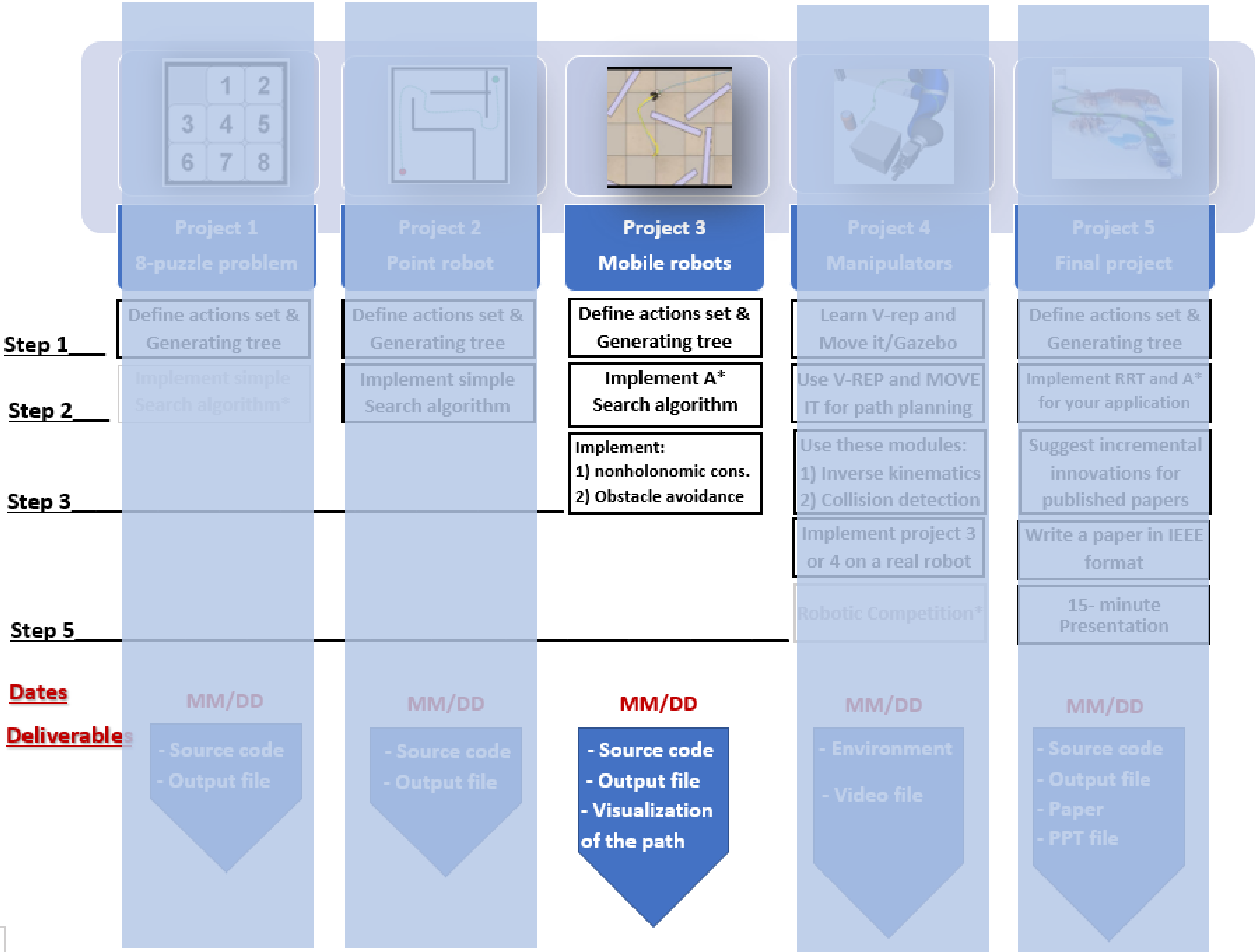
Project 3-Phase 3

Implementation of A* algorithm on a differential drive (non-holonomic) TurtleBot robot
(In groups of two)

