Robot Motion Planning

EN.530.663: Professor Jin Seob Kim
Shashank Goyal, Shuhang Xu, Steve Liu

Task 1: Planar Serial Manipulator

In Task 1, the challenge is to implement path planning for an n-link planar serial manipulator. The manipulator operates in a 2D environment populated with multiple convex-polygonal obstacles. The primary objective is to compute a collision-free path from a specified initial configuration to a goal configuration using two distinct motion planning algorithms: PRM and RRT.

- Manipulator Configuration:
  - **Number of Links (n):** We can select from {4, 5}, indicating the number of joints and hence the complexity of the manipulator.
  - **Link Properties:** Each link is modeled as a line segment, simplifying the geometric representation and collision detection process.
  - **Obstacles:** At least three convex-polygonal obstacles are present in the workspace, which have to be avoided by the manipulator.

- Path Planning Tasks:
  - **PRM Algorithm:** Implement the PRM algorithm to generate a roadmap of possible paths and then determine the optimal path to the goal avoiding obstacles.
  - **RRT Algorithm:** Implement the RRT algorithm to dynamically explore the workspace and generate a path from the initial to the goal configuration.

- Algorithmic Details:

  Probabilistic Roadmap (PRM) -
  - **Initialization:** Sample n configurations in the configuration space avoiding collisions with obstacles.
  - **Roadmap Construction:** Connect each configuration to K-nearest neighbors ensuring path segments do not intersect obstacles.
  - **Path Finding:** Utilize graph search techniques to find the shortest path from the initial to the goal configuration on the roadmap.

Rapidly-exploring Random Trees (RRT) -
- **Tree Initialization:** Start the tree with the initial configuration as the root.
- **Tree Expansion:** Iteratively extend the tree towards randomly sampled configurations using a specified step size, ensuring new branches do not intersect with obstacles.
- **Goal Reaching:** Once the tree reaches near the goal configuration within a defined tolerance, trace back the path to the initial configuration.

- Modeling and Simulation:
  - The manipulator's joints and links are assumed to be fully rotative, ignoring self-collision for simplification.
  - Obstacle positions and dimensions, along with manipulator link lengths and initial and goal configurations, need to be defined explicitly in the code.

- Collision Detection:
  Implemented functions to check if a given manipulator configuration or a proposed move between configurations intersects with any workspace obstacle.

- Environment Setup:
  The workspace should be modeled as a 2D grid where each obstacle is represented as a polygon. The manipulator's movement is confined within predefined workspace boundaries.

- Inverse Kinematics Control:
  - Jacobian Matrix (J): It relates the rate of change of the joint angles to the rate of change of the end-effector's position. It is defined as:

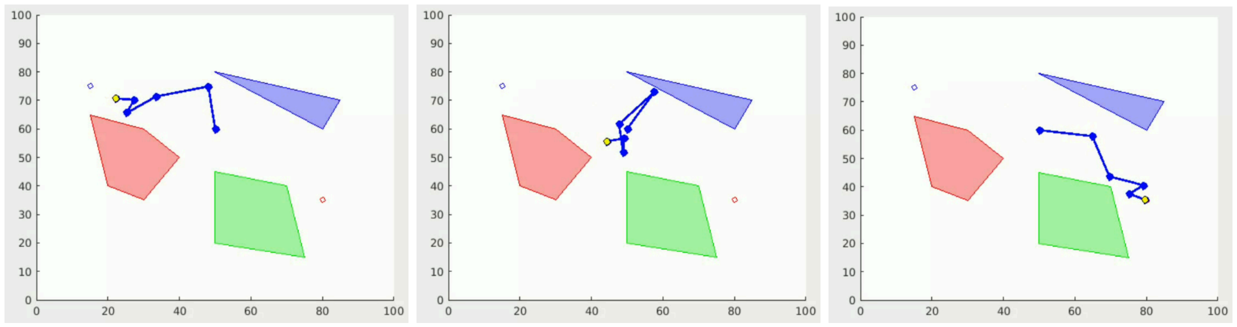
$$J = \frac{\partial[x, y]}{\partial[\theta_1, \theta_2, \ldots, \theta_n]}$$

  - Error Calculation: The error vector is the difference between the target position of the end-effector and its current position.
  - Iterative Update: The joint angles are updated iteratively using the pseudo-inverse of the Jacobian to move the end-effector toward the target.

