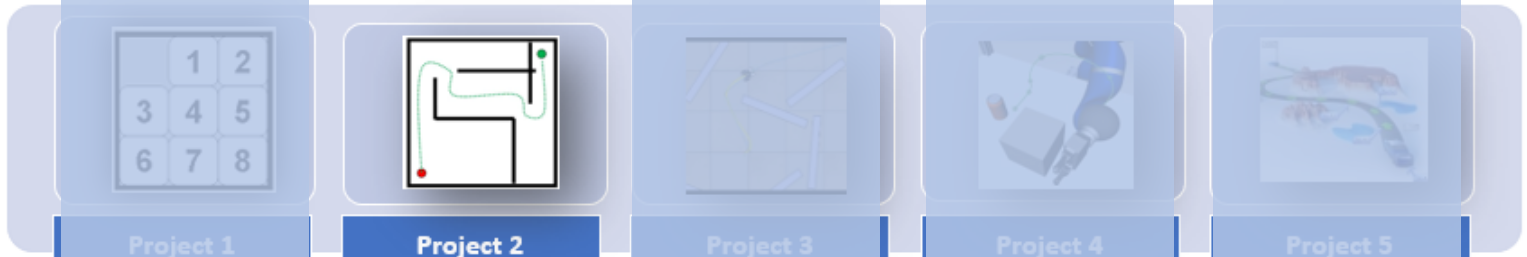
Project 2: Implementation of Dijkstra algorithm for a Point Robot

This is an Individual project.
Due Date – March 5th, 11.59 PM



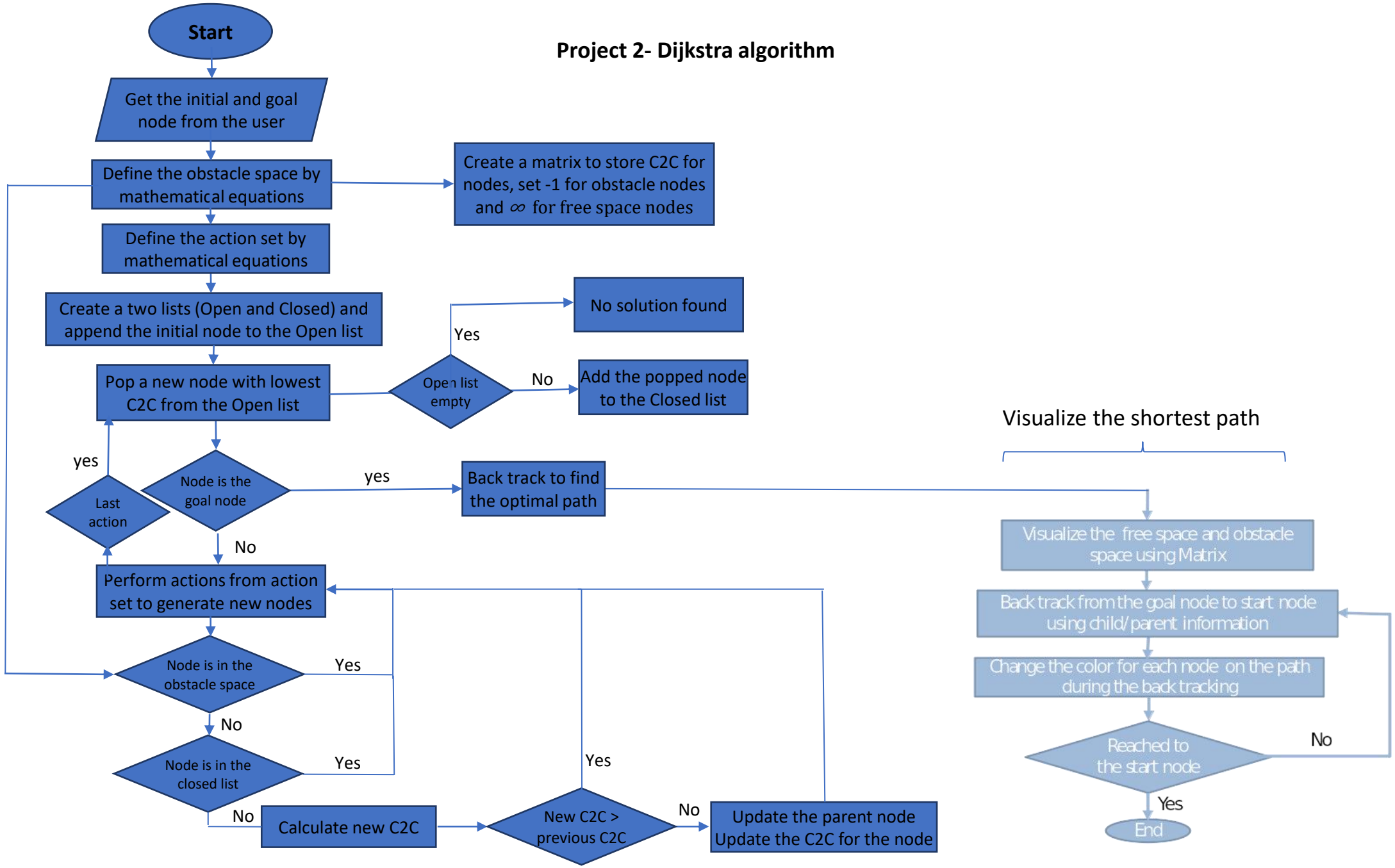
	Project 1 8-puzzle problem	Project 2 Point robot	Project 3 Mobile robots	Project 4 Manipulators	Project 5 Final project
Step 1	Define actions set & Generating tree	Define actions set & Generating tree	Define actions set & Generating tree	Learn V-rep and Move it/Gazebo	Define actions set & Generating tree
Step 2	Implement simple Search algorithm*	Implement Dijkstra search algorithm	Implement A* Search algorithm	Use V-REP and MOVE IT for path planning	Implement RRT and A* for your application
Step 3	queue		Implement: 1) nonholonomic cons. 2) Obstacle avoidance	Use these modules: 1) Inverse kinematics 2) Collision detection	Suggest incremental innovations for published papers
				Implement project 3 or 4 on a real robot	Write a paper in IEEE format
Step 5				Robotic Competition*	15- minute Presentation
Dates	MM/DD	MM/DD	MM/DD	MM/DD	MM/DD
Deliverables	- Source code - Output file	- Source code - Output file	- Source code - Output file - Visualization of the path	- Environment - Video file	- Source code - Output file - Paper - PPT file

