ENPM661 - Spring 2024

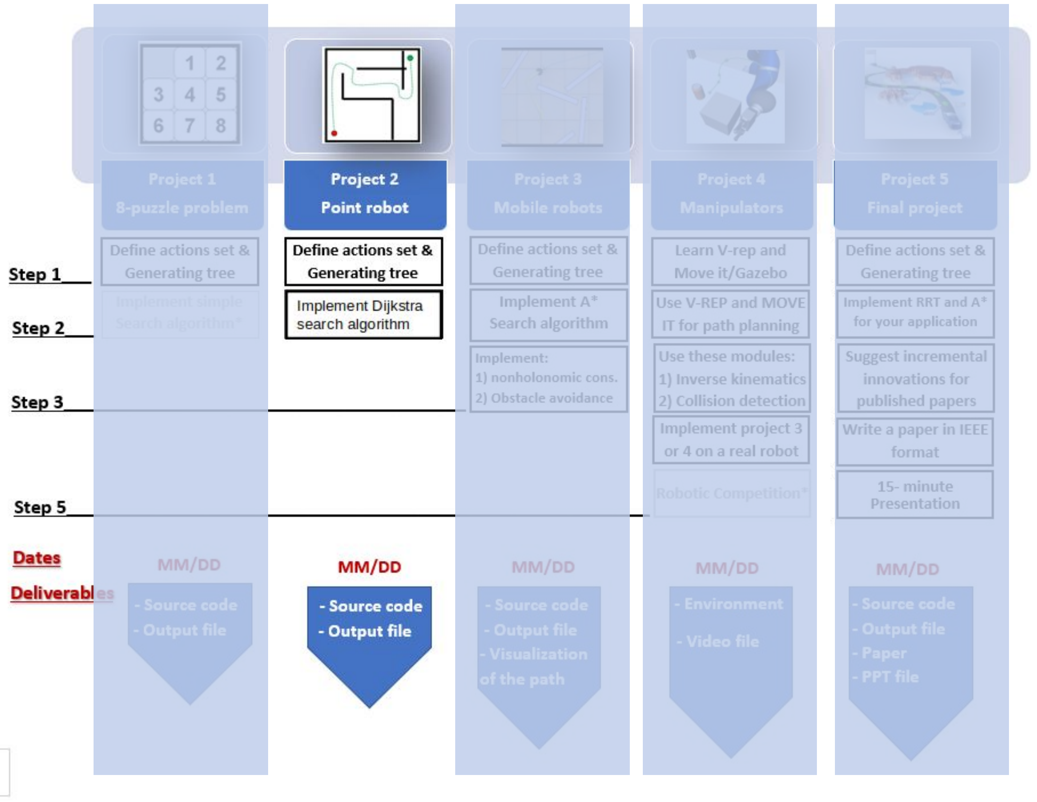
Project 02

Implementation of the Dijkstra Algorithm
for a Point Robot

Note: This is an individual project

Due Date: March 10, 11:59 PM **Points/Weightage: 10**

Overview



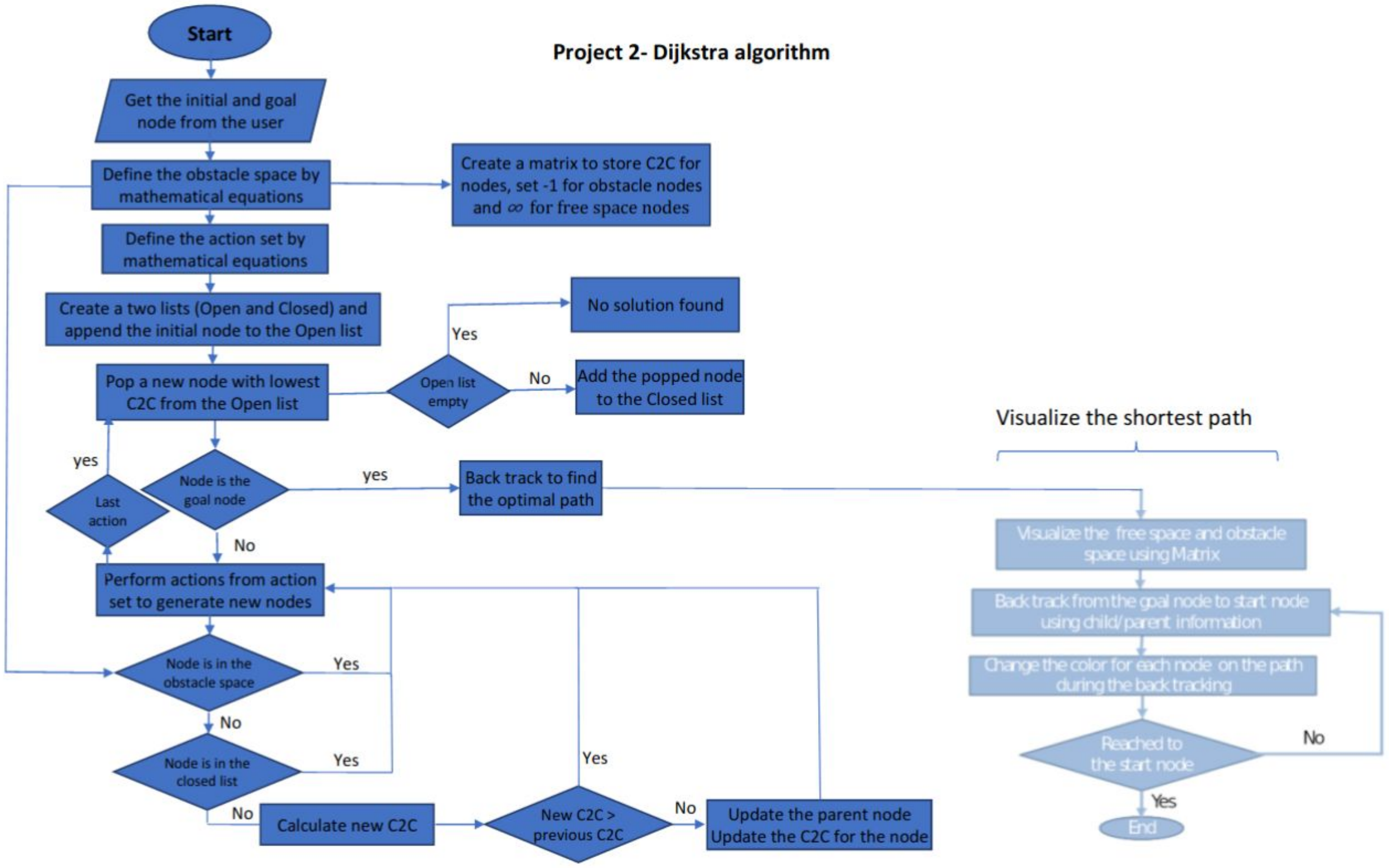
	Project 1 8-puzzle problem	Project 2 Point robot	Project 3 Mobile robots	Project 4 Manipulators	Project 5 Final project
Step 1	Define actions set & Generating tree	Define actions set & Generating tree	Define actions set & Generating tree	Learn V-rep and Move it/Gazebo	Define actions set & Generating 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			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

