ENPM661 - Spring 2023

Project 03 - Phase 1

Implementation A* algorithm for a mobile Robot

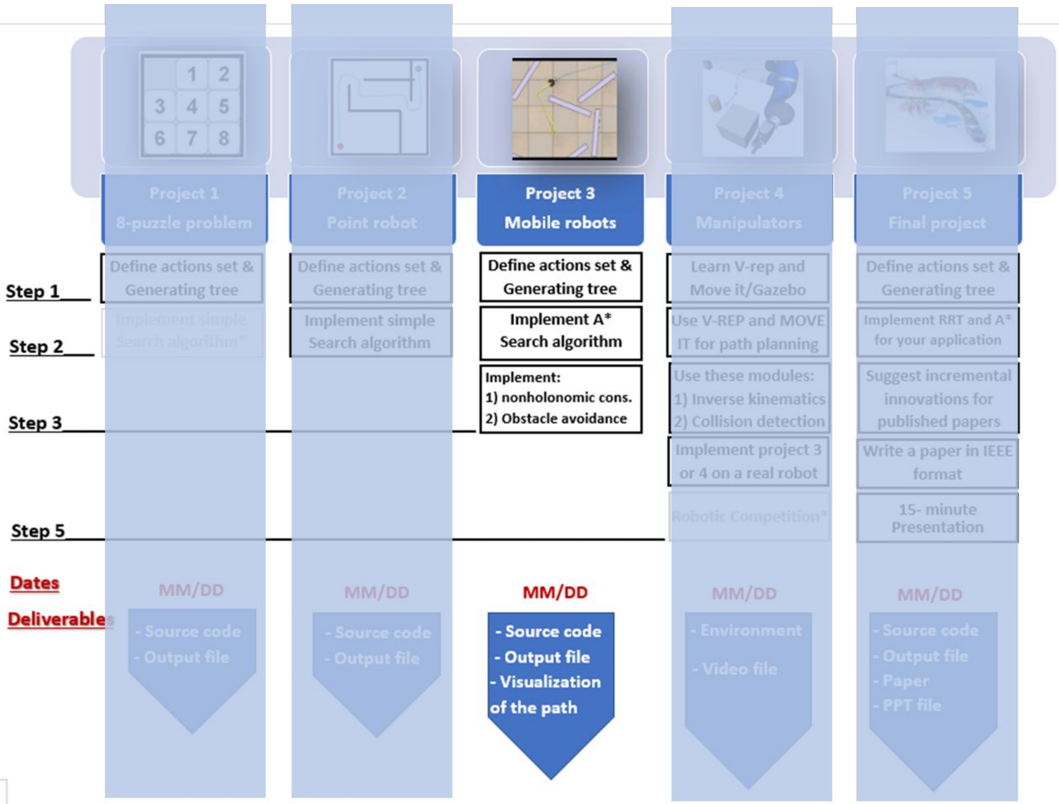
Note: This is a group project (max two students*)

*If you have received permission to team up as a group of 3, ignore this.

Due Date: March 19, 11:59 PM

Points/Weightage: 5

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 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