*Optional

Dijkstra algorithm - pseudo code

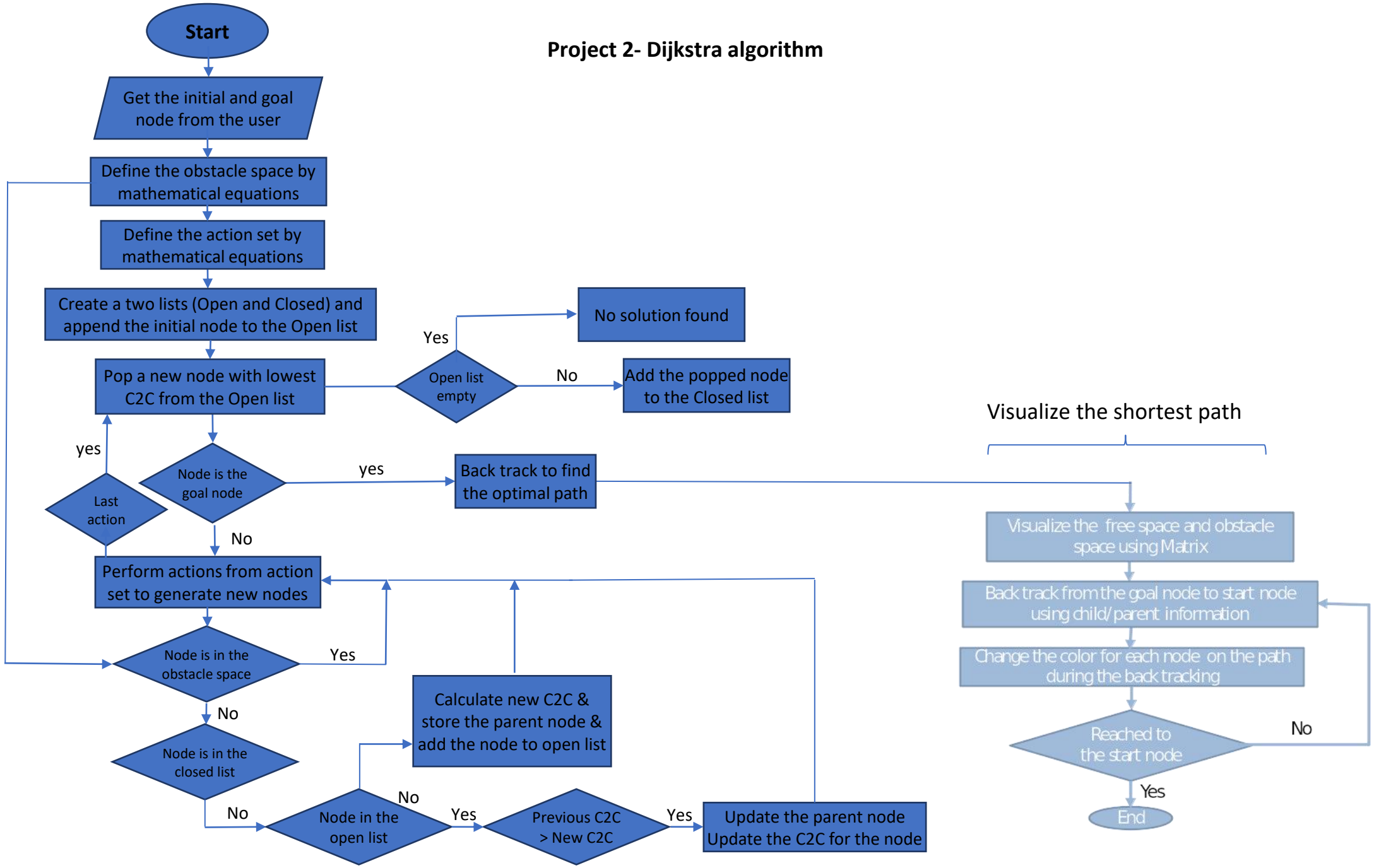
```
Create two empty lists named OpenList and ClosedList
Get the initial (Xi) and goal node (Xg) from the user
OpenList.put(Xi)
While (OpenList not empty) and (Not reached the goal) do
    x ← OpenList.Get ()
    Add x to ClosedList
    If x = Xg
        Run backtrack function
        Return SUCCESS
    else
        Forall u ∈ U(x)
            x' ← f(x,u)
            if (x' ∉ ClosedList) and (NOT in the obstacle space)
                if (x' ∉ OpenList) or (CostToCome(x')=∞)
                    Parent(x') ← x
                    CostToCome(x') ← CostToCome(x) + L(x,u)
