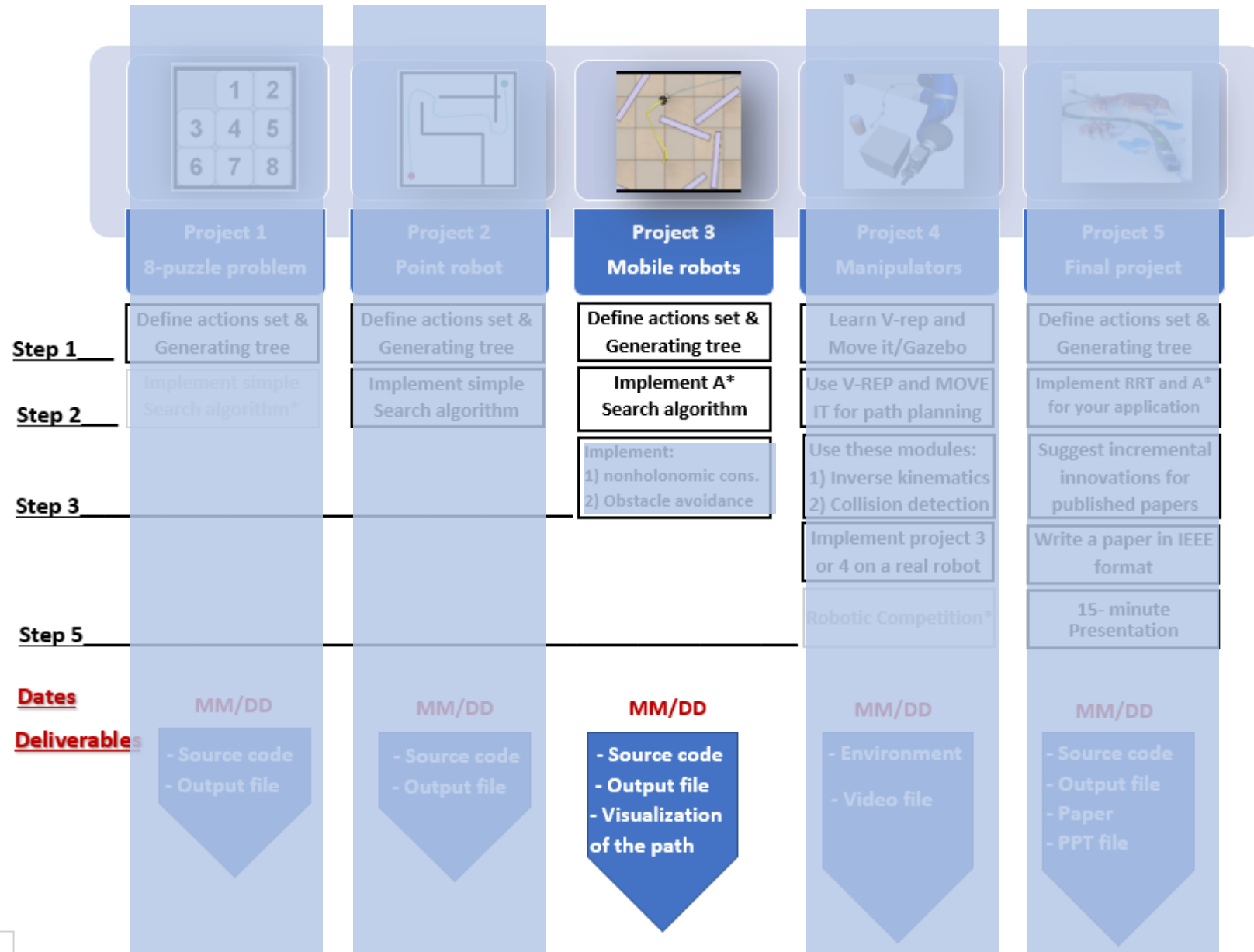# Project 3 Phase 1: Implementation A* algorithm for a mobile Robot

