Semester 2, 2019
COMP3702/7702 ARTIFICIAL INTELLIGENCE
ASSIGNMENT 2: Canadarm – Continuous Motion Planning
Due: 5pm, Friday 27 September

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:
  - Please register your group name at the following link before **5pm on Monday, 9 September 2019:**
https://docs.google.com/spreadsheets/d/1X3ak5KwEWVdgCwgsFypBHNQr-pq7jiI6ZJ_3m4KBc_4/edit#gid=0
  - All students in the group must be enrolled in the same course code, i.e., all COMP3702 students or all COMP7702 students.
  - All group members are expected to work in both programming and report. The demo will involve Q&A to each group member, individually.
- **Submission Instructions:**
  - Your program should be called "solver" and run directly from the command prompt, e.g.:

    ```
    > python solver.py inputFileName outputFileName
    ```

  - If you are not programming in python, you must provide a makefile for compiling your code with Ant/Cmake/Javac. You must also provide a Usage document for compiling your code. Your code should be able to compile on the lab computers without any need to open an IDE.
  - You should submit **only the source code** required to compile and/or run the program, i.e., remove all object files and executables before submission.
  - The report should be in .pdf format and named a1-[courseCode]-[ID].pdf.
    If you work individually, ID is your student number.
    If you work in a group, ID is the student number of all group members separated by a dash. For instance, if you work in a group of two, and the student number is 12345 and 45678, then
    [ID] should be replaced with 12345-45678
  - The report and all the source code necessary to compile your program should be placed inside a folder named a1-[courseCode]-[ID].
  - The folder should be zipped under the same name, i.e., a1-[courseCode]-[ID].zip, and the zip file should be submitted via Turnitin before **5pm on Friday, 27 September 2019**.
  - If you work in a group, only 1 group member should make the submission.
- **Demo Instructions:**
  - Demos will be scheduled the week following the submission deadline (i.e. Wk 11 of semester).
  - Each group must demo to receive the programming marks, and all group members must be present for the demo.
  - You must use the lab PC for the demo, and therefore you must make sure that your code compiles (if in Java or C/C++) and runs correctly on the lab PCs.

Background

Canadarm2 (e.g., Fig. 1) is a remote-controlled mechanical arm aboard the International Space Station (ISS). The robotic arm is used to deploy, capture and repair satellites, position astronauts, maintain equipment, and move cargo. You've been hired to develop Motion Planning software for a simplified version of the robotic arm. The rest of this specification document details the system and task.

Fig. 1. Canadarm2 with astronaut Stephen K. Robinson. Source: NASA

The system

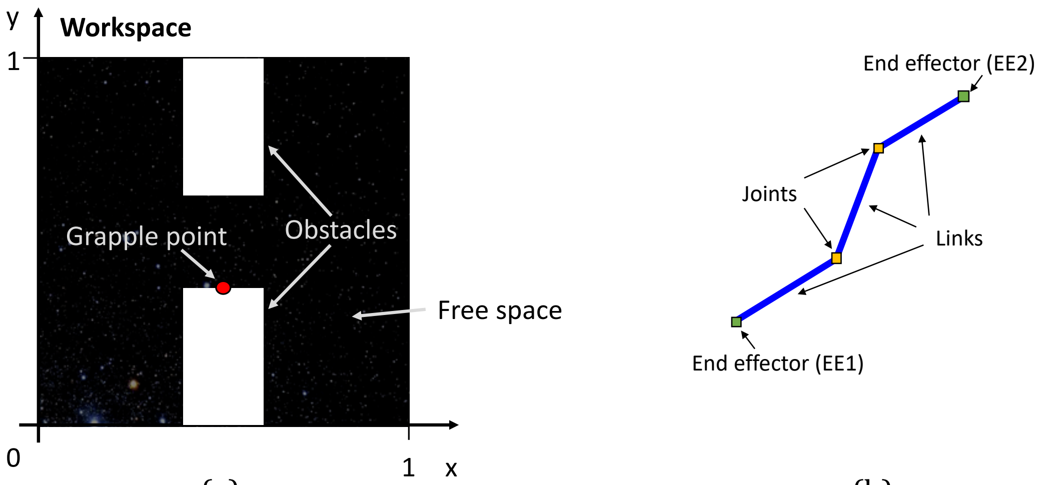

Fig. 2 (a) An example of the workspace. (b) The robot arm and its components

The simplified Canadarm operates in a 2D workspace (rather than 3D). In particular, the 2D workspace is a plane, represented as $[0, 1] \times [0, 1] \subset R^2$, and is populated by rectangular obstacles. In addition, there are grapple point(s) which the end effectors of the robotic arm can attach to. One of the end effectors must be grappled to a grapple point at any time. The exact dimensions and positions of each obstacle and the number and position of the grapple points in the environment are known prior to execution. Fig. 2(a) illustrates this environment.