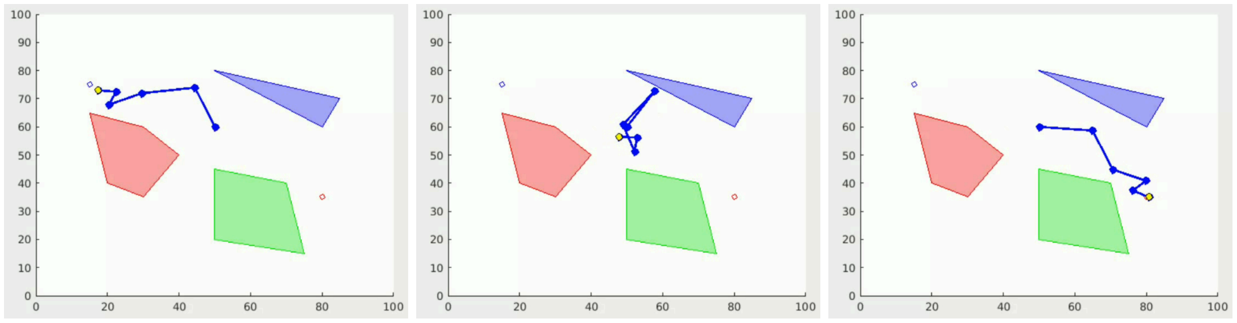
$$\Delta\theta = J^{+} \cdot \text{error}$$

- <u>Results:</u>

Probabilistic Roadmap



Rapidly-exploring Random Trees



Please refer to Task1.mp4 for the complete animation and README.md for an overview of the code. Below are the parameters corresponding to the above results -

- Initial Position: [15, 75]
- Goal Position: [80, 35]
- Obstacles -
    - Blue Triangle: [50, 80, 85; 80, 60, 70]
    - Green Quadrilateral: [50, 75, 70, 50; 20, 15, 40, 40]
    - Red Pentagon: [30, 15, 20, 30, 40; 60, 65, 45, 35, 50]
- PRM -
    - N: 1000
    - K: 15
- RRT -
    - N: 5000
    - Delta_q: 1
    - Tolerance: 3

Task 2

2.1 Equations and formulas used in task 2

$U(q) = U_{att}(q) + U_{rep}(q)$

$$Attractive\ Potential\ Function:\ \Delta U_{att}(q) = \zeta(q - q_G)\ when\ \rho(q, q_G) \le \rho^*_G$$
$$= \frac{\rho^*_G \zeta(q-q_G)}{\rho(q,q_G)}\ when\ \rho(q, q_G) > \rho^*_G$$

$where\ \rho^*_G\ is\ the\ threshold\ distance\ from\ the\ goal.$

$$Repulsive\ Potential\ Function:\ \Delta U_{rep}(q) = \eta\left(\frac{1}{Q^*} - \frac{1}{\rho(q)}\right)\frac{1}{\rho^2(q)}\Delta\rho(q)\ when\ \rho(q) \le Q^*$$
$$= 0\ when\ \rho(q) > Q^*$$

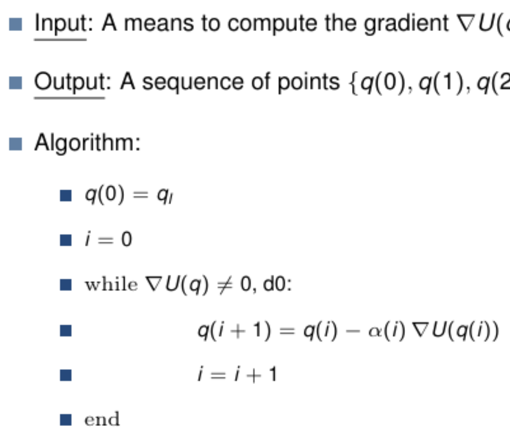
$where\ \Delta\rho(q) = \frac{q-c}{\rho(q,c)}\ and\ c\ is\ the\ closest\ point\ on\ C_{obs}\ to\ q.$

