Assignment 3

Total points - 40 + 10 bonus.

Name:
UNCC email:
Total points: 40 (+ 10 bonus)
(Grading Rubrics are below each question)

Submission Format: Submit your solution as **a single .zip file** containing (1) a single PDF file and (2) other files for the implementation portions of the assignment. Your files should run without having to download any third-party package and should be able to reproduce all figures and results reported in your PDF file.
Ps. Please include a readme file containing the instructions to run your program.

List of Collaborators and Acknowledgements: List the names of all people you have collaborated with and for which question(s).

Questions about this assignment should be directed to TA Yash Ahire, yahire@uncc.edu

Contents:

1. **Given data**
2. **Questions**
3. **Explanations**
4. **Deliverables**
5. **Rubrics**
6. **References**

1 . Given Data ==========

Data to be used for both questions.

Bots that can be used for this assignment :-
Turtlebot,
KUKA Youbot,
Clearpath Husky
(You can design your own bot or use any other similar bot)

1 . Map Specifics:-

Size of the map: (0, 0) and (8, 8).

- **Obstacle 1: (0.5, 0.5), (1.5, 0.5), (1.5, 1.5), (0.5, 1.5)**
- **Obstacle 2: (3, 3), (4, 3), (4, 4), (3, 4)**
- **Obstacle 3: (2, 1), (2.5, 0.5), (3, 1), (2.5, 1.5)**

- **Obstacle 4: (1, 3), (1.5, 2.5), (2, 3), (1.5, 3.5)**
- **Start Position: (0.1, 0.1)**
- **Goal Position: (6,6)**

RRT Parameters:-

- **Length of incremental straight-line during tree extension: 0.2**
- **Maximum number of iterations: 5000**
- **Maximum distance from goal to connect to a node: 0.2**
- **Robot radius: 0.25**

2. Questions ==========

Potential Field Method (PFM) Question

1. Create a map showing the workspace in CoppeliaSim with four obstacles.
2. Circumscribed Circles: Calculate the circumscribed circle of each obstacle to fully cover the polygonal obstacle regions specified by the vertices.
3. Potential Field Method: Implement a potential field-based method to plan a path for a mobile robot to navigate through the map and reach the goal position. Note that you will need to construct the C-space of the robot first by accounting for the robot shape.
4. **[BOT - You can use existing Bots mentioned above]** Sensors: Use proximity sensors to detect obstacles in the robot's path and adjust its trajectory accordingly. You can use a ray sensor or a proximity sensor to detect the distance and angle of the nearest obstacle within a certain range and field of view. You can use this information to generate a repulsive force or torque that directs the robot away from the obstacle.

5. **Bonus:** Implement a solution to address local minima that may occur in the potential field-based method. Show a scenario where the PFM will lead to local minima and how your approach addresses this issue (See [3] for details).

(RRT / RRT connect) + Short-cutting / RRT* Question

1. Create a map in CoppeliaSim with four obstacles.
2. Start and goal positions of the robot are given above.
3. Use RRT algorithm to generate a path for the robot to reach the goal configuration while avoiding all obstacles, and report your plot results in the write-up. Note that you will need to compute the C-space of the robot first. There is no need to use circumscribed circles to represent the obstacles here.
    - Parameters for RRT algorithm, such as the length of the incremental straight-line during tree extension at each step are given.

4. Plan a path for the robot using RRT+short-cutting in CoppeliaSim and demonstrate the robot navigating through the map without hitting any obstacles. In the write-up, provide the comparison between paths generated by RRT and RRT with short-cutting.
5. Plan a path for the robot using RRT connect + short-cutting and demonstrate the robot navigating through the map without hitting any obstacles.
6. If considering kinematics constraints for the robot, such as when the robot can not simply move in a straight line fashion, please modify the RRT algorithm to generate a smooth path for the robot to track.
7. Implement the modified RRT program to generate a smooth path and demonstrate the robot navigating through the map smoothly.
8. **Bonus:** Implement RRT* algorithm to find an improved path with kinematics constraints.

