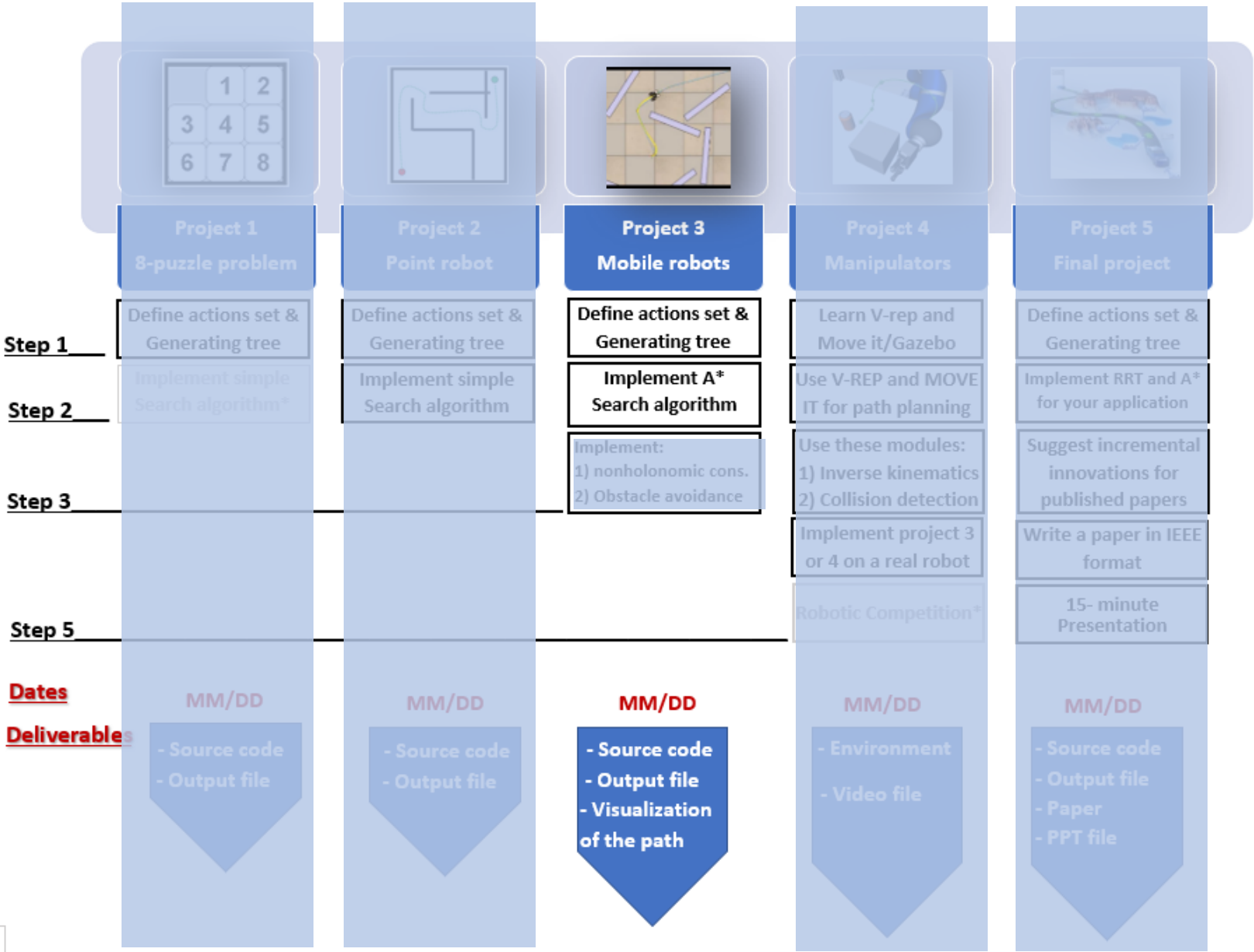
Project 3 Phase 2: Implementation of A* algorithm for a Rigid Robot