$Since\ we\ are\ working\ on\ a\ rigid\ body\ (rectangle\ robot\ car),\ we\ need\ to\ treat\ gradient\ as\ force.$

$Let\ f, u\ be\ the\ forces\ in\ W, C,\ respectively,\ then\ u = J^T f\ where\ J\ is\ the\ jacobian\ of\ the\ object.$

$For\ our\ program,\ we\ use\ 4\ control\ points\ to\ generate\ force\ vectors.$

$When\ we\ find\ U_{att}\ and\ U_{rep},\ we\ follow\ the\ pseudocode\ in\ lecture\ slides\ to\ find\ the\ path$

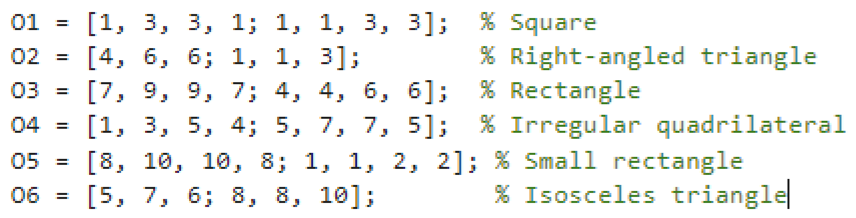


- Input: A means to compute the gradient $\nabla U(q)$ at q.
- Output: A sequence of points $\{q(0), q(1), q(2), \cdots q(i), \cdots\}$
- Algorithm:

```
q(0) = q_I
i = 0
while ∇U(q) ≠ 0, d0:
        q(i+1) = q(i) − α(i) ∇U(q(i))
        i = i + 1
end
```

2.2 Details of the System

- Dimension of obstacles
    - We create six obstacles with different size and shape for testing

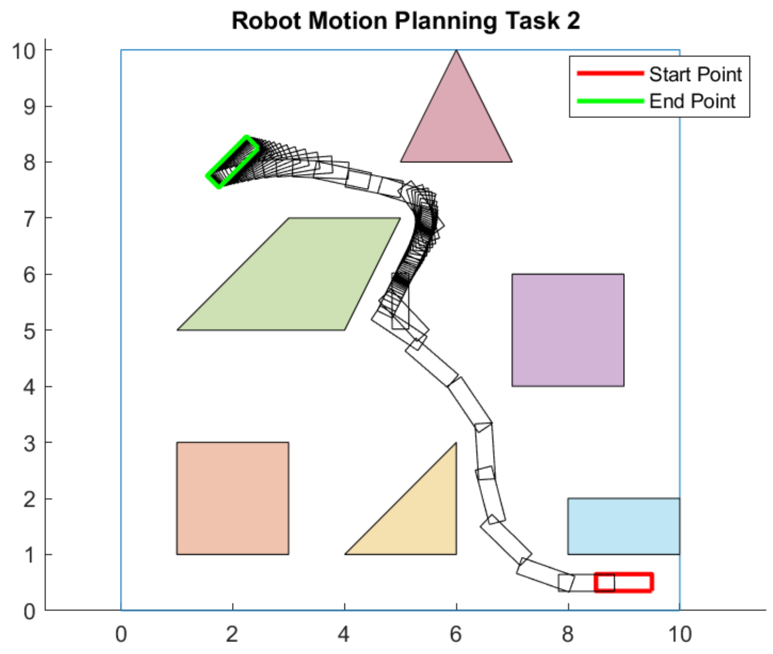


```
O1 = [1, 3, 3, 1; 1, 1, 3, 3];  % Square
O2 = [4, 6, 6; 1, 1, 3];        % Right-angled triangle
O3 = [7, 9, 9, 7; 4, 4, 6, 6];  % Rectangle
O4 = [1, 3, 5, 4; 5, 7, 7, 5];  % Irregular quadrilateral
O5 = [8, 10, 10, 8; 1, 1, 2, 2]; % Small rectangle
O6 = [5, 7, 6; 8, 8, 10];        % Isosceles triangle
```