3. Explanation ==========

Explanation for problem 1

To design a potential field-based path planning algorithm in CoppeliaSim, we need to follow the given requirements and map specifics.

Map Creation:
Firstly, we need to create a map in CoppeliaSim with at least four static convex and concave obstacles. We can use the coordinates provided to create four obstacles of different shapes and sizes.

Circumscribed Circle:
Next, we need to use the circumscribed circle to cover all of the obstacles in the map. We can use the smallest circle that contains all the obstacle points as the circumscribed circle.
You can use the center and radius of the circle as the equivalent representation of the obstacle. The center can be calculated as the midpoint of any two non-collinear vertices, and the radius can be calculated as half the distance between any two vertices of the polygon.

Start and Goal Positions:
We need to specify the start and goal positions of the robot as (x, y) coordinates. The start position can be set to (0, 0) and the goal position can be set to (6, 6).

Potential Field-Based Method:

Designing a potential field-based method involves generating a **repulsive potential field around each obstacle and an attractive potential field towards the goal position**. The total potential field is the sum of the attractive and repulsive potential fields.
Repulsive Potential Field:
The repulsive potential field around an obstacle is given by:
Ur = k * (1/d - 1/Dr)^2
where k is a constant, d is the distance between the robot and the obstacle, and Dr is the desired safe distance from the obstacle.

Attractive Potential Field:
The attractive potential field towards the goal position is given by:
Ua = k * d
where k is a constant and d is the distance between the robot and the goal position.
Total Potential Field:
The total potential field is given by:
U = Ua + Ur1 + Ur2 + Ur3 + Ur4

where Ur1, Ur2, Ur3, and Ur4 are the repulsive potential fields around obstacles 1, 2, 3, and 4, respectively.

Path Planning Algorithm:
The path planning algorithm involves moving the robot in the direction of the negative gradient of the total potential field until it reaches the goal position. We can use a PID controller to control the motion of the robot towards the goal position.

Bonus: Address Local Minima:
Local minima can be addressed by introducing random perturbations in the robot's motion. We can add a random component to the robot's velocity to generate a more diverse set of trajectories, which may help the robot escape from local minima.

Explanation for problem 2

Map creation :
You will need same maps for both the problems
Create the map with obstacles in CoppeliaSim We need to create the map with four static convex and concave obstacles in CoppeliaSim. We can use the following coordinates for the obstacles:

Start and Goal Positions: : Set the start and goal positions of the robot We need to set the start position of the robot as (0, 0) and the goal position as (6, 6).

Implement the RRT algorithm:

Generate a collision-free path from start to goal We can implement the RRT algorithm to generate a collision-free path from start to goal using the following steps:

1. Initialize the tree with the start position.
2. For each iteration, generate a random point in the space.
3. Find the nearest point in the tree to the random point.
4. Generate a new point by extending the tree from the nearest point towards the random point by given length.
5. If the new point collides with any obstacle, discard it and repeat from step 2.
6. If the new point is within a distance given from the goal, connect it to the goal node.
7. If the maximum number of iterations is reached, exit the loop.
8. Backtrack from the goal node to the start node to generate the path.

Implement the path in CoppeliaSim
Show the robot navigating through the map without hitting any obstacles. We can implement the path generated by the RRT algorithm in CoppeliaSim by connecting the nodes using the **"Line" object.** We can then simulate the robot moving through the path to show that it does not collide with any obstacle.

kinematics constraints

Modify the RRT algorithm to generate a smooth path . To modify the RRT algorithm to generate a smooth path, we can use the RRT* algorithm. RRT* algorithm uses a cost function to bias the tree expansion towards the goal region and generates a smoother path by rewiring the tree.

1. Define a cost function that considers the distance to the goal and the smoothness of the path. The cost function can be used to evaluate the cost of each new node during the tree expansion.

2. Modify the RRT algorithm to use the cost function to bias the tree expansion towards the goal region. During the tree expansion, the RRT* algorithm selects the node with the lowest cost to expand the tree towards.