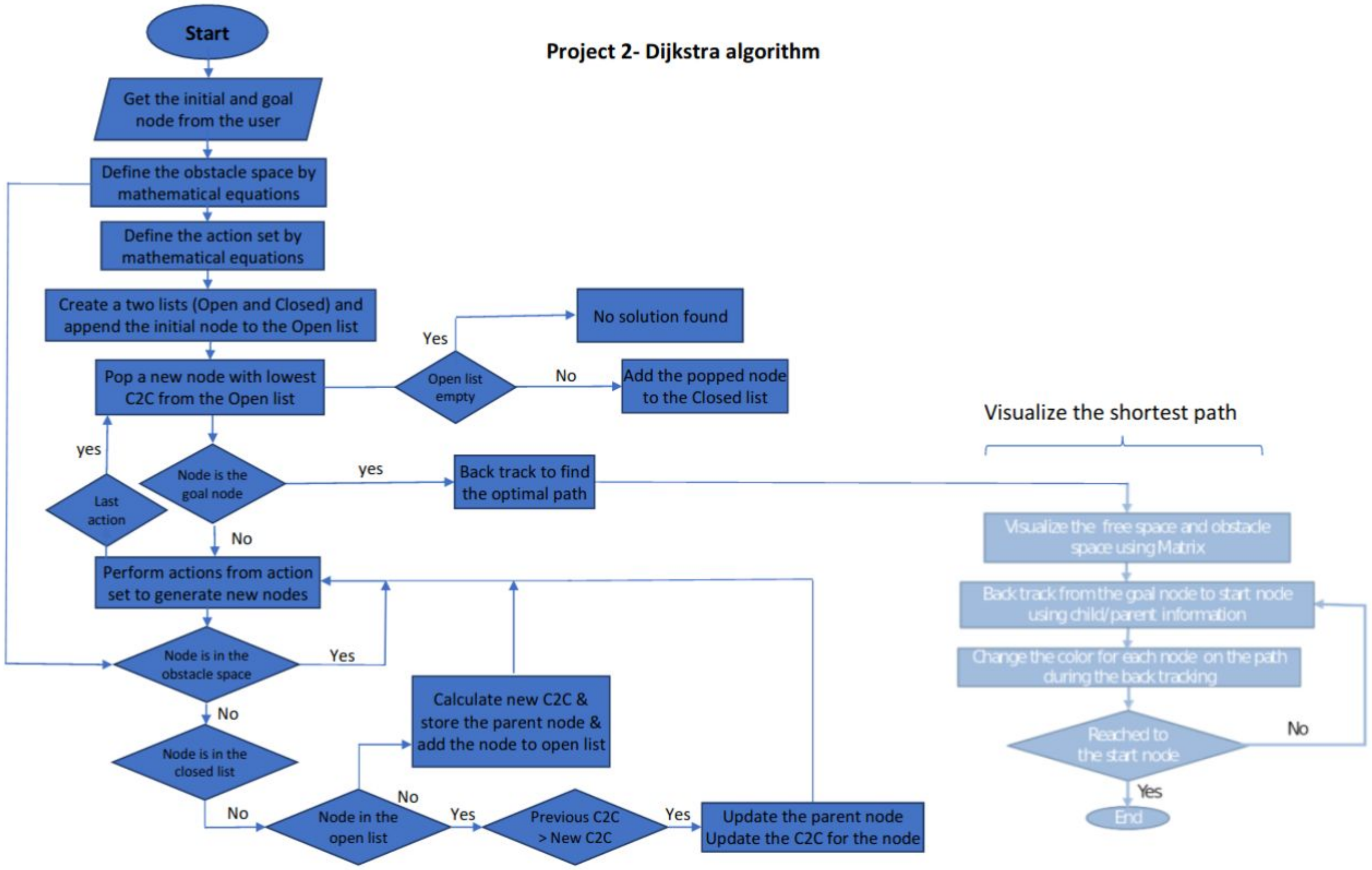
```
Create two empty lists named OpenList and ClosedList
Get the initial ($X_i$) and goal node ($X_g$) from the user
OpenList.put($X_i$)
While (OpenList not EMPTY) and (Not reached the goal) do
      x $\leftarrow$ OpenList.get()
      Add x to ClosedList
      if x = $X_g$
            Run backtrack function
            return SUCCESS
      else
            forall u $\in$ U(x)
                  x’ $\leftarrow$ f(x,u)       # Generating each valid action
                  if (x’ $\notin$ ClosedList) and (NOT in the obstacle space)
                        if (x’ $\notin$ OpenList) or (CostToCome(x’) = $\infty$)
                              Parent(x’) $\leftarrow$ x
                              CostToCome(x’) $\leftarrow$ CostToCome(x) + L(x,u)     # L(x,u) is the cost of the action
                              Cost(x’) $\leftarrow$ CostToCome(x’)
                              OpenList.put(x’)
                  else
                        If Cost(x’) > CostToCome(x) + L(x,u)
                              Parent(x’) $\leftarrow$ x
                              CostToCome(x’) $\leftarrow$ CostToCome(x) + L(x,u)
                              Cost(x’) $\leftarrow$ CostToCome(x’)
return FAILURE
```

Project 02: Flow Chart (Option #1)