This is a group project, done in teams of 2. You can form your own groups

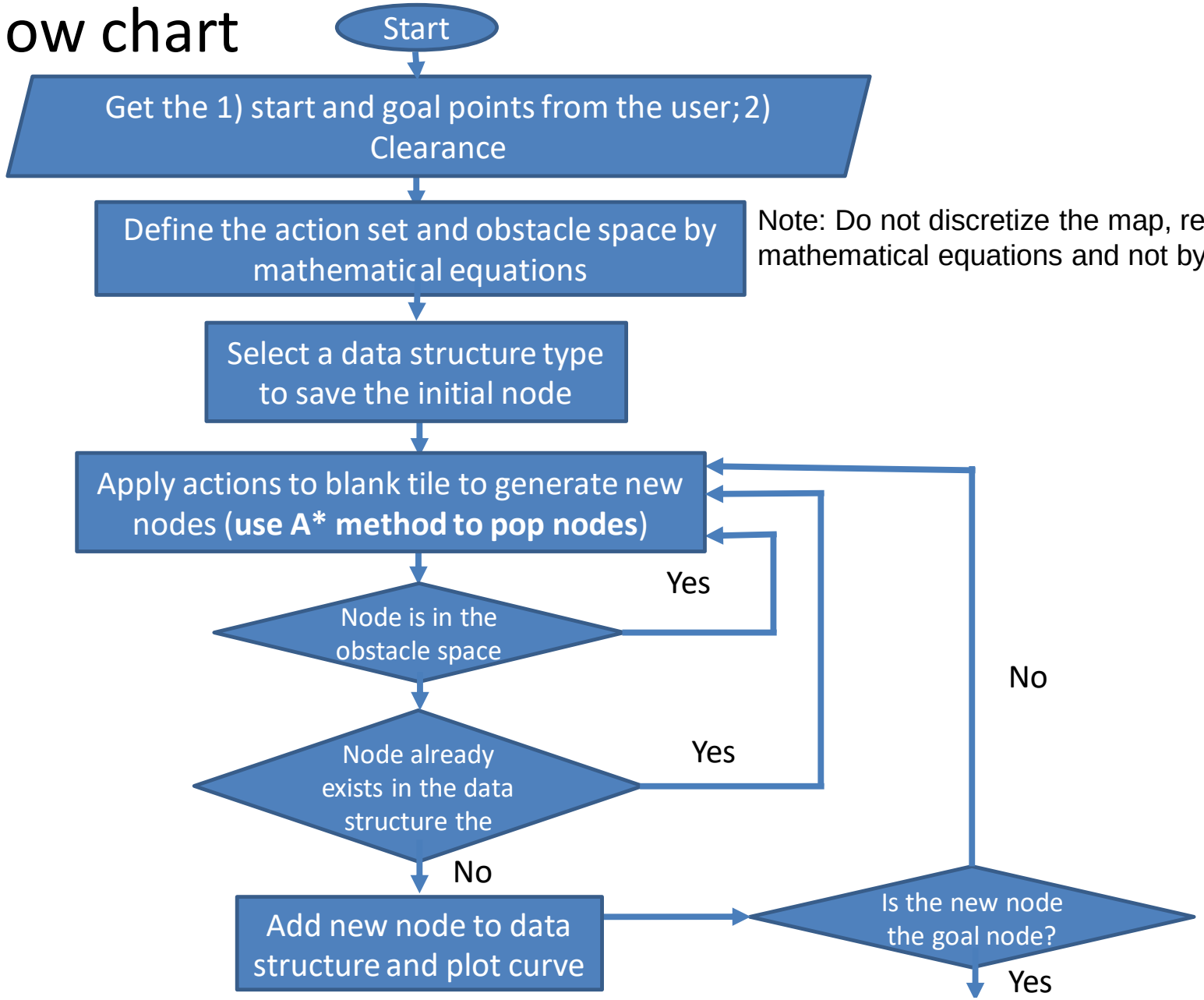
Due Date – March 20th, 11.59 PM

Project3

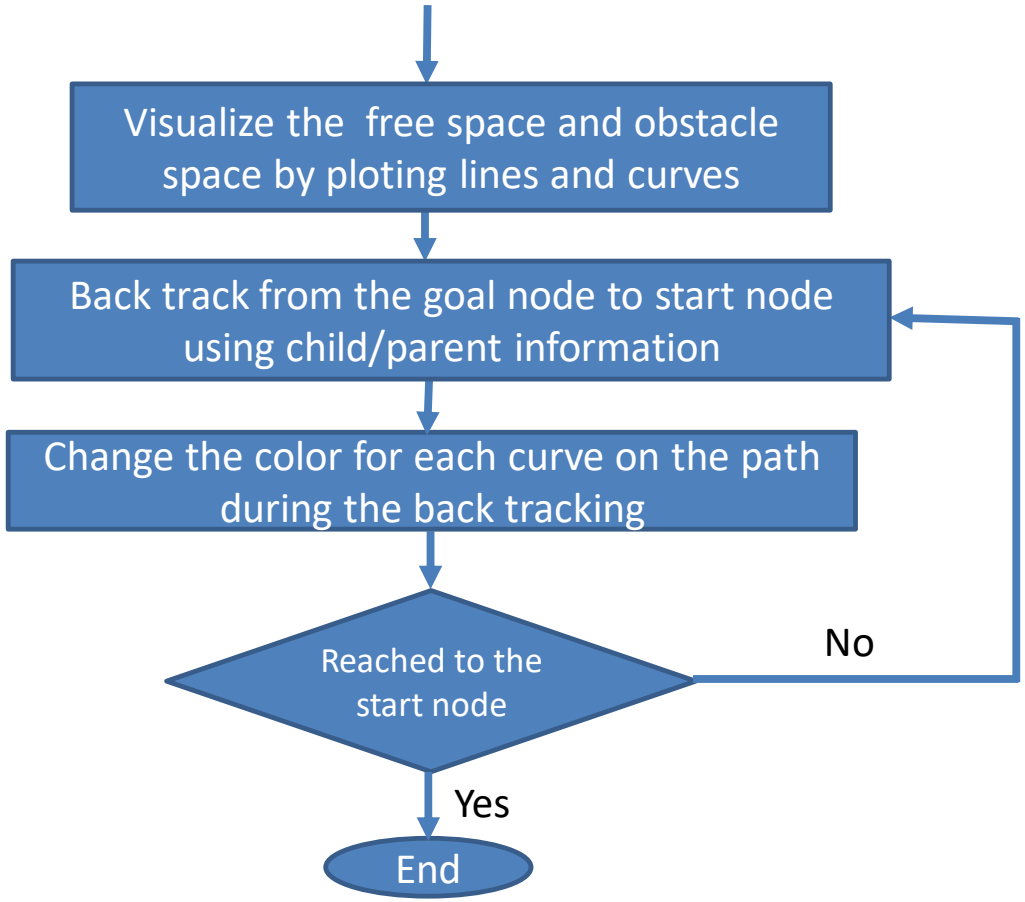


*Optional

Flow chart



Note: Do not discretize the map, represent the map by mathematical equations and not by a matrix


Visualize the free space and obstacle space by ploting lines and curves
Back track from the goal node to start node using child/parent information
Change the color for each curve on the path during the back tracking
Reached to the start node
No
Yes
End

Step 1: 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 <= d <=10)
5) Theta** (angle between consecutive actions)

*Cartesian Coordinates should be used