This is a group project (max two students)
Due Date – March 21st, 11.59 PM

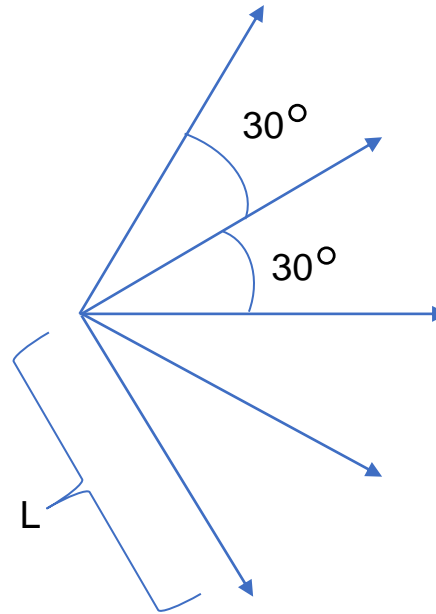# Project3

|	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 simple 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

2

Action Set

To check for duplicate nodes:
- Euclidean distance threshold is 0.5 unit (for x, y)
- Theta threshold is 30 degrees (for Θ)
- Python file provided can be used to plot

30°

30°

L

Consecutive angles are 30 degrees.

Front

Front
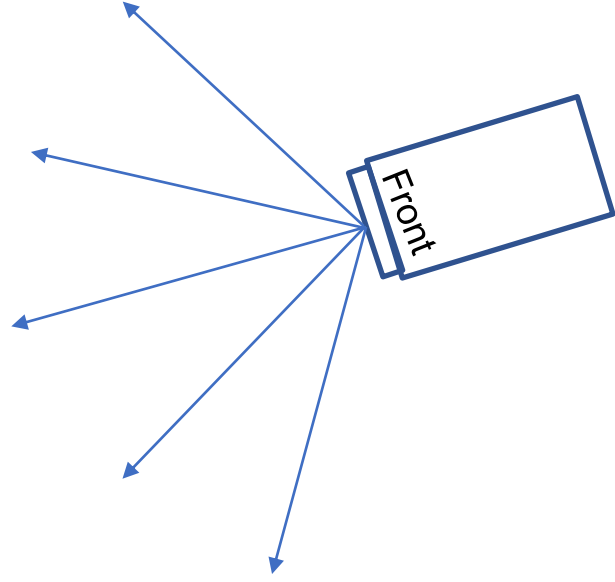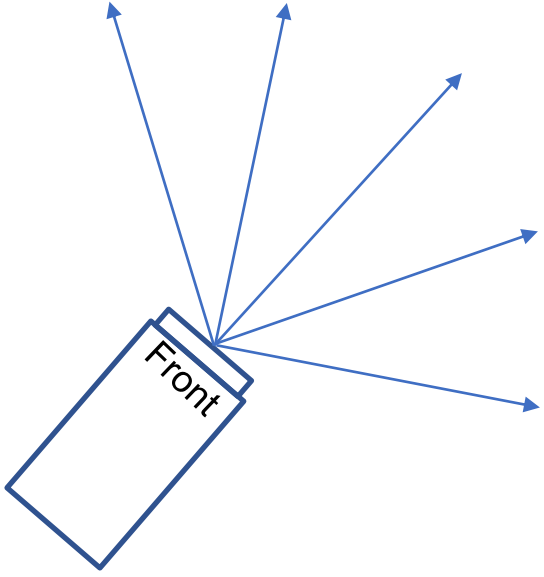
Project 3 Description

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). (retain from previous project)

2) Implement A* Algorithm to find a path between start and end point on a given map for a mobile robot (radius = 10; clearance = 5).

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). (sample code using Matplotlib is provided)

Step 0: Get the Inputs from the User

- Your code must take following values from the user:

1) Start Point Co-ordinates (3-element vector): (Xs,Ys,Theta_s)
2) Goal Point Co-ordinates (3-element vector): (Xg,Yg, Theta_g**)
3) Clearance and robot radius
4) Step size of movement in units (1 <= L <=10)

* Cartesian Coordinates should be used

**Theta_g & Theta in degrees.

Theta_g and Theta

- Theta_g is the orientation at goal point of the robot.

- Theta is the angle between the action set at each node.