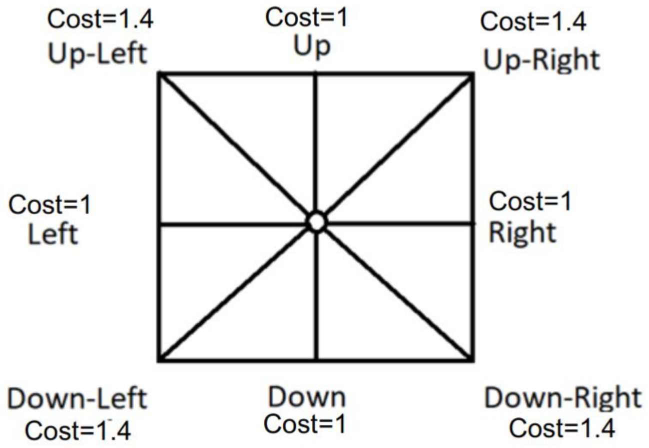
Project 2- Dijkstra algorithm
Start
Get the initial and goal node from the user
Define the obstacle space by mathematical equations
Create a matrix to store C2C for nodes, set -1 for obstacle nodes and ∞ for free space nodes
Define the action set by mathematical equations
Create a two lists (Open and Closed) and append the initial node to the Open list
Pop a new node with lowest C2C from the Open list
Open list empty
Yes
No solution found
No
Add the popped node to the Closed list
Node is the goal node
yes
Back track to find the optimal path
Last action
yes
No
Perform actions from action set to generate new nodes
Node is in the obstacle space
Yes
No
Node is in the closed list
Yes
No
Calculate new C2C
New C2C > previous C2C
Yes
No
Update the parent node Update the C2C for the node
Visualize the shortest path
Visualize the free space and obstacle space using Matrix
Back track from the goal node to start node using child/parent information
Change the color for each node on the path during the back tracking
Reached to the start node
No
Yes
End

Project 02: Flow Chart (Option #2)


Project 2- Dijkstra algorithm
Start
Get the initial and goal node from the user
Define the obstacle space by mathematical equations
Define the action set by mathematical equations
Create a two lists (Open and Closed) and append the initial node to the Open list
Pop a new node with lowest C2C from the Open list
Open list empty
Yes
No solution found
No
Add the popped node to the Closed list
Node is the goal node
yes
Back track to find the optimal path
Last action
yes
No
Perform actions from action set to generate new nodes
Node is in the obstacle space
Yes
No
Node is in the closed list
No
Node in the open list
No
Calculate new C2C & store the parent node & add the node to open list
Yes
Previous C2C > New C2C
Yes
Update the parent node Update the C2C for the node
Visualize the shortest path
Visualize the free space and obstacle space using Matrix
Back track from the goal node to start node using child/parent information
Change the color for each node on the path during the back tracking
Reached to the start node
No
Yes
End