A* Algorithm: Pseudo Code

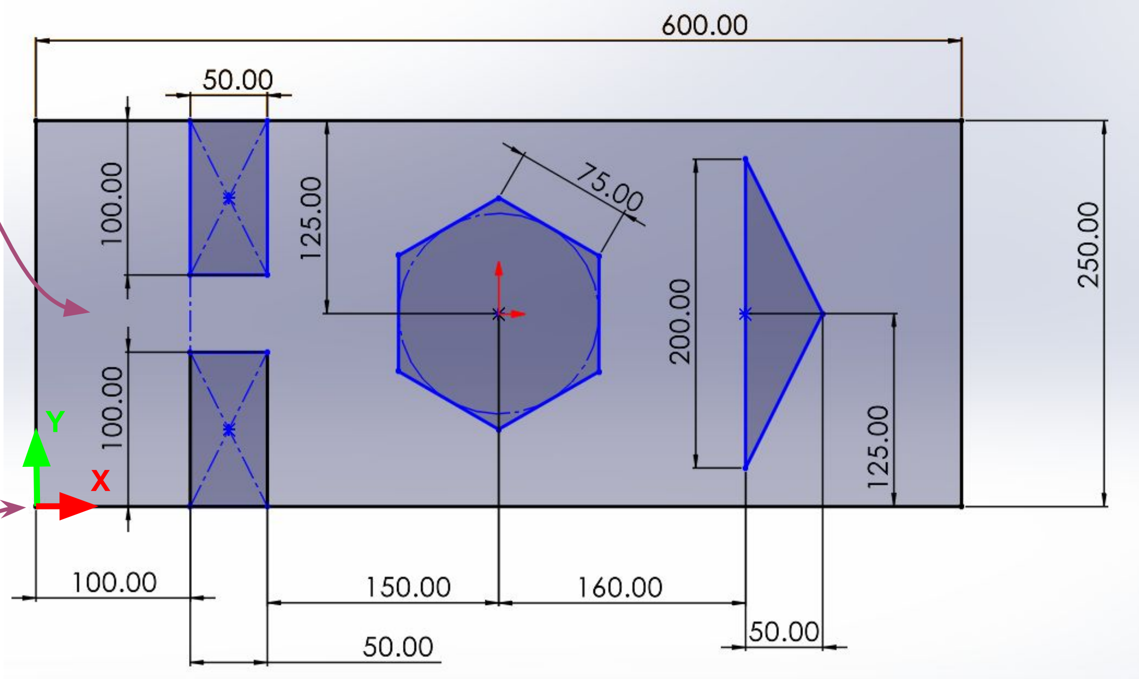
```
Create two empty lists named OpenList and ClosedList
Get the initial ($X_i$) and goal node ($X_g$) from the user
OpenList.put($X_i$)
While (OpenList not EMPTY) and (Not reached the goal) do
    x ← OpenList.get()
    Add x to ClosedList
    if x = $X_g$
        Run backtrack function
        return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

Project 03: 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 map.
- Implement A* Algorithm to find a path between the start and end point on a given map for a mobile robot (**radius = 5mm; 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]

Project 03: Map

Free space



Origin

Start and Goal coordinates are defined w.r.t. this origin

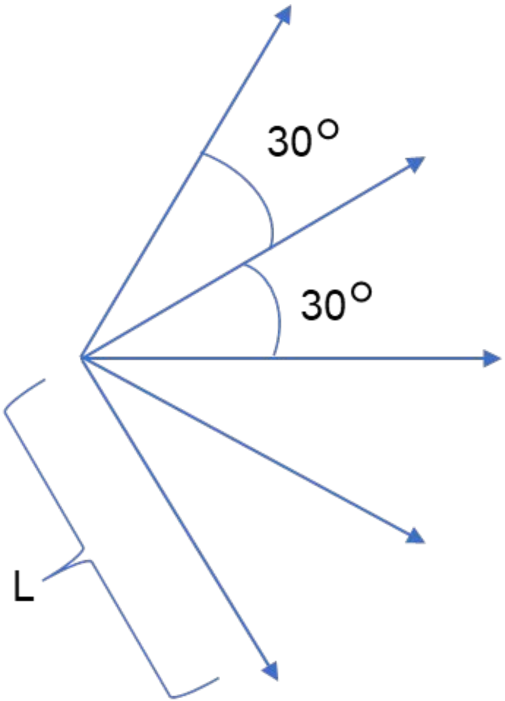
- The above map represents the space for clearance = 0 mm. For a clearance of 5 mm, the obstacles (including the walls) should be bloated by 5 mm distance on each side.

Project 03: Action set

To check for duplicate nodes:

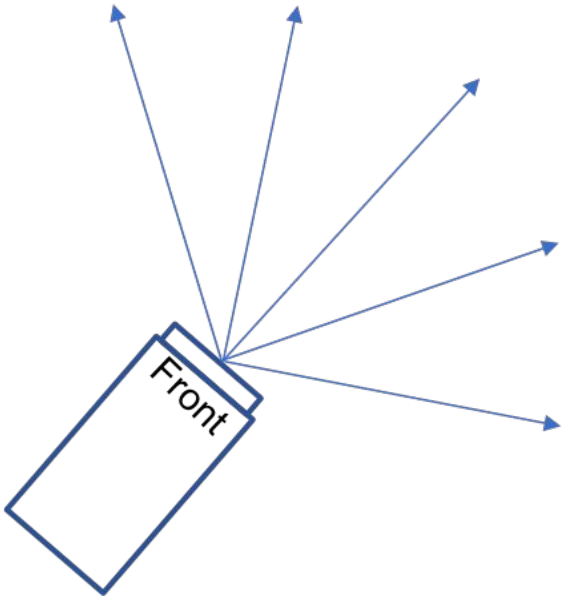
- Euclidean distance threshold is 0.5 unit (for x, y)
- Theta threshold is 30 degrees (for ϴ)

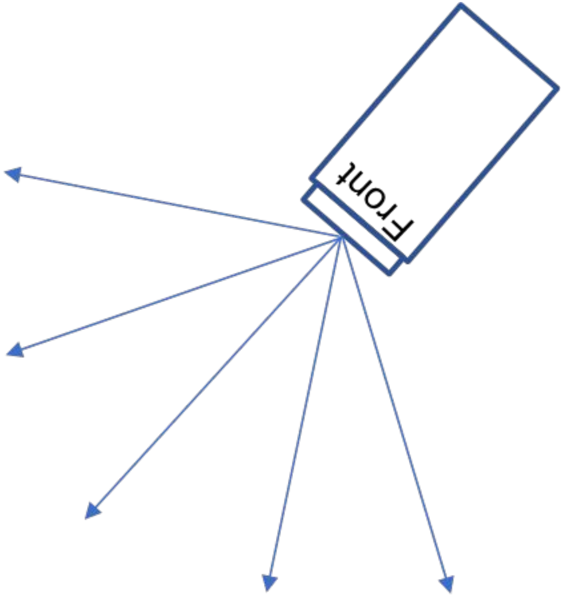
Python file provided can be used to plot



Consecutive angles are 30 degrees.

Project 03: Action set





Project 03: 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/

Project 03: User input

- Get the following values from the user
  - Start Point Coordinates (3-element vector): (X_s, Y_s, Θ_s).
    - Θ_s - The orientation of the Robot at the start point.
  - Goal Point Coordinates (3-element vector): (X_g, Y_g, Θ_g)
    - Θ_g - The orientation of the Robot at the goal point.
  - Clearance and robot radius
  - Step size of the robot in units. ($1 <= L <= 10$)

*Cartesian Coordinates should be used.

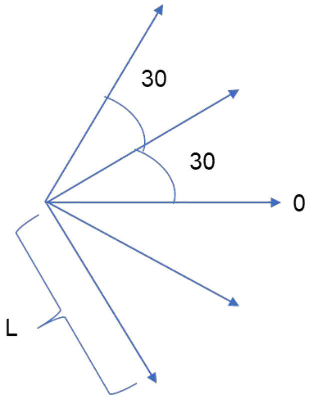