**Theta_g & Theta are optional. Mentioned in next slide.

Theta_g and Theta (optional input argument)

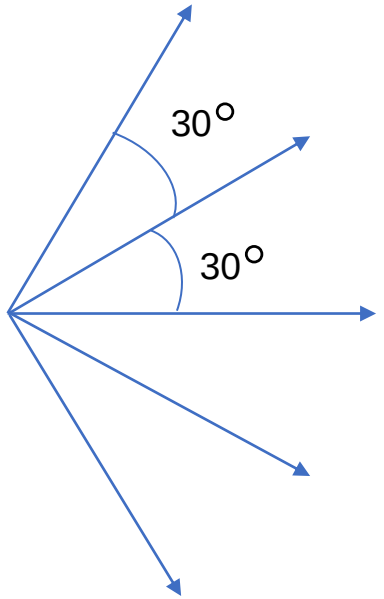
- Theta_g is the orientation at goal point of the robot.
- Theta is the angle between the action set at each node.
- To begin your approach, ignore Theta_g and keep the angle for Theta as 30 degrees. Once your solution is converging as required, experiment with these parameters to check for convergence.
- Finally, take these parameters as user inputs.

Note – This is an optional step. There are no extra points for them. By default, theta_g would not be present and theta would be 30 deg.

Step 2: Define Action Set

To check for duplicate nodes:

- Euclidean distance threshold is 0.5 unit (for x, y)
- Theta threshold is 30 degrees (for Θ)



Consecutive angles are 30 degrees.