                    Cost(x') ← CostToCome(x')
                    OpenList.put(x')
                else
                    If Cost(x') >  CostToCome(x) + L(x,u)
                        Parent(x') ← x
                        CostToCome(x') ← CostToCome(x) + L(x,u)
                        Cost(x') ← CostToCome(x')
Return FAILURE
```

Project 2- Dijkstra algorithm

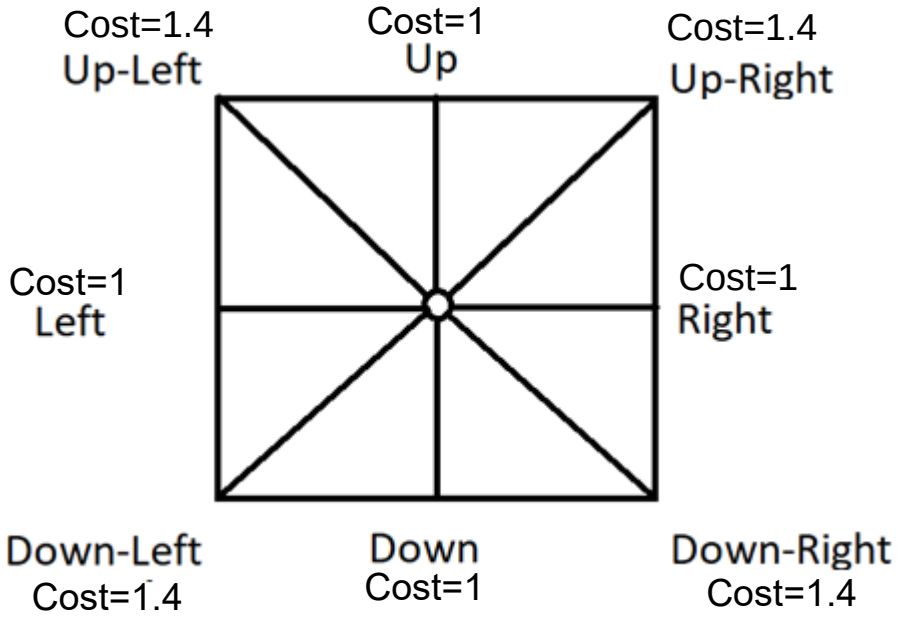


Or

Project 2- Dijkstra algorithm



Project 2 Description



Project Assumption: Workspace is an 8 connected space, that means now you can move the robot in up, down, left, right & diagonally between up-left, up-right, down-left and down-right directions.