- Dimension of world
    - 2D world range from [0,10]

- Dimension of robot
    - 1L x 0.3W
- Start point
    - (9,0.5) and parallel with x-axis
- End point
    - (2,8) and 45 degree with respect to x-axis

2.3 Resulting Path Plots



Please refer to Task2.mp4 for the complete animation

Task 3

3.1 Kinematics Equations

Variables and Parameters:
u_ϕ: Wheel rotation rate
u_ω: Angular velocity
r: Radius of the wheel
The linear speed of the wheel v_s is given by the formula:

$$u_s = r \cdot u_\phi$$

Kinematic Model Equations:
Using the parameters above, the kinematics of the wheel can be described by the following equations:

$$\dot{x} = r \cdot u_\phi \cdot \cos(\theta)$$

$$\dot{y} = r \cdot u_\phi \cdot \sin(\theta)$$

$$\dot{\theta} = u_\omega$$

Where θ is the angle of rotation, and $\dot{x}$, $\dot{y}$, and $\dot{\theta}$ represent the derivatives of x, y, and θ with respect to time.

The velocity $\dot{q}$ can be expressed in terms of angular velocity u_ϕ and the radius r as follows:

$$q = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

$$\dot{q} = \begin{bmatrix} r \cdot \cos(\theta) & 0 \\ r \cdot \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} u_\phi \\ u_\omega \end{bmatrix}$$

define $\dot{q} = B(u) \cdot u$
Lie Group Approach:
Transformation Matrix
The transformation matrix G for the system is defined as:

$$G = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Jacobian Matrix
The Jacobian matrix J for the system is defined as:

$$J^b = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Twist Vector:

The twist vector V uses the equation:

$$V^b = (g^{-1} \cdot \dot{g})^v = \begin{bmatrix} v \\ w \end{bmatrix}$$

$$\begin{bmatrix} v \\ w \end{bmatrix} = J^b(q) \cdot B(u) \cdot u = \begin{bmatrix} u_\phi \cdot (r \cdot \cos(\theta)^2 + r \cdot \sin(\theta)^2) \\ u_w \end{bmatrix}$$

$$= \begin{bmatrix} r \cdot \cos(\theta)^2 + r \cdot \sin(\theta)^2 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} u_\phi \\ u_\omega \end{bmatrix}$$

Using the Jacobian transformation, the kinematic equations can be expressed in a compact form, taking into account the rotation and translation movements of the wheel.

3.2 Details of the System

The model uses the height and the base width of the isosceles triangle. With the input of height, width, and tip positions (detailed in *find_points.m*), all vertices can be calculated. The path is generated by the following algorithms: firstly, all potential paths for each step will be calculated, the paths with collision or out of the workspace are excluded, finally, search path by RRT. The process keeps looping while reaching the goal.

The details of the system are described hereafter:
Geometry parameters:

$height = 1.6$
$width = 1.0$
$tip = [1.6; 0]$

General parameters:

$u_{\emptyset} = 1$
$u_w \in \{0, -\frac{\pi}{6}, \frac{\pi}{6}\}$
$r\ (radius) = 0.8$
$u_s\ (Linear\ speed) = 0.8$
$maximum\ number\ of\ RRT = 500$

Test case1:

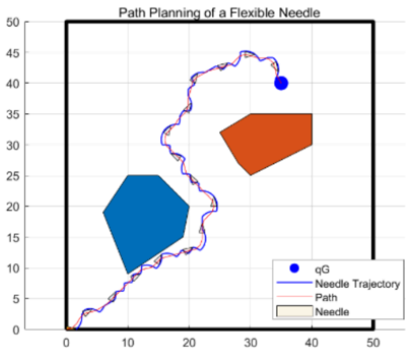
$workspace: rectangle\ [0\ 50\ 50\ 0; 0\ 0\ 50\ 50]$
$obstacles1: [10\ 19\ 20\ 15\ 10\ 6; 9\ 15\ 20\ 25\ 25\ 19]$
$obstacles2: [30\ 40\ 40\ 30\ 25\ 28; 25\ 30\ 35\ 35\ 32\ 27]$
$qI = [0;\ 0;\ 0]$
$qG1 = [35; 40]\ qG2 = [10; 35]\ qG3 = [45; 35]$