3. Use the cost function to rewire the tree after generating the tree. Iteratively check if any nodes can be connected to their neighbors with a lower cost path, and update the tree if possible. Repeat this process until no more improvements can be made.
4. Connect the start node to the goal node through the lowest cost path to generate the final path. This path is guaranteed to be smooth and optimal.

5. Implement the modified RRT* algorithm in CoppeliaSim to demonstrate how the robot navigates smoothly through the map. By using this modified algorithm, the robot can generate a smooth and efficient path, making it easier to control and more stable during movement.

More info on BOT

1. Create a new scene in CoppeliaSim and add a car-like robot model to it. You can use the "model browser" to select and add a car-like robot from the available models.
2. Define the kinematic model of the car-like robot. The kinematic model includes the robot's pose and motion constraints. In CoppeliaSim, you can define the robot's pose using a 3D transform, which includes the robot's position and orientation in the world frame. You can define the robot's motion constraints using joint constraints, which limit the robot's motion along its steering and driving axes.
3. Define the control inputs for the robot. The control inputs for a car-like robot typically include a steering angle and a velocity. In CoppeliaSim, you can define the control inputs using a script or a controller. You can write a script to control the robot's motion based on the RRT algorithm's output, or you can use a controller to track a trajectory generated by the RRT algorithm.
4. Configure the RRT algorithm parameters for the robot. The RRT algorithm parameters include the step size, the maximum number of iterations, and the goal bias. You can adjust these parameters based on the robot's kinematic constraints and the desired path.
5. Implement the RRT algorithm for the robot. You can implement the RRT algorithm using a script or a plugin in CoppeliaSim. You can use the RRT algorithm to generate a path from the robot's initial pose to the goal pose while avoiding obstacles. The RRT algorithm will output a sequence of poses that the robot should follow to reach the goal.

In terms of sensors, you would likely need to use some combination of the following sensors:

1. LIDAR sensor: This would allow the robot to detect obstacles in its environment and avoid collisions.
2. IMU sensor: This would allow the robot to measure its orientation and angular velocity, which can be used for control and navigation.
3. Encoder sensor: This would allow the robot to measure the rotation of its wheels and calculate its linear and angular velocity.
4. Camera sensor: This could be used for object recognition or visual odometry, which can improve localization and mapping.
5. GPS sensor: This could be used for global positioning and navigation, although its accuracy can be limited in indoor environments.

4. Deliverables ==========

1 . Scene in .ttt format which should demonstrate both the above problems (Submit a single scene or two different ones)
2. Python/Matlab code files for both the problems.
3. Readme files with instructions to execute the scene/program.

4. Write-up to include the planning results from your algorithms.

5. Rubrics ==========

Scene - 5 points
Documentation - 5 points

Question 1 :
Executable program - 15 points
Local minima (Bonus) - 5 points

Question 2 :
Executable program - 15 points
RRT* (Bonus) - 5 points

6. Additional References ==========

Refer to our ppt slides as the main reference. Also, please find some more references below:

[1] Howie Choset, Robotic Motion Planning: Potential Functions.
https://www.cs.cmu.edu/~motionplanning/lecture/Chap4-Potential-Field_howie.pdf

[2] Michael A. Goodrich, Potential Fields Tutorial:
https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.115.3259&rep=rep1&type=pdf

[3] Ding Fu-guang, Jiao Peng, Bian Xin-qian and Wang Hong-jian, "AUV local path planning based on virtual potential field," IEEE International Conference Mechatronics and Automation, 2005, 2005, pp. 1711-1716 Vol. 4, doi: 10.1109/ICMA.2005.1626816.
https://ieeexplore.ieee.org/document/1626816

[4] Steven M. LaValle, Chapter 11. Planning With Dynamics and Uncertainty
http://motion.cs.illinois.edu/RoboticSystems/PlanningWithDynamicsAndUncertainty.html

[5] Sertac Karaman and Emilio Frazzoli, “Sampling-based Algorithms for Optimal Motion Planning”, International Journal of Robotics Research, 30(7), pages 846-894, 2011
https://arxiv.org/pdf/1105.1186.pdf