- Take these parameters as user inputs (k*30), i.e. {...,-60,-30,0,30,60,...}

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 5 subfunctions, one for each action. The output of each subfunction is the state of a new node after taking the associated action.
- Step size is the length of the vectors.
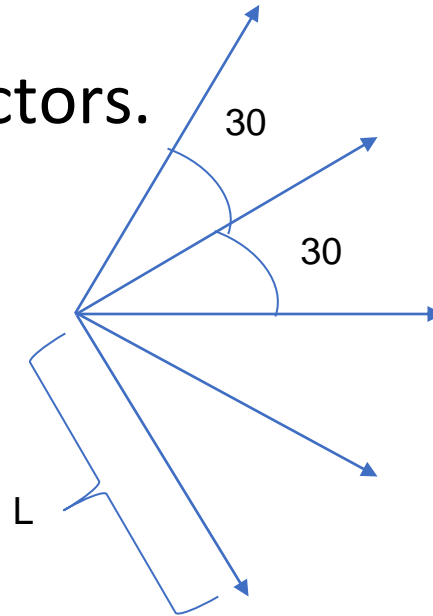
Step 2) Find mathematical representation of free space
(Retain from previous project)

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

Step 3): implement A* search algorithm to search the tree and to find the optimal path

- Forward search

- Consider Euclidean distance as a heuristic function.

- Note - You have to define a reasonable threshold value for the distance to the goal point. Due to the limited number of moves. the robot cannot reach the exact goal location, so program a threshold distance has to be defined.

   Goal threshold (1.5 units radius)

Threshold distance

Goal

Robot
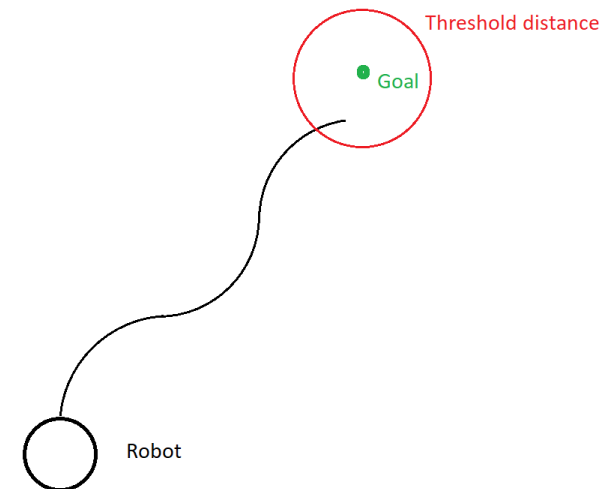
Step 3 (Continue) - Graph
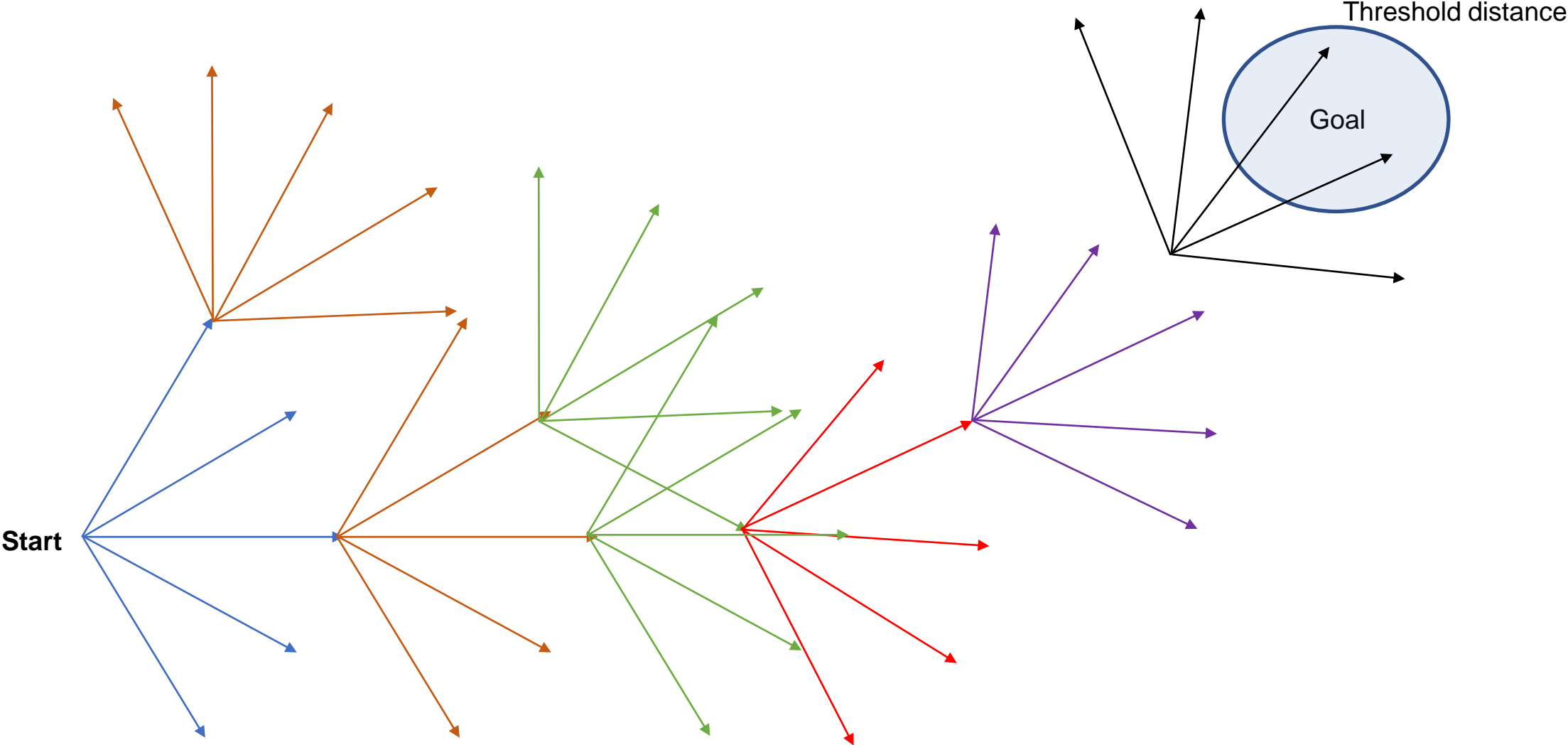
Generate the graph