**Θ_s and Θ_g are in degrees. Take these parameters as user inputs (k * 30), i.e. {.., -60°,-30°, 0°, 30°, 60°, ..}

Step 01: Define the Actions in a Mathematical Format

- You can use the same data structure from Project 02 to store the node information.
    - Refer Priority Queues/Heap Queues
    - Refer Updating a node information using different data structures.ipynb

- Write 5 functions, one for each action. The output of each function is the state of a new node after taking the associated action.

- The step size is the length of the vectors.



Step 02: Find the Mathematical Representation of Free space

- Use **Half planes and semi-algebraic models** to represent the obstacles in the map.
    - Refer Session03. Basic- 2023.pptx
    - The equations must account for the 5 mm clearance of the robot.

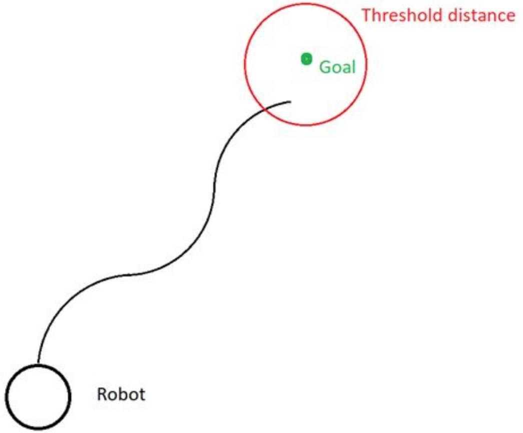
- Using OpenCV/Matplotlib/Pygame/Tkinter (or any other graphical plotting library of your choice), create an empty canvas of size height=250 and width=600.
    - With the above equations, assign a different color for all the pixels within obstacles and walls. You may choose to represent the clearance 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]

Retain this part from the previous project

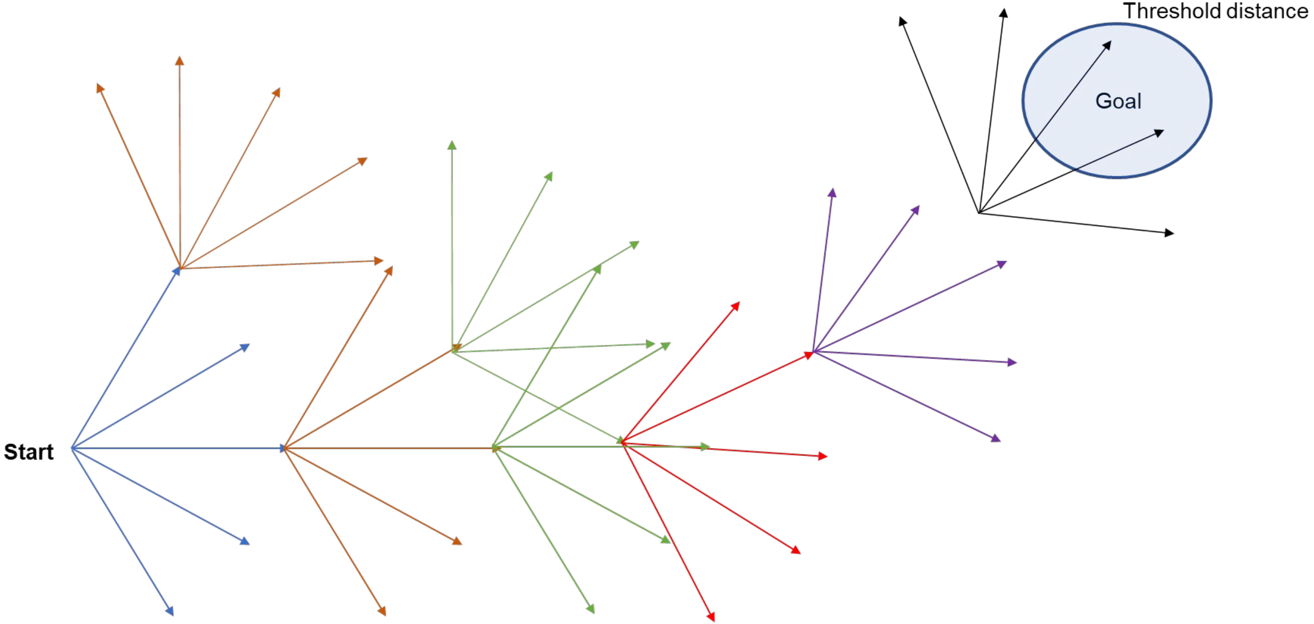
Step 03: Implement A* search algorithm to search the tree and to find the optimal path

- Forward search using A* algorithm.
- Consider Euclidean distance as a heuristic function.
- Note:- Define a reasonable threshold value for the distance to the goal point. Do to the limited number of moves the robot cannot reach the exact goal location. So, use a threshold distance to check the goal.
    - Goal threshold(1.5 units radius)



Step 03: Implement A* search algorithm to search the tree and to find the optimal path