Action sets= {(1,0), (-1,0), (0,1), (0,-1), (1,1), (-1,1),(1,-1),(-1,-1)}

Project 2 Description

1) Check the feasibility of all inputs/outputs (if user gives start and goal nodes that are in the obstacle space they should be informed by a message and they should try again).

2) Implement Dijkstra's Algorithm to find a path between start and end point on a given map for a point robot (radius = 0; clearance = 5 mm).

3) Your code must output an animation of optimal path generation between start and goal point on the map. You need to show both the node exploration as well as the optimal path generated. (Some useful tools for simulation are OpenCV/Pygame/Matplotlib).

Visualization

https://drive.google.com/file/d/1OTvRGCmQ35oXbf5HEe70rL6czHS3PJf5/view

Step 1) Define the actions in a mathematical format

- Use can use the same data structure from project 1 to store the node information.
- Write 8 subfunctions, one for each action. The output of each subfunction is the state of a new node after taking the associated action.

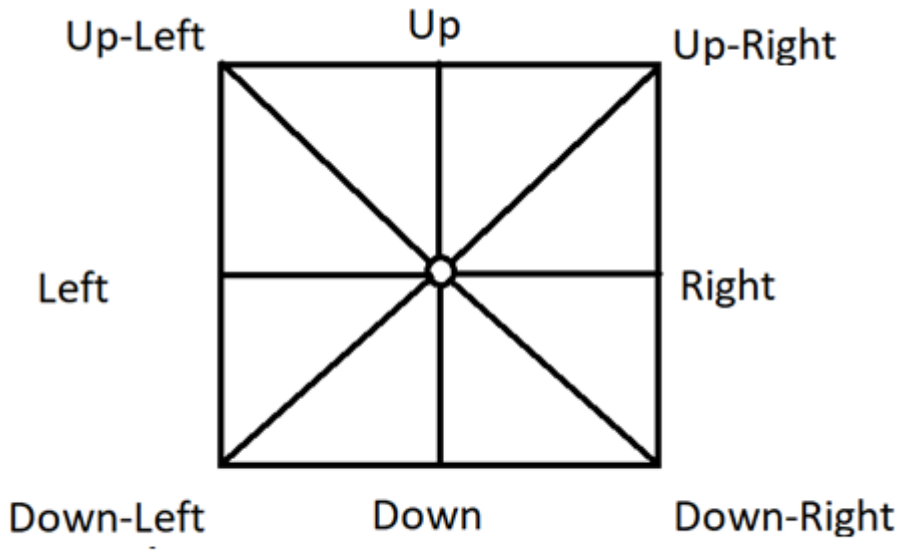
Action sets= {(1,0), (-1,0), (0,1), (0,-1), (1,1), (-1,1),(1,-1),(-1,-1)}

Step 2) Find mathematical representation of free space

- **Use Half planes and semi-algebraic models** to represent the obstacle space.

Step 3) Generate the graph and check for the goal node in each iteration

- Generate the graph using action set for a 8-connected space and save in a data structure format
- Before saving the nodes, check for the nodes that are within the obstacle space and ignore them



Step 4) Find the optimal path (Backtracking)

- Once the goal node is popped, stop the search and back track to find the path

Steps:

- Write a subfunction that compares the current node with the goal node and return TRUE if they are equal.
- While generating each new node this subfunction should be called
- Write a subfunction that once the goal node is reached, using the child and parent relationship, it backtracks from the goal node to initial node and outputs all the intermediate nodes.