- Consider the configuration space as a 3 dimensional space.

First method for finding the duplicate node: In order to limit the number of the nodes, before adding the node make sure that the distance of the new node is greater than the threshold in x, y, and theta dimensions with respect to all existing nodes. This method is very slow. **You should avoid using this method.**

Second method for finding the duplicate node: Use a matrix to store the information of the visited nodes. For threshold of 0.5 unit for x and y, and threshold of 30 degree for Theta in the given map, you should use a matrix V with 250/(threshold) by 400/(threshold) by 12 (i.e. 360/30) to store the visited regions information i.e. your matrix dimension will be (500x800x12).

Example:

Set V[i][j][k]=0.

If node1= (3.2, 4.7, 0) visited   → visited region: (3, 4.5, 0) →V[6][9][0]=1

If node2= (10.2, 8.8, 30)   visited → visited region: (10, 9, 30) → V[20][18][1]=1

If node3= (10.1, 8.9, 30)   visited → visited region: (10, 9, 30) → V[20][18][1]=1

(Here node2 and node 3 are duplicate nodes)

Step 4) Find the optimal path (Backtracking)
(Retain from previous project)

- 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
(Retain from previous project)

- 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 will be provided later this coming week to describe this step

Step 5 (continue): Display the graph in the configuration space

- Use a line to connect the new node to previous nodes and display it on the Map as the search space is explored.

- The visualization of the optimal path will start once your algorithm has found the optimal path using A*.

