from command prompt as:

    ```
    > a2-[courseCode]-[ID] inputFileName outputFileName
    ```

    Please replace courseCode with either 3702 or 7702, depending on which class you are enrolled in. If you work individually, please replace ID with your student ID. Otherwise, please replace ID with your group name.
  - The report should be in .pdf format and named a2-[courseCode]-[ID].pdf.
  - The report and all the source codes necessary to compile your program should be placed inside a folder named a2-[courseCode]-[ID]. Please submit only source codes (i.e., remove all object files).
  - The folder should be zipped under the same name, i.e., a2-[courseCode]-[ID].zip, and the zip file should be submitted via turnitin before **11.59pm on Friday, 22 September 2017**.

In the recent years, there has been renewed interest and development of wheeled robots that can crawl and clean the wall (see Fig. 1 for examples). Let's call a wall crawling mobile robot, an Autonomous Surface Vehicle (ASV). Note: This is a non-typical use of the term ASV; the "surface" in ASV usually refers to water surface; in our assignment, "surface" in ASV means wall surface.

Fig. 1. Examples of wheeled robots that can crawl walls. Left: WallRover. Right: Harvard's Root robot.

To cut the cost of cleaning the interior of many of its new remarkable buildings, UQ-Facilities would like to use multiple of the aforementioned ASVs to clean the wall. Their idea is to have multiple ASVs connected by push brooms and let the robots move in synchrony, so that the broom will clean the area. They have developed the hardware mechanism for this, and are now struggling to coordinate these vehicles. In this assignment, your task is to help UQ-Facilities by developing a prototype software to coordinate multiple ASVs carrying push brooms.

For this prototype, the problem is simplified as follows:

1. Each ASV is modeled as a point, each broom as a straight line segment, and the environment as a normalized 2D plane (i.e., a 2D Euclidean space of size [0,1]X[0,1]).
2. You only need to be concerned about the positions of the ASVs (i.e., velocity, acceleration, momentum, gravity, tethered cables, etc. are ignored).
3. The environment is fully observable, deterministic, static, and continuous.
4. All obstacles in the environment have rectangular form, and are axis-aligned.
5. The cost of moving from one configuration to another is the sum of the straight-line distance traveled by each ASV.

Your task is to develop a program that calculates a collision free path for the ASVs and brooms to move from a given initial configuration to a given goal configuration, while satisfying the following requirements:

1. The length of each broom is fixed at 0.05 units in length.
2. The system must form a connected chain at all times. A connected chain means each ASV can be connected to at most two brooms and each end of each broom is tied to an ASV.
3. The polygon formed by connecting the two ends of the connected chain with a straight line segment must, at all times, be convex and have an area of at least πr_{min}^2, where $r_{min} = 0.007(n\text{-}1)$ and n is the number of ASVs.
4. The brooms must never intersect with each other.
5. Brooms and ASVs must never intersect with obstacles.
6. Brooms & ASVs cannot move outside the [0,1]X[0,1] workspace.
7. The planned path must be given as a sequence of positions (primitive steps) such that on each step, each individual ASV moves by a distance of at most 0.001 units.
8. Requirements 1-6 must hold at each primitive step. Since the distances are very small (at most 0.001 unit length for each ASV), it is sufficient to test the requirements only at the end of each primitive step.

We have provided a supporting code in Java to help check the above requirements.

Input format

The input of the program is a text file containing information on ASVs and environment. In particular, the file consists of k+4 lines, where k is the number of obstacles. Below is the format of each line:

- The first line is the number of ASVs, denoted as *n*.
- The next two lines are the initial and goal configurations. The initial & goal configurations are written as x-y positions of each ASV. Broom-1 is attached to ASV-1 and ASV-2, ..., broom-(*n*-1) is attached to ASV-(*n*-1) and ASV-*n*.
- The fourth line is the number of obstacles.
- Each line in the last *k* lines represents the vertices of each rectangular obstacle. Each rectangular obstacle is written as a quadruple of the x-y coordinates of its vertices in counter-clockwise order, starting with the lower-left vertex.