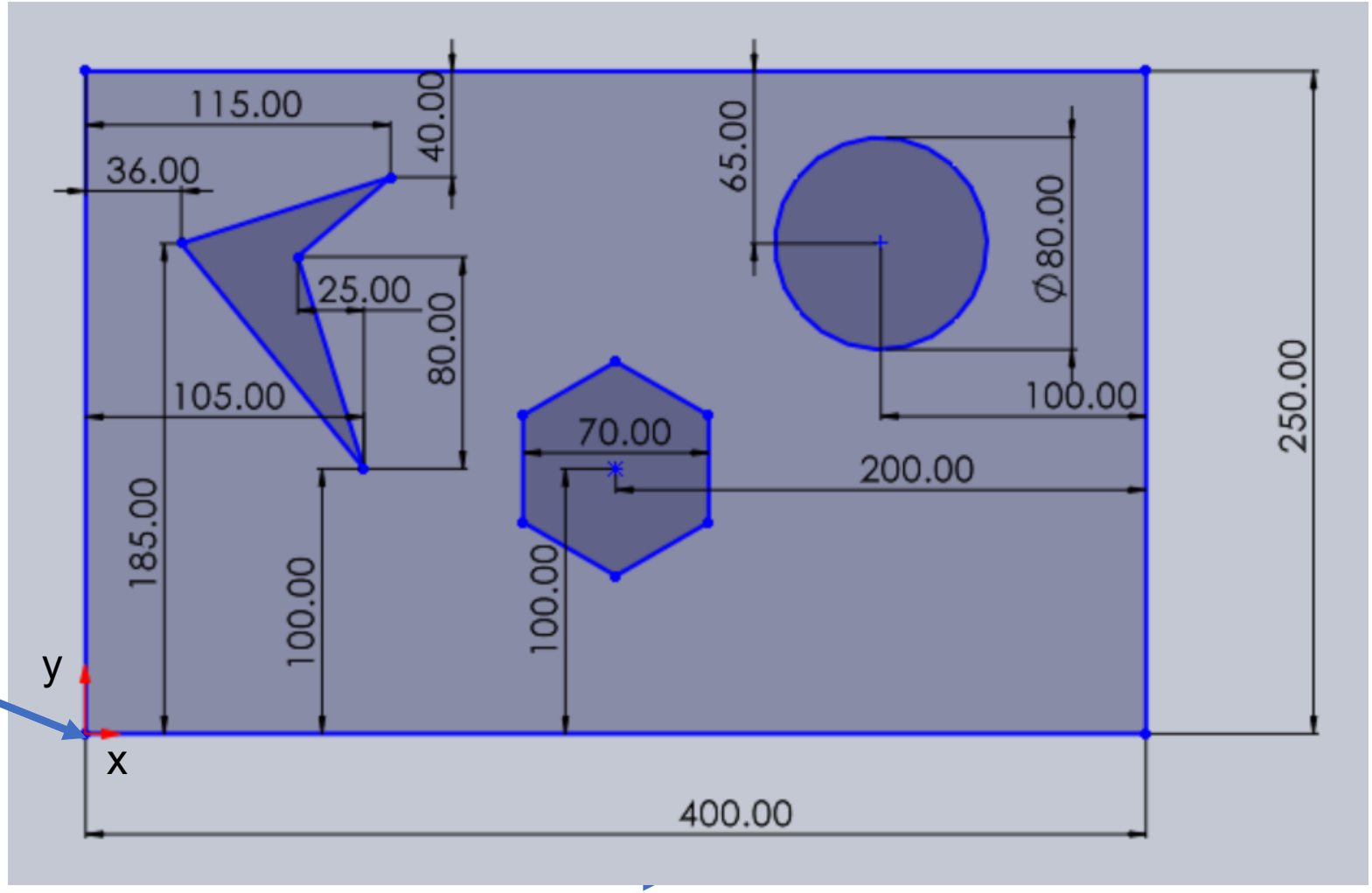
Step 5) Represent the optimal path

- Show optimal path generation animation between start and goal point using a simple graphical interface. You need to show both the node exploration as well as the optimal path generated.

The visualization of (exploration and optimal path) should start only after the exploration is complete and optimal path is found.

Note: A separate document is provided to describe this step

Final Map


115.00
40.00
36.00
65.00
Ø80.00
25.00
80.00
105.00
100.00
70.00
200.00
250.00
185.00
100.00
100.00
Origin
y
x
400.00

Deliverables

Deliverables:

1. ReadMe.txt (Describing how to run the code in a txt format)
2. Source files
   - Dijkstra-pathplanning-StudentFirstName-StudentLastName.py
   - GitHub repository link in the URL submission
   - Video recording (start and goal point can be random)

   Note: The code should accept start and goal points from the user

   Note: Submit all the file as a single zip file. Name of the file should be in this format:

   Project2-StudentFirstName-StudentLastName.zip