- Exploration and Optimal Path should be in different colors.
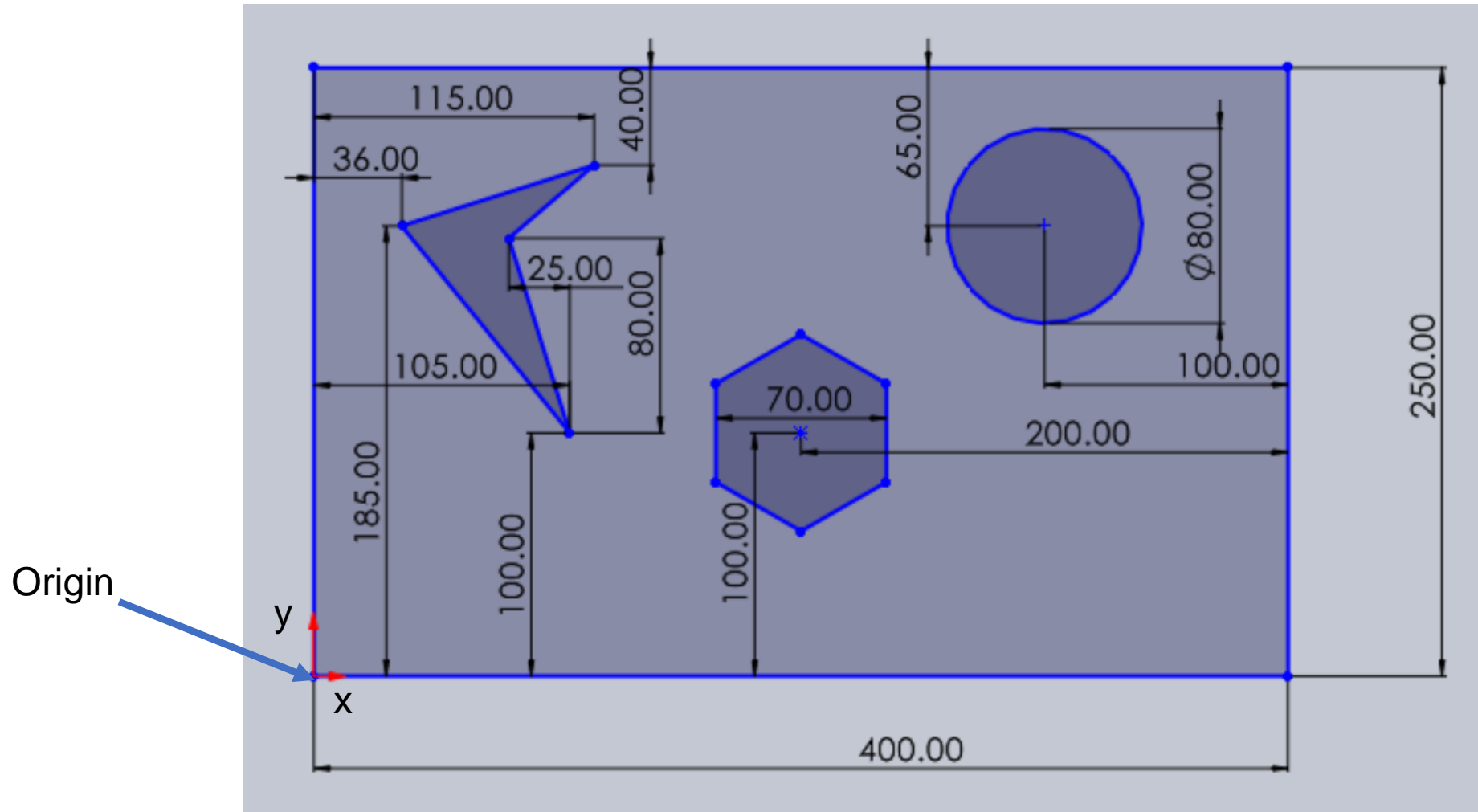
- Sample code for visualization is provided.