Step 3: Define the map by algebraic equations

Note: Do NOT discretize the map by reducing its representation to a matrix representation

Step 4: 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 300/(threshold) by 200/(threshold) by 12 (i.e. 360/30) to store the visited regions information i.e. your matrix dimension will be (600x400x12).

Example:

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

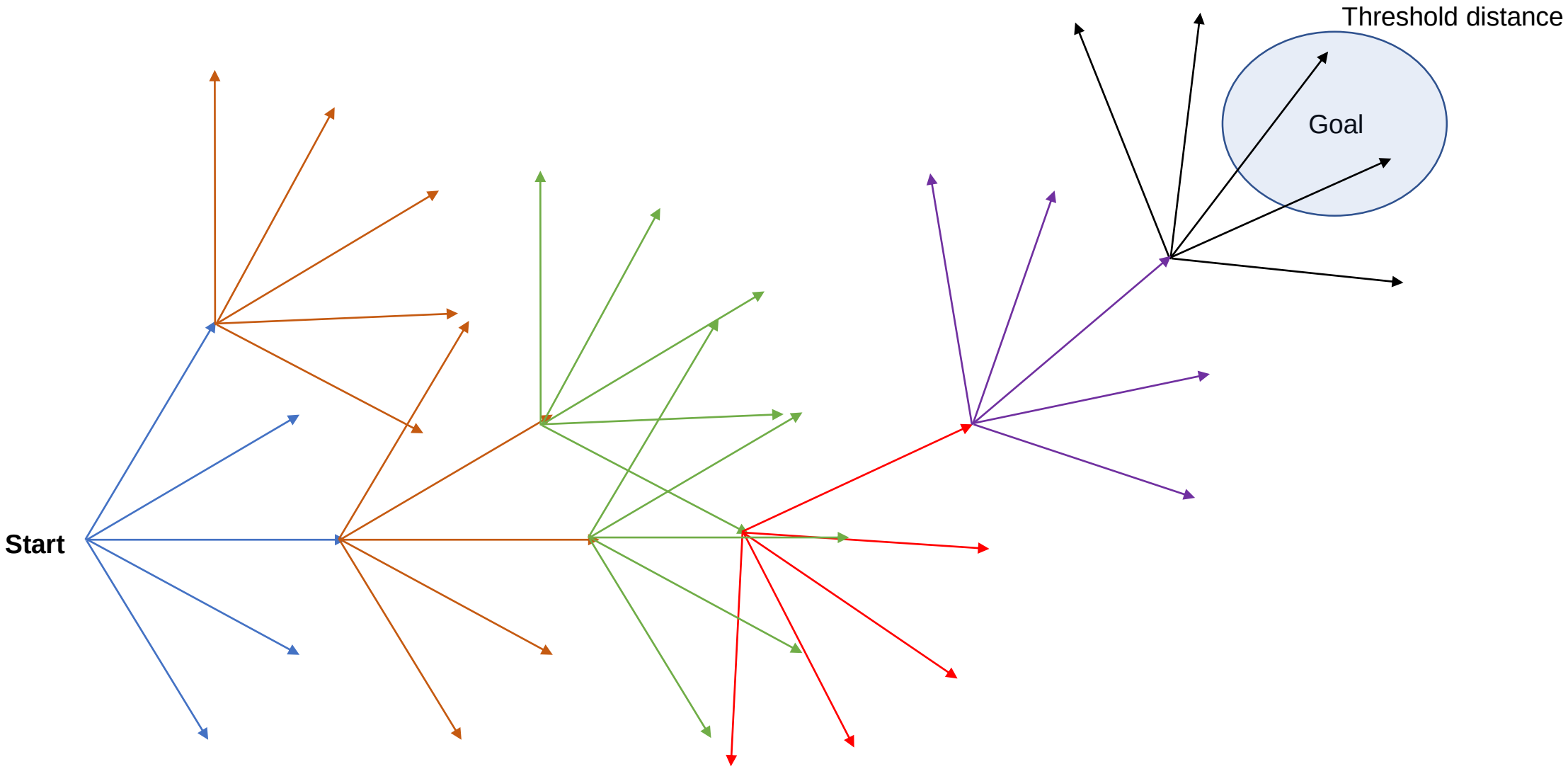
If node1= (3.2, 4.7, 0) visited → visited region: (3, 4.5, 0) →V[6][9][0]=1