Test case2:

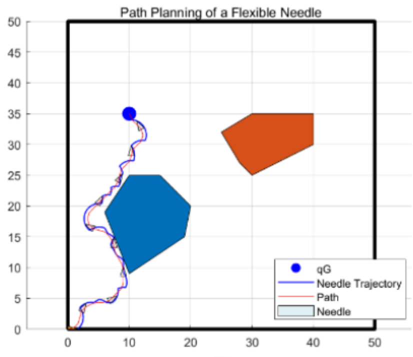
$workspace: rectangle\ [0\ 50\ 50\ 0; 0\ 0\ 50\ 50]$
$obstacles1: [10\ 15\ 20\ 15\ 10\ 5;\ 15\ 15\ 20\ 25\ 25\ 20]$
$obstacles2: [30\ 40\ 40\ 30\ 25;\ 30\ 30\ 35\ 35\ 32]$
$obstacles3: [25\ 30\ 40\ 45\ 40\ 30;\ 20\ 15\ 15\ 20\ 25\ 25]$
$qI = [0;\ 0;\ 0]$
$qG1 = [35; 40]\ qG2 = [30; 27]\ qG3 = [48; 25]$

3.3 resulting path plots (Some frames are dropped for neat display)

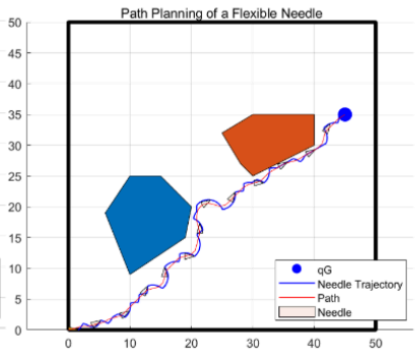
Test case1 (A: qG1; B: qG2; C: qG3):



(A)

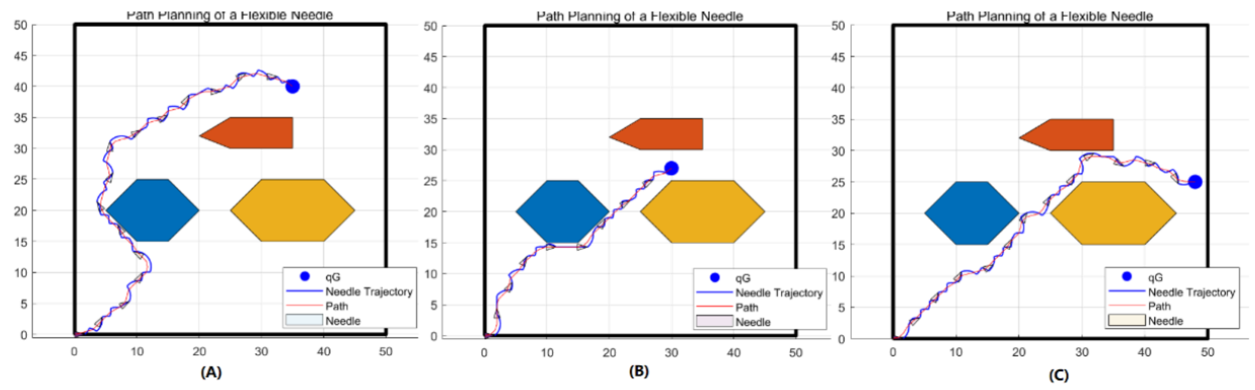


(B)