The robotic arm is composed of *x* links and *x* joints, where *x* is a non-negative integer, with two end effectors EE1 and EE2, which can attach onto grapple points. An example robotic arm with 3 links is shown in Fig. 2(b). Each link of the robot is a line segment attached to a joint. The link connected to the grappled end effector acts as a joint. Each joint is a rotational joint which means it can only rotate. A local coordinate system is attached to each joint. The co-ordinate system of the joint at the location of the grappled end effector, coincides with the coordinate system of the end effector. For the remaining joints, the x-axis is the line that coincides with the previous link. We define the angle of segment-i as the angle between link-i and the X axis of the coordinate system attached to joint-i. The joints are numbered relative to the grappled end-effector. Fig. 3 illustrates the rotational joints. In some tasks, the links are telescopic and can change length (i.e. when the specified min &

max segment lengths differ). This allows the robotic arm to more easily reach the grapple points.

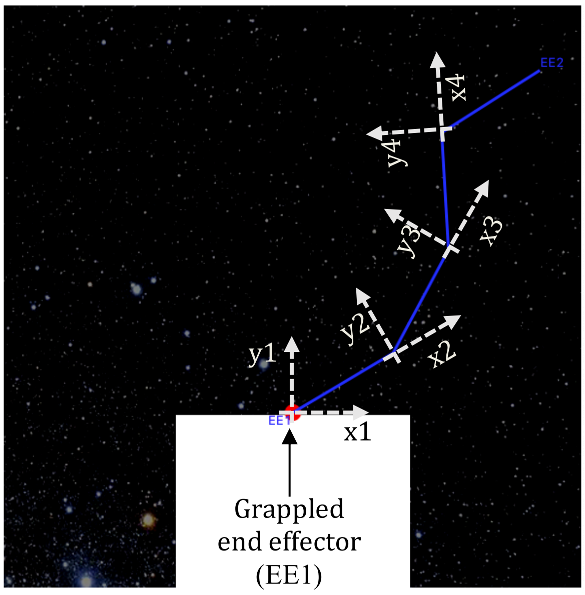

Fig. 3 (a) An illustration of the co-ordinate scheme of each joint on a 4-segment robotic arm.

What your program should do

Given the initial and goal configurations of the Canadarm robotic arm, as well as a map of the environment, your program must find a valid path from the initial to the goal configurations. A valid path means that when the Canadarm executes the path, it will satisfy the following requirements:

1. The path consists of primitive steps. In each primitive step, each joint of the robot arm cannot move more than 0.001 units (i.e. radians or arm length).
2. It will not collide with any of the obstacles
3. It will not collide with itself
4. The entire Canadarm robotic arm must lie inside the workspace
5. The angle between adjacent arm segments cannot be tighter than 15 degrees (i.e. angles 2, 3, 4... must be between -165 degrees and +165 degrees).
6. The segment lengths must be within the bounds specified in the input file (i.e. within min and max lengths)
7. Since a primitive step is very small, it is sufficient to satisfy requirements 2-4 at the beginning and end of each primitive step.

Input and Output format

The input is a .txt file. To describe the format of this file, we first need to describe the format of a configuration.

Format of a configuration: A configuration is represented by *n* real numbers, where *n* is the dimension of the C-space. See "example_output.txt" in the support code for an example of how this is formatted. Each number is separated by a space, and each group of numbers is separated by a semicolon. The first two numbers are the position of the origin of the grappled end effector in the workspace. The next *x* numbers are the joint angles (ordered relative to the grappled end effector), with each joint angle defined in degrees. The last *x* numbers are the lengths of each link (line segment) of the robot ordered relative to end effector 1.