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)

Graph



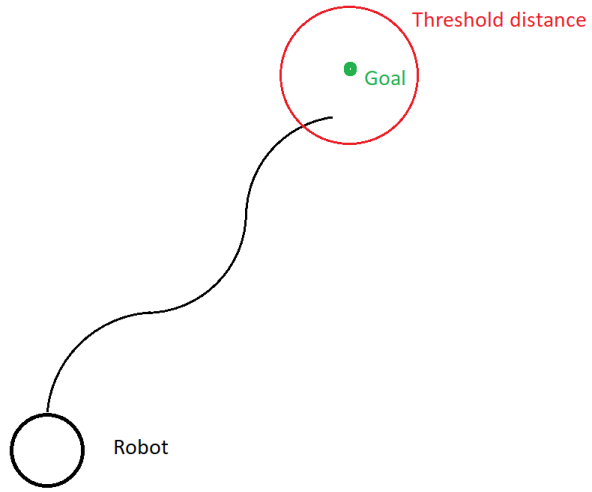
Step 5: Display the graph in the configuration space

- 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.

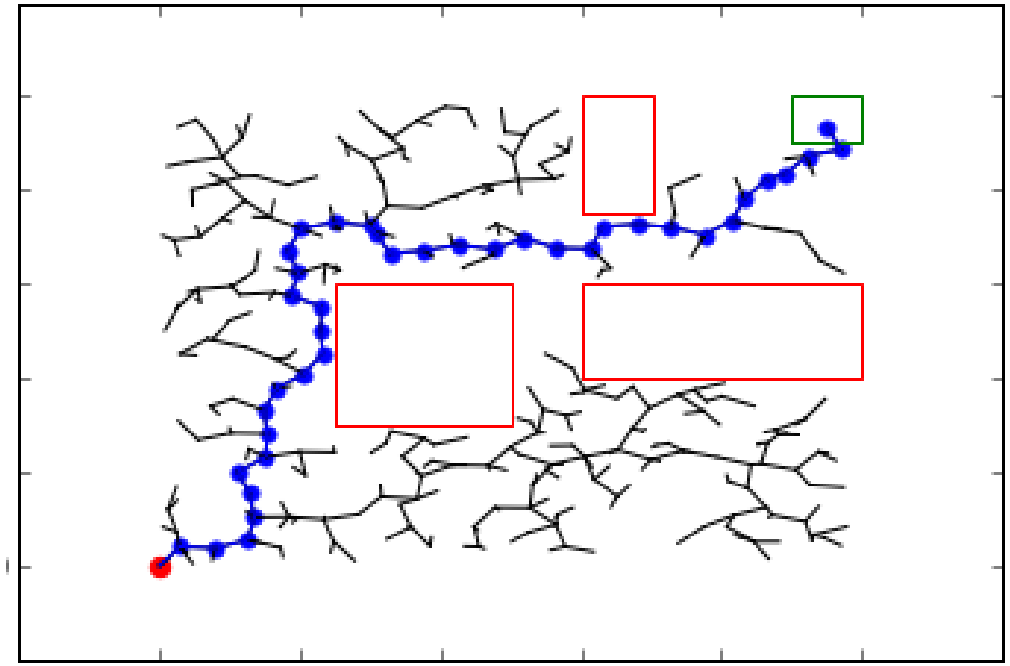
Step 6: implement A* search algorithm to search the tree and to find the optimal path

- 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 to terminate the program a threshold distance has to be defined.