- A sample graph, just for visualization. The graph generated may/maynot look similar to the example below.



Step 03: Implement A* search algorithm to search the tree and to find the optimal path

- To generate the graph consider the configuration space as a 3 dimensional space.
- There are two method to find the duplicate node:
  - **First method** - 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** - 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) x 600/(threshold) x 12 (i.e. 360/30) to store the visited regions information i.e. your matrix dimension will be (500x1200x12).
    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 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).

Retain this part from the previous project

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.

Retain this part from the previous project

Project 03: Deliverables

- **Proj3_firstname#1_firstname#2_firstname#3.zip.** This zip file should contain the following files.
  - 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)
    - Must mention all Team Member names along with their respective directory ID and UID.
  - Source Code (.py)
    - Filename: **a_star_firstname#1_firstname#2_firstname#3.py**
    - Code should accept start and goal coordinates as a user input
      - Values should be w.r.t the map origin frame
      - Θ_s and Θ_g should be in degrees
    - 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
- **Proj3_firstname#1_firstname#2_firstname#3.pdf.** This file should contain the source code for plagiarism check. ***This file needs to be submitted ALONG with the ZIP file, and NOT INSIDE the ZIP file***.

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 250x600 nodes, optimize your code whenever possible.
- Use the time library to print the runtime of your algorithm.
- Since the canvas size is 600x250, 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).