Deadline – April 3, 11.59PM

Project3

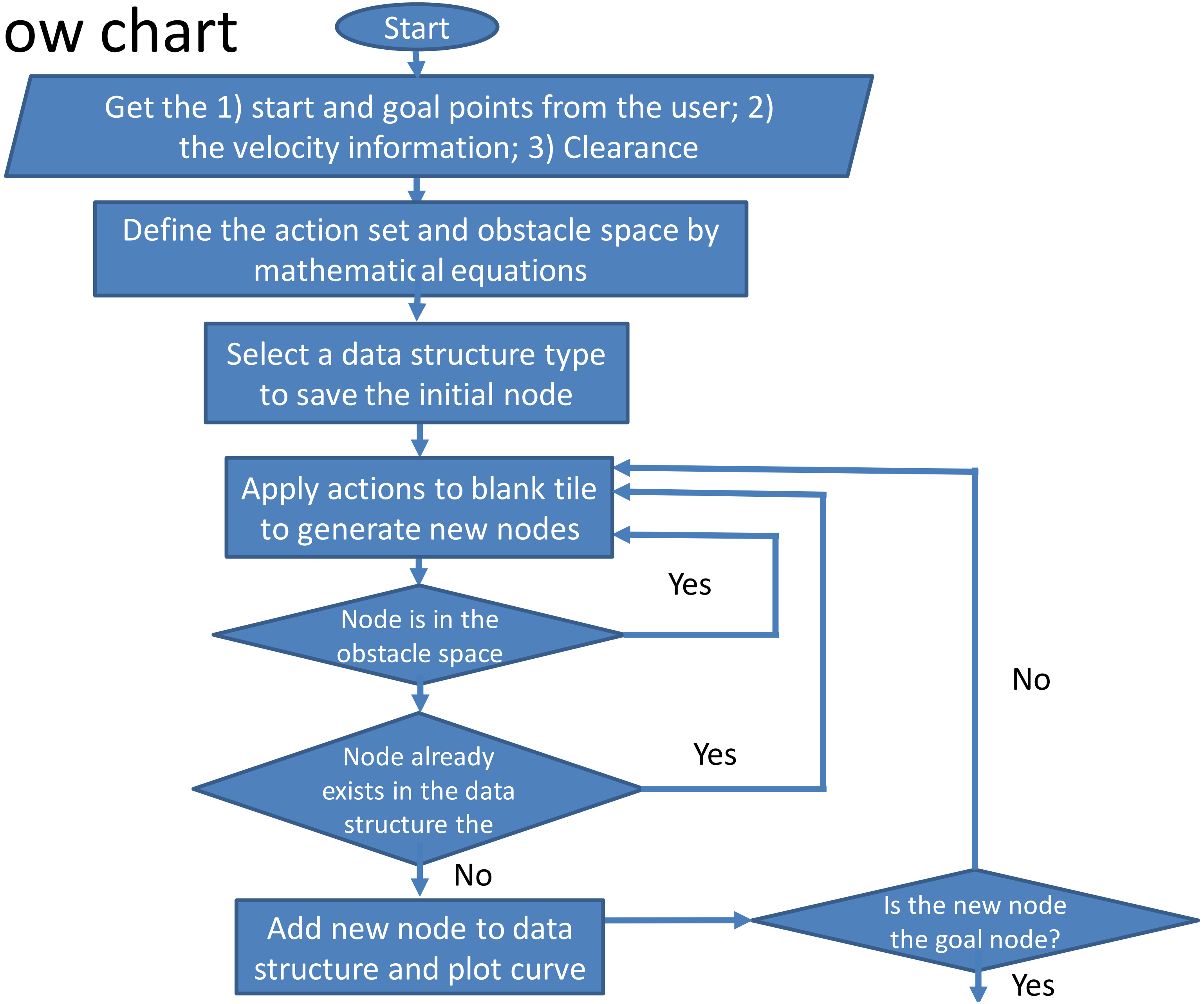


*Optional

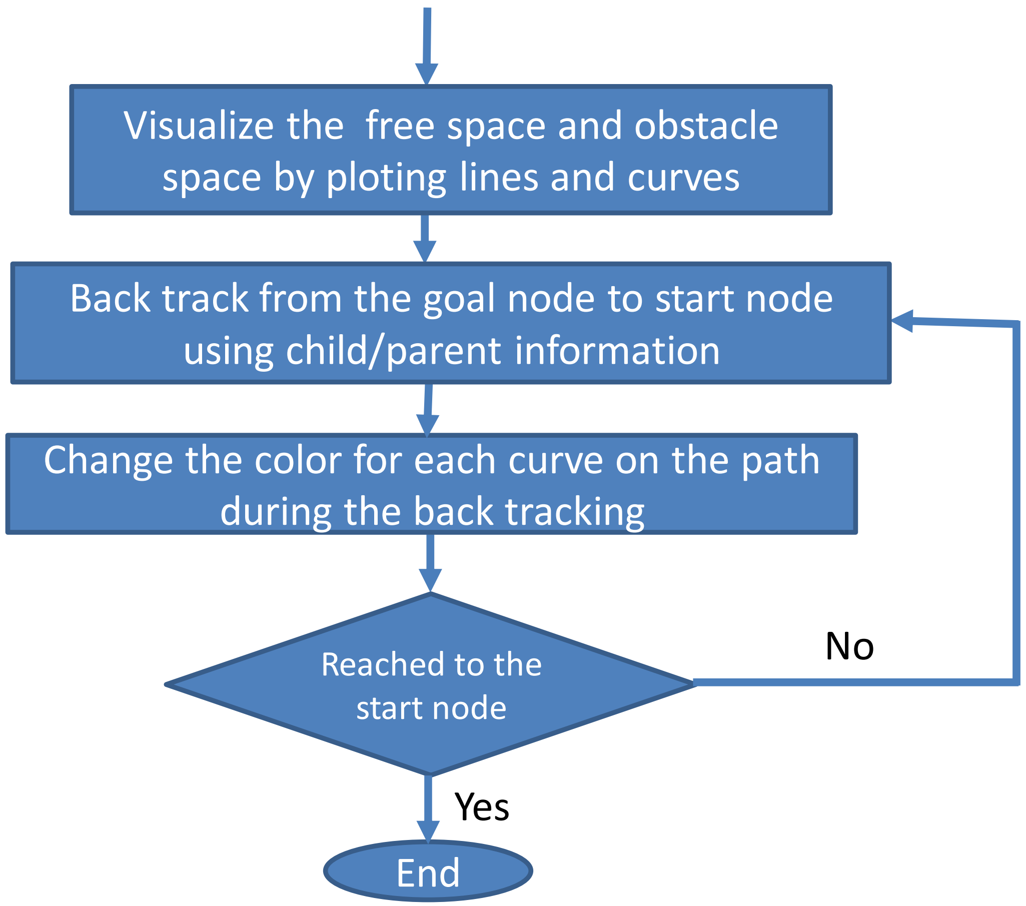
Project description

- Navigate a differential drive robot (TurtleBot 2 / TurtleBot 3) in a given map environment from a given start point to a given goal point.
- Consider differential drive constraints while implementing the A* algorithm, with 8-connected action space .

Flow chart



Start
Get the 1) start and goal points from the user; 2) the velocity information; 3) Clearance
Define the action set and obstacle space by mathematical equations
Select a data structure type to save the initial node
Apply actions to blank tile to generate new nodes
Node is in the obstacle space
Yes
Node already exists in the data structure the
Yes
No
Add new node to data structure and plot curve
Is the new node the goal node?
No
Yes


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

Inputs from the User

- Your code must take following values from the user:

1) Start Point Co-ordinates (3-element vector – x, y, theta)
2) Goal Point Co-ordinates (2-element vector – x, y)
3) Wheel RPMs (2-element vector) => Two possible values for the wheel RPMs
4) Clearance

Parameters to be Defined

- Your code must take the following parameters into consideration:

1) Robot Diameter (from the datasheet)
2) Wheel Distance –L (to be computed using the datasheet)
3) Reasonable Clearance