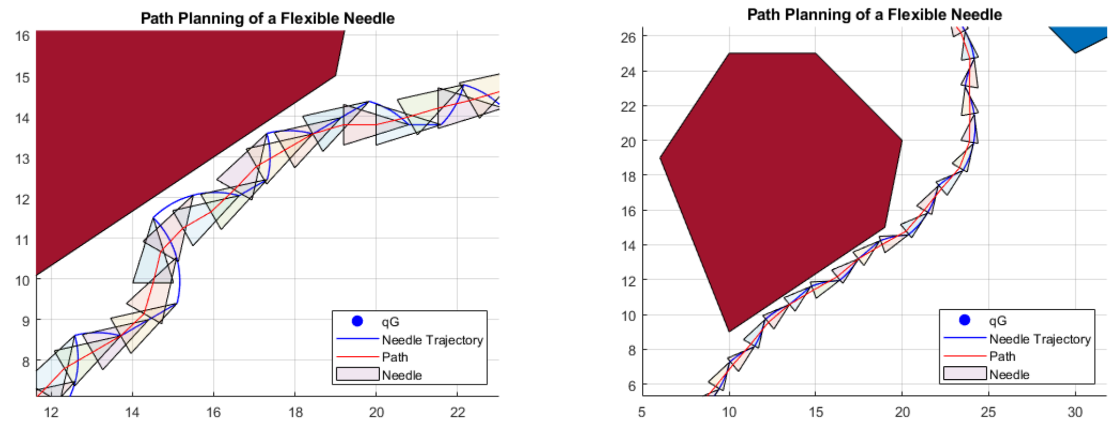
(C)

Test case2 (A: qG1; B: qG2; C: qG3):



3.4 trajectory of the needle

For the given parameters in previous test cases, the needle accurately reaches the target point. However, in certain instances, the needle's trajectory exhibits a zig-zag pattern as shown in the left figure below, even when the intended path is straight. To achieve a smoother trajectory, we can adjust the needle's linear speed and reduce its angular speed. Specifically, in test case 2, by increasing uφ to 2 and decreasing uω to 20 degrees, with the same qG, the needle's trajectory visibly smooths out as shown in the right figure below. Nevertheless, simply modifying the linear and angular speeds may complicate path planning, necessitating multiple algorithm runs to find an optimal path. Thus, it is crucial to find a balance between trajectory smoothness and the efficiency of algorithm execution.



Task 4 (Extra Credit): Grid-Based Maze Generation and Pathfinding Performance Evaluation

The primary objective of this task is to evaluate the effectiveness of three different pathfinding algorithms: Manhattan A*, Euclidean A*, and Dijkstra on a series of grid-based mazes. Each algorithm's performance is measured in terms of computation time, path length, and the number of steps required to find a path from a specified start point to a goal point.

Methods and Materials

- **Grid Setup:** Grids of varying sizes are generated to test the pathfinding algorithms. Each grid is initialized as an empty space (⬜) with obstacles (⬛) randomly placed except for a guaranteed path generated from start to goal to ensure solvability.
- **Heuristics Used:**
    - **Manhattan Heuristic:** Uses the sum of the absolute differences of the cartesian coordinates.
    - **Euclidean Heuristic:** Uses the straight-line distance between two points.
    - **Dijkstra's Algorithm:** Uses no heuristic (uniform cost search).
- **Environment:** Each grid starts with the robot (🤖) at the bottom-left corner and the goal (❤️) at the top-right corner.