Input format. The program you develop should accept an input file. The file contains information on the set-up of the robotic arm, the initial configuration, the goal configuration, and the environment. This input file uses comments (marked with '#') to label which values correspond to which parameters.

The input parameters are as follows:

```
----------
# Robotic arm details
number of segments
min lengths for each segment (order from ee1)
max lengths for each segment (order from ee1)

# Initial Configuration
initial grappled ee (either 1 or 2, representing ee1 or ee2)
initial grappled eex, eey
initial angles from grappled ee +ve axis (degrees)
initial segment lengths from ee1

# Goal Configuration
goal grappled ee (either 1 or 2, representing ee1 or ee2)
goal grappled eex, eey
goal angles from grappled ee +ve axis (degrees)
goal segment lengths from ee1

# Environment
number of grapple points
x, y for each grapple point

number of obstacles
x1, y1, x2, y2 for each obstacle
----------
```

Output format. Your program should output the robot arm's path to a file with the following format.

1. The file consists of m+1 lines, where m is the number of primitive steps in your path.
2. The first line is the initial configuration.
3. The next m lines are the end configuration of each primitive step, where the final line should correspond to the goal configuration.

Examples of the input and output files are in the accompanying supporting software.

What is provided to you:
Support code can be found at:
https://gitlab.com/3702-2019/assignment-2-support-code

The supporting code is provided in Python only and includes:

1. Support code to represent the problem (problem_spec.py, robot_config.py, obstacle.py, angle.py)
2. A visualiser
3. A tester
4. Test cases to test and evaluate your solution

Please ensure that your solution works by running it through the provided tester.

You will be graded on your **programming** (60%) and a **report** (40%).

Grading for the Programming component (total points: 60/100)
For marking, we will use scenarios and queries of varying difficulty levels to evaluate your solution:
Level 1: at most 3 arm segments, all with fixed length, 1 grapple point, wide gaps between obstacles
Level 2: at most 4 arm segments, some with telescoping length, 1 grapple point, narrower gaps between obstacles
Level 3: at most 4 arm segments, all with telescoping length, 2-3 grapple points, narrower gaps between obstacles
Level 4: at most 5 arm segments, all with telescoping length, 3+ grapple points, tight gaps between obstacles

If you use a sampling-based method, we will run your program up to 3X for each query. Your program is deemed as solving the query if it solves at least 1 out of 3 runs within the given time limit. The time limit is 1 minute per query for levels 1&2 and 2 minutes for the higher-level test cases.

The details of the grading scheme are as follows:

For all students:

- The code to be tested will be downloaded from your Turnitin submission.
- No attempt will be made to make any changes to your code during your demo.

COMP3702:

- >= 1 & < 10: The program does not run (staff discretion).
- >= 10 & < 20: The program runs but fails to solve any query. The program fails to solve a query when it does not find a valid path after more than 2X the given time limit.
- >=20 & <30: The program solves at least one of the queries within 1-2X the given time limit.
- >=30 & <40: The program solves at least one Level 2 problem
- >=40 & <50: The program solves at least one Level 3 problem
- >=50 & <=60: The program solves at least one Level 4 problem

COMP7702:

- >= 1 & < 5: The program does not run (staff discretion).
- >= 5 & < 10: The program runs but fails to solve any query. The program fails to solve a query when it does not find a valid path after more than 2X the given time limit.
- >=10 & <20: The program solves at least one of the queries within 1-2X the given time limit.
- >=20 & <30: The program solves at least one Level 2 problems
- >=30 & <45: The program solves at least one Level 3 problems
- >=45 & <=60: The program solves at least one Level 4 problems

Grading for the Report component (total points: 40/100)

Please answer the following questions as part of the report:

1. [5 points] Please define the Configuration Space of the problem and describe which method you use for searching in continuous space.
2. [20 points] If you used a sampling-based method, describe the strategy you applied to develop each of its components (sampling strategy, connection strategy, checking if a configuration is valid or not, checking if a line segment in C-space is valid or not). Otherwise, please describe the equivalent details of your discretization method.
3. [15 points] Which class of scenarios do you think your program will be able to solve and under what situation(s) do you think your program will fail? Please explain your answers.

Please format the report with each question under its own heading. Any explanation should include a logical explanation and include experimental results to back your explanation.