Note that, these parameters are not defined by the user. These are the parameters you need to consider while developing the code.

Project description (Continued..)

- Let the two RPMs provided by the user are RPM1 and RPM2 (shown in next slides). Then the action space for the A* algorithm are:

1. [0, RPM1]
2. [RPM1, 0]
3. [RPM1, RPM1]
4. [0, RPM2]
5. [RPM2, 0]
6. [RPM2, RPM2]
7. [RPM1, RPM2]
8. [RPM2, RPM1]

- Here the first element corresponds to the left wheel RPM and the second element corresponds to the right wheel RPM.

Step 1) Write a subfunction for non-holonomic constraints

- This subfunction will take two arguments (Rotational velocity of the left wheel and right wheel) and return the new coordinate of the robot, i.e. (x,y, theta). Where x and y are the translational coordinate of the robot and theta shows the orientation of the robot with respect to axis x.

- A sample is provided in the python file. You may choose to modify the function or implement your own.

Differential Drive Constraints

- For this project you consider the robot as a non-holonomic robot, which means the robot cannot move in y-direction independently.
- You will have to define smooth moves for the robot by providing left and right wheel velocities. The time for each move will have to be fixed.
- The equations for differential drive robot

$$\dot{x} = \frac{r}{2}(u_l + u_r)\cos\theta$$
$$\dot{y} = \frac{r}{2}(u_l + u_r)\sin\theta$$
$$\dot{\theta} = \frac{r}{L}(u_r - u_l)$$

Differential Drive Constraints (Continued..)