Example of an input file:

```
2
0.3 0.5 0.35 0.5
0.6 0.5 0.65 0.5
2
0 0.2 0.2 0.2 0.2 0.4 0 0.4
0.5 0.6 0.7 0.6 0.7 0.9 05 0.9
```

The input file means:

- The system consists of 2 ASVs (and 1 broom).
- The initial positions for ASV-1 and ASV-2 are (0.3, 0.5) and (0.35, 0.5).
- The goal positions for ASV-1 and ASV-2 are (0.6, 0.5) and (0.65, 0.5).
- There are two rectangular obstacles. The vertices of the first obstacle are (0, 0.2), (0.2, 0.2), (0.2, 0.4), (0, 0.4). The vertices of the second obstacle are (0.5, 0.6), (0.7, 0.6), (0.7, 0.9), (0.5, 0.9).

Output format

The output of the program is a text file containing the path. In particular, the file consists of *m*+2 lines. The first line is the number of primitive steps, denoted as *m*, and the total length of the path, separated by a single white space. The next line is the initial configuration, and then each line in the next *m* lines contains the position of each ASV at the end of every primitive step.

Example of an output file:

```
300 0.6
0.3 0.5 0.35 0.5
0.301 0.5 0.351 0.5
:
0.4 0.5 0.45 0.5
0.401 0.5 0.451 0.5
:
0.5 0.5 0.55 0.5
0.501 0.5 0.551 0.5
:
0.6 0.5 0.65 0.5
```

The above output means ASV-1 moves from (0.3, 0.5) to (0.301, 0.5) to … to (0.6, 0.5), ASV-2 moves from (0.35, 0.5) to (0.351, 0.5) to … to (0.65, 0.5).

We have provided a Java program to visualize the input and output files.

Grading for the Programming Part (total points: 60/100)

If you use sampling-based method, we will run your program 10X for each query. Your program is deemed as solving the query if it solves at least 5 out of 10 runs within the given time limit.

The time limit is 1min CPU time for open environment and 5min CPU time for cluttered environment. The CPU time will be taken on a PC with the same specification as the ones in prac & tutorial lab.

The details of the grading scheme is as follows. If your situation satisfies more than one marking band, we will use the higher band.

COMP3702:

- >= 1 & < 10: The program does not compile nor run (staff discretion).
- >= 10 & < 20: The program runs but fails to solve any query within the given time limit (staff discretion).
- >= 20 & < 30: The program solves at least 1 of 4 queries with > 1 and <= 3 ASVs in open environments. Each query is worth 2.5 points.
- >=30 & < 40: The program solves at least 1 of 4 queries with > 3 and <= 10 ASVs in open environment. Each query is worth 2.5 points.
- >=40 & < 50: The program solves at least 1 of 4 queries with > 3 and <= 10 ASVs in cluttered environment.. Each query is worth 2.5 points.
- >=50 & <= 60: The program solves at least 1 of 5 queries with > 10 ASVs operating in cluttered environment. Each query is worth 2.5 points.

COMP7702:

- >= 1 & < 5: The program does not compile nor run (staff discretion).
- >= 5 & < 10: The program runs but fails to solve any query within the given time limit (staff discretion).
- >= 10 & < 20: The program solves at least 1 of 4 queries with > 1 and <= 3 ASVs in open environments. Each query is worth 2.5 points.
- >=20 & < 30: The program solves at least 1 of 4 queries with > 3 and <= 10 ASVs in open environment. Each query is worth 2.5 points.
- >=30 & < 40: The program solves at least 1 of 4 queries with > 3 and <= 10 ASVs in cluttered environment.. Each query is worth 2.5 points.
- >=40 & < 50: The program solves at least 1 of 4 queries with > 10 and <= 15 ASVs operating in cluttered environment. Each query is worth 2.5 points.
- >=50 & <= 60: The program solves at least 1 of 5 queries with > 15 ASVs operating in cluttered environment. Each query is worth 2.5 points.

Report (total points: 40/100)

Your report must contain answers to the following questions:

1. [5 points] Please define the Configuration Space of the problem and describe which method for searching in continuous space do you use.
2. [10 points] If you use sampling-based method, please describe the strategy you apply or develop for each of its four components. Otherwise, please describe the details of your discretization method.
3. [12.5 points] Which class of scenarios do you think your program will be able to solve? Please explain your answer.
4. [12.5 points] Under what situation do you think your program will fail? Please explain your answer.

Please note that in each of the above question, when explanation is requested, the explanation part is 80% of the total points. For the explanation part, you will get higher mark for stronger argument. A strong argument should at least include a logical explanation of why and include experimental results to back your explanation. Please also note that good explanation is NOT equal to long explanation!!!

oOo That's All, Folks oOo