Visualization
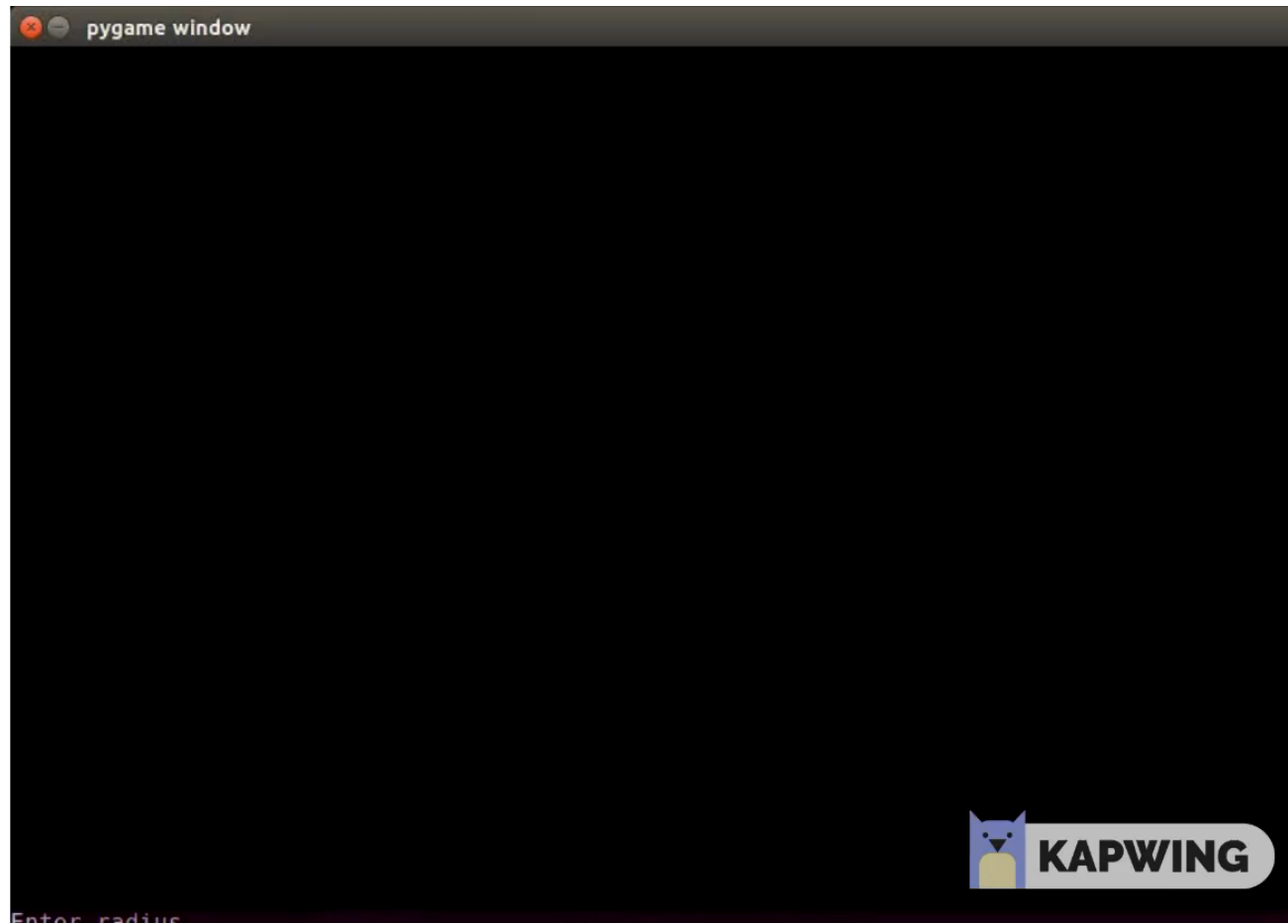
Step 6) Display the optimal path in the map
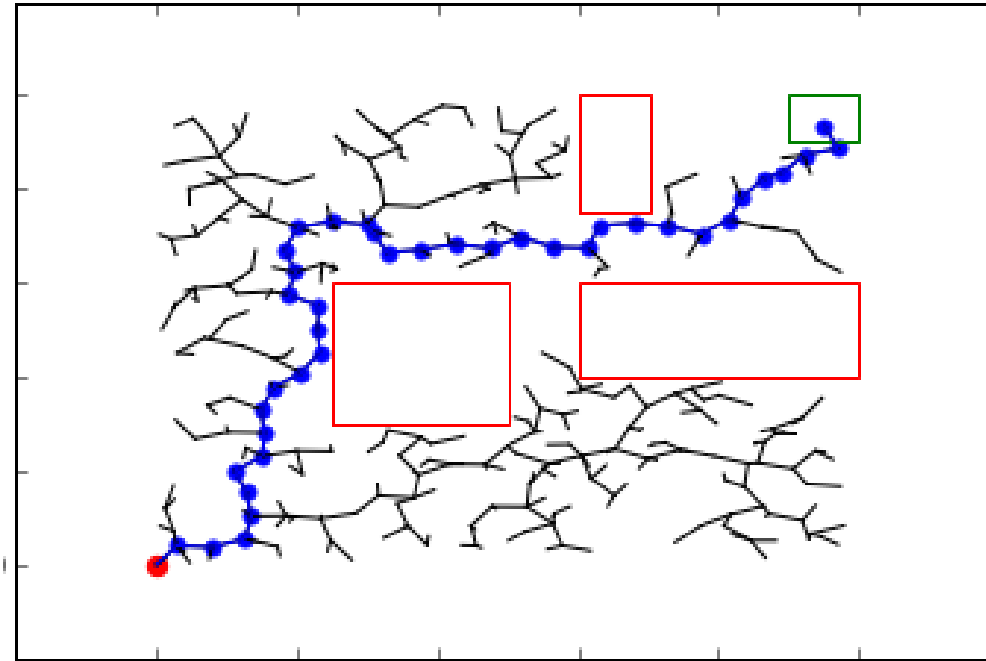
Final Map

Deliverables

Deliverables:

1. ReadMe.txt (Describing how to run the code in a txt format)

2. Source files
   - Student_name.py
   - GitHub repository link in the URL submission (private repository)

3. Output
   - Video recording (start and goal point can be random)

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