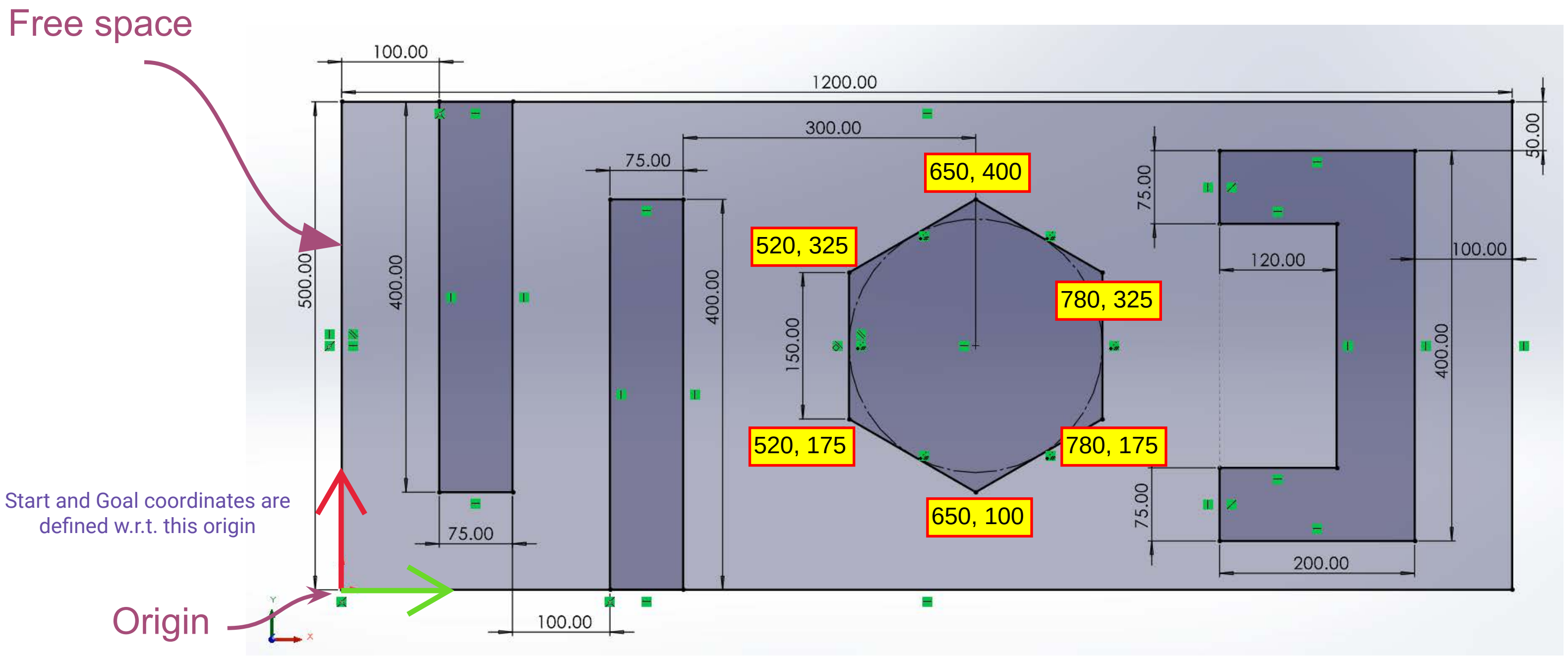
Project 02: Assumptions

- The robot is assumed to be a point robot (size/radius of the robot = 0)
- The robot has a clearance of 5 mm (described in the next page).
- The workspace is an 8-connected space
    - The robot can move UP, DOWN, LEFT, RIGHT [Cost: 1.0]
    - It can also move diagonally between UP-LEFT, UP-RIGHT, DOWN-LEFT, and DOWN-RIGHT [Cost: 1.4]



Actions Set = {(1,0), (-1,0), (0,1), (0,-1), (1,1), (-1,1), (1,-1), (-1,-1)}

Project 02: Map



- The above map represents the space for clearance = 0 mm. For a clearance of 5 mm, the obstacles (including the walls) should be bloated by 5 mm distance on each side.

Project 02: Description

- Check the feasibility of all inputs/outputs
    - If the start and/or goal nodes are in the obstacle space, the user should be informed by a message and **they should input the nodes again** until valid values are entered.
    - The user input start and goal coordinates should be w.r.t. the origin shown in the previous page.
- Implement Dijkstra's Algorithm to find a path between the start and end point on a given map for a point robot (**radius = 0; clearance = 5 mm**).
- 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**.
    - Sample videos from previous years: [Link], [Link]

Project 02: Visualization

- Some useful tools for drawing shapes on a canvas (map generation/visualization):
  - OpenCV
    - https://pyimagesearch.com/2021/01/27/drawing-with-opencv/
    - https://medium.com/analytics-vidhya/opencv-tutorial-drawing-shapes-and-texts-5cba52c76cd0
  - Pygame
    - https://www.geeksforgeeks.org/how-to-draw-rectangle-in-pygame/
    - https://medium.com/theta-hat/using-pygame-to-draw-grided-canvas-ba7e7e409373
    - https://ryanstutorials.net/pygame-tutorial/pygame-shapes.php
  - Matplotlib
    - https://nickcharlton.net/posts/drawing-animating-shapes-matplotlib.html
    - https://www.askpython.com/python/examples/draw-shapes-using-opencv
  - Tkinter
    - https://stackoverflow.com/questions/25701347/how-to-draw-a-line-on-a-canvas
    - https://pythonbasics.org/tkinter-canvas/

Step 01: Define the Actions in a Mathematical Format

- You can use the same data structure from Project 01 to store the node information.
    - Refer Priority Queues/Heap Queues
    - Refer Updating a node information using different data structures.ipynb

- Write 8 functions, one for each action. The output of each function is the state of a new node after taking the associated action. The names of these functions can be as per your wish as long they achieve the intended result.