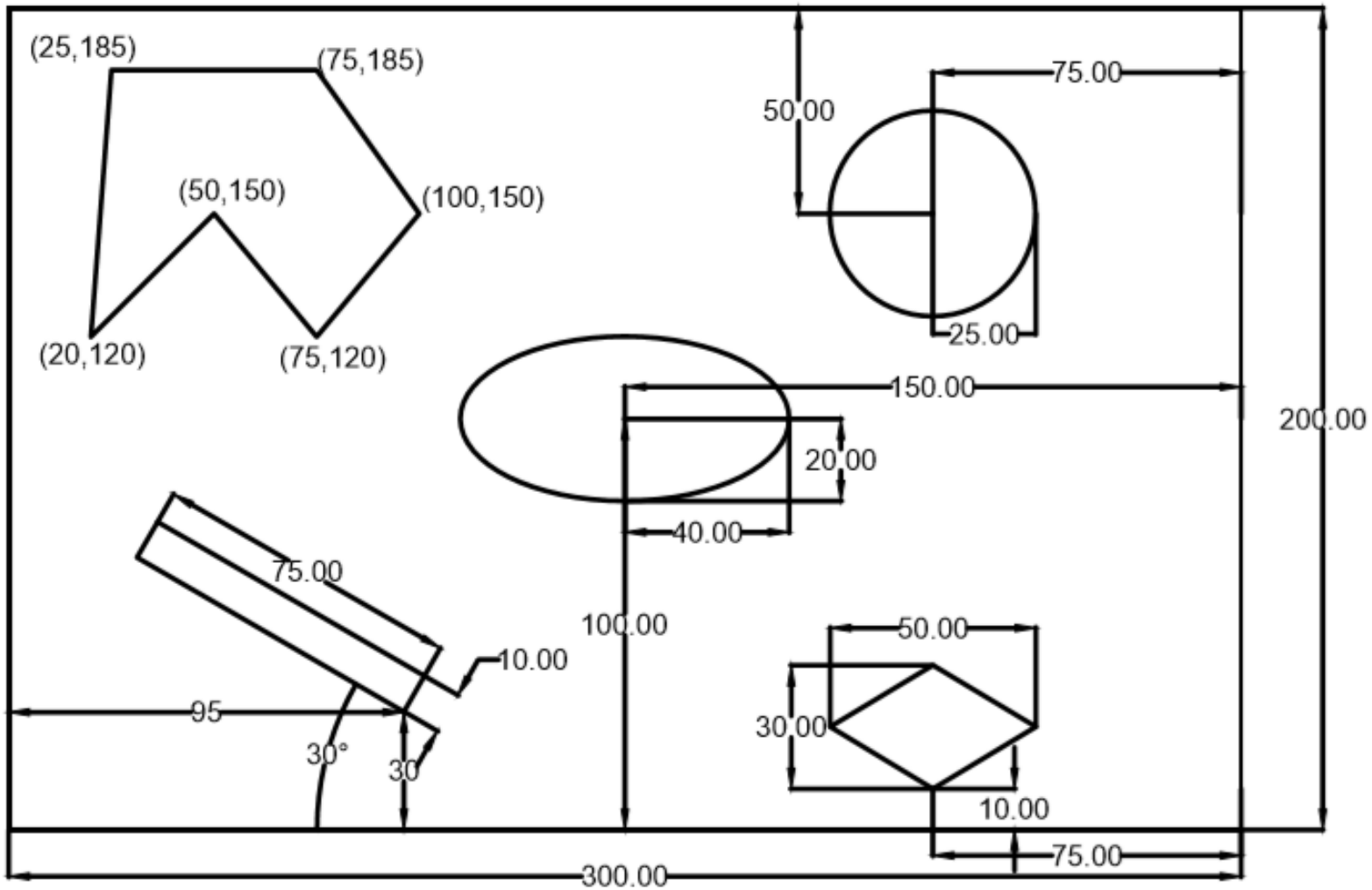
Goal threshold (1.5 units radius)



Step 7) Display the optimal path in the map



Final Map (Project 2, Project 3:Phase 2)



Deliverables

Deliverables:

1. Simulation results (video of the simulation)

Inputs –

Start : (50, 30, 60)

Goal : (150, 150)

Radius and clearance : 1 unit each

Step Size: 1 unit

Deliverables

Deliverables:

2. ReadMe
   - (Describing how to run the code in a txt format)
   - Dependencies and their installation
3. Source files for
   I. Astar_rigid.py

4. GitHub repository link in the URL submission box (One repository link with commits from both team members)

Submission Details

- You are required to submit a zip file with the file structure as shown

proj3p2_groupnumber_codingLanguage

— codes

— readme.txt

—Simulation video

— ***Please add your group to the excel sheet sent in announcement.***

Important pointers

- Only one person has to submit the final zip folder. Mention the other teammates in the comments section.
- Mention the GitHub Repository's link in the comments section as well.
- **User input - taking input from terminal only**