Implementation Details

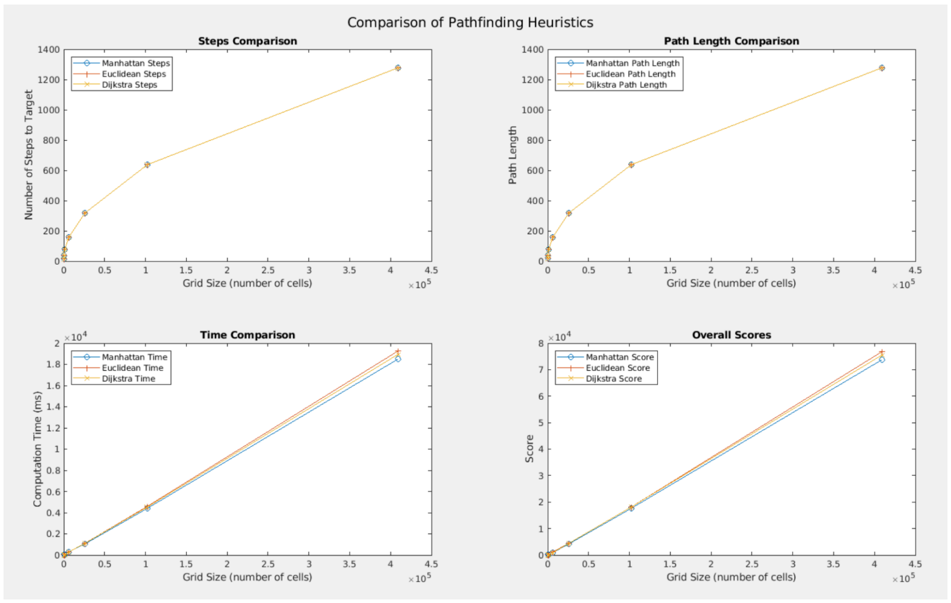
- **Pathfinding Algorithms:**
    - **Initialization:** Each grid is processed to initialize the start and goal positions.
    - **Performance Metrics Calculation:** The algorithms are timed, and their path length and step count are recorded for analysis.
- **Performance Metrics:**
    - **Steps:** Number of moves from start to goal.
    - **Path Length:** Geometric distance covered by the path.
    - **Computation Time:** Time taken to compute the path.
    - **Overall Score:** Calculated as the product of time, steps, and path length normalized by grid size, providing a measure of efficiency.
- **Data Collection:**
    - **Grid Sizes:** {10, 20, 40, 80, 160, 320, 640} cells.
    - **Number of Obstacles:** Twice the size number for each grid to maintain a consistent obstacle density.

Results and Discussion

- **Performance Analysis:** The algorithms were tested across grids of increasing size to observe scalability and performance under more complex scenarios.
    - **Step Count Analysis:** Indicates efficiency in pathfinding with fewer steps denoting more direct routes.
    - **Path Length Analysis:** Assesses the directness and efficiency of the path.
    - **Time Analysis:** Reflects algorithm efficiency under computational constraints.
    - **Overall Score:** Combines all metrics to provide a holistic view of algorithm performance.
- **Visualizations:** Four plots were generated to compare the algorithms across all measured metrics. These visualizations help in identifying trends and performance bottlenecks.

Please refer to the README for an overview of the code.

Conclusion



Graph 1: Steps Comparison

- **Trend:** All three algorithms show an increasing trend in the number of steps as the grid size increases.
- **Analysis:** This is expected as larger grids likely require longer paths. The steps taken by Dijkstra seem slightly higher or on par with Manhattan and Euclidean, suggesting it may not always find the most direct path, likely due to its exhaustive nature.

Graph 2: Path Length Comparison

- **Trend:** Similar to the steps, the path length for all algorithms increases linearly with grid size.
- **Analysis:** This indicates that all three algorithms are effective at scaling with grid size. The path lengths remain very close across the algorithms, which suggests that each is capable of finding nearly optimal paths in terms of physical distance.

Graph 3: Time Comparison

- **Trend:** The computation time increases linearly for all algorithms as the grid size increases.
- **Analysis:** Notably, the times for the three algorithms are closely aligned, with Dijkstra's algorithm occasionally requiring marginally more time. This could be due to its non-heuristic nature requiring more computation to assess paths.

Graph 4: Overall Scores

- **Trend:** The scores of all three algorithms increase linearly with grid size, remaining relatively close to each other.
- **Analysis:** This metric, combining time, steps, and path length relative to the grid size, suggests that all algorithms perform similarly under the tested conditions. The close scores indicate that no single algorithm consistently outperforms the others across the different grid sizes.

Additional Note: To maintain the integrity of this comparative analysis, no parameters or system performance characteristics were altered during the tests. All algorithms were run under identical conditions, including any background tasks and system states, to ensure that the observed differences and behaviors are solely attributable to the algorithmic implementations and not to external factors

Work Contributions -

- Shashank Goyal: Task 1 and Task 4 (and respective Report Sections)
- Shuhang Xu: Task 2 (and respective Report Sections)
- Steve Liu: Task 3 (and respective Report Sections)