Actions Set = {(1,0), (-1,0), (0,1), (0,-1), (1,1), (-1,1), (1,-1), (-1,-1)}

Step 02: Find the Mathematical Representation of Free Space

- Use **Half planes and semi-algebraic models** to represent the obstacles in the map.
    - Refer Session 03
    - The equations must account for the 5 mm clearance of the robot.
- Using OpenCV/Matplotlib/Pygame/Tkinter (or any other graphical plotting library of your choice), create an empty canvas of size height=500 and width=1200.
    - With the above equations, assign a different color for all the pixels within obstacles and walls. You may choose to represent the clearance pixels around the obstacle with another color.
    - IMPORTANT: While creating the map/canvas, pay attention to the coordinate system representation of the corresponding library.
        - For example: In OpenCV, the image origin is at the top left corner. Refer: [Link]

Step 03: Generate the Graph and Check for Goal Node in each Iteration

- Generate the graph using the action set for an 8-connected space, similar to as done in Project 01.
- Before saving the nodes, check for the nodes that are within the obstacle space and ignore them.
    - To verify if a specific coordinate (node) is in the obstacle space, simply check its pixel color value in the created map.

Step 04: Optimal Path (Backtracking)

- Once the goal node is popped, stop the search and backtrack to find the path.
    - Write a function that compares the current node with the goal node and return TRUE if they are equal.
        - While generating each new node this function should be called
    - Write a function, when once the goal node is reached, using the child and parent relationship, backtracks from the goal node to start node and outputs all the intermediate nodes in the reversed order (start to goal).

Step 05: 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.
  - **Note: The visualization of (exploration and optimal path) should start only after the exploration is complete and optimal path is found.**
    - To save the visualization animation as a video:
      - OpenCV:
        - https://theailearner.com/2018/10/15/creating-video-from-images-using-opencv-python/
        - https://www.geeksforgeeks.org/saving-a-video-using-opencv/
      - Matplotlib:
        - https://matplotlib.org/stable/gallery/animation/dynamic_image.html
        - https://holypython.com/how-to-save-matplotlib-animations-the-ultimate-guide/
      - Pygame
        - https://pypi.org/project/vidmaker/
      - Tkinter
        - https://www.quickprogrammingtips.com/python/how-to-create-canvas-animation-using-tkinter.html
      - Alternatively, you may run the code + visualization, screen record the animation and save it as a .mp4 file.

Project 02: Deliverables

- README file (.md or .txt)
  - Must describe how to run the code and give inputs (start and goal coordinates)
  - Must mention the libraries/dependencies used (for ex: numpy, matplotlib, etc)

- Source Code (.py)
  - Filename: **dijkstra_firstname_lastname.py**
  - Code should accept start and goal coordinates as a user input
    - Values should be w.r.t the map origin frame
  - GitHub Repository Link containing the code(s)
    - Should contain commits (at least 5) with appropriate commit messages
    - Simple Tutorial: https://www.youtube.com/watch?v=iv8rSLsi1xo

- Animation Video (.mp4)
  - Recording of the Node exploration and Optimal path
  - Start and Goal coordinates can be random

- Submit all the files by dragging and dropping in the ELMS portal. Do not zip the files

Additional information + Tips

- When defining the obstacle space, use any graphing tool (like desmos) to visualize and check the validity of your equations.
    - https://www.desmos.com/calculator
- It is important to keep your code modular so that it can be reused for the upcoming projects.
- Since your code needs to analyze 500x1200 nodes, optimize your code whenever possible.
- Use the time library to print the runtime of your algorithm.
- Since the canvas size is 1200x500, the resolution of the visualization might be low. For this you may choose to scale the size of the canvas (for example, the *resize()* function in the *imutils* library).