Where, $\dot{x}$ and $\dot{y}$ are the velocities in x and y directions

u_l and u_r are left and right wheel velocities

r is a wheel radius and L is the distance between two wheels

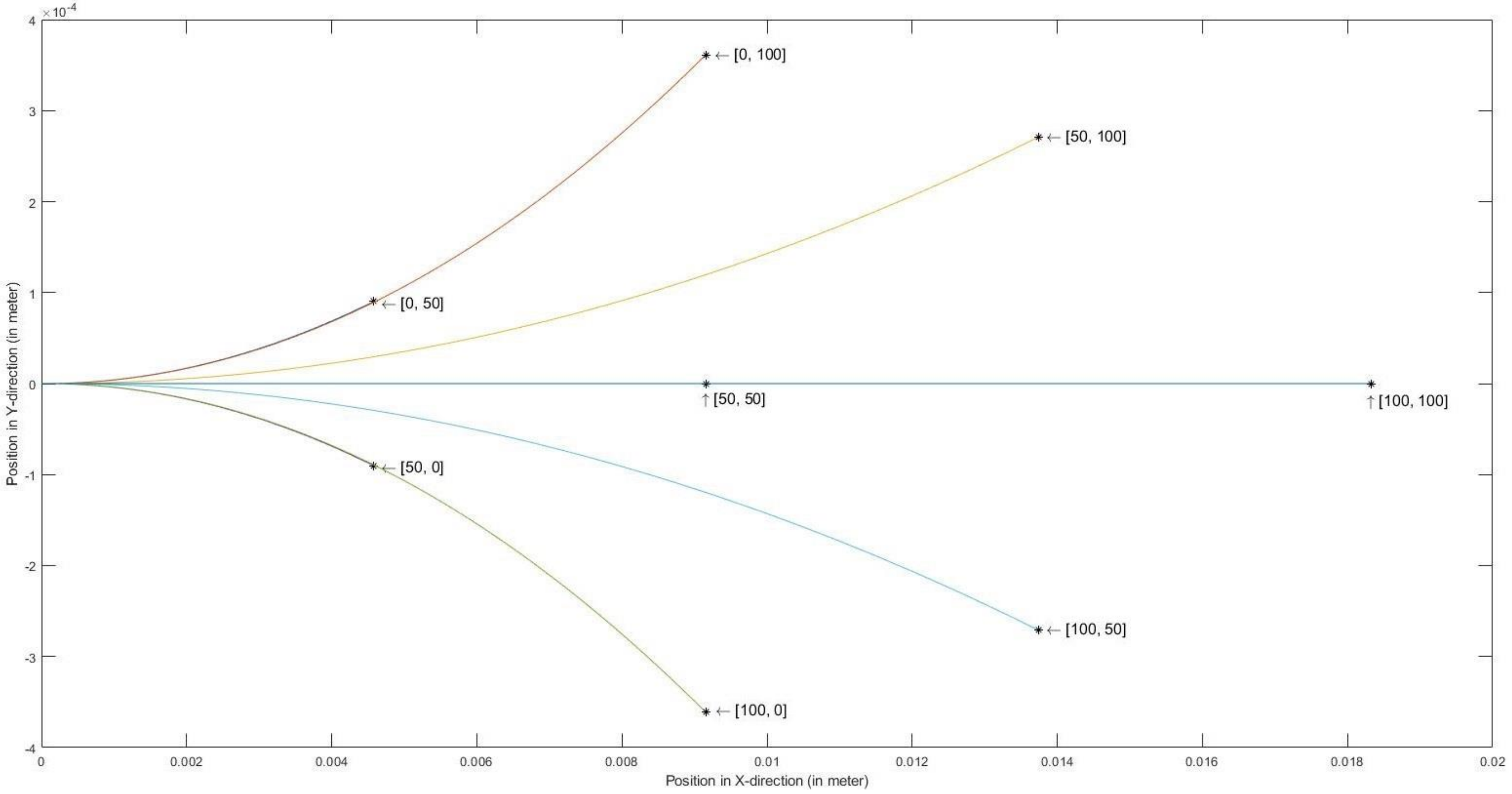
- From the velocity equations we can calculate the distance travelled and angle covered in each time step

$$dx = \frac{r}{2}(u_l + \ u_r)\cos\theta \ dt$$

$$dy = \frac{r}{2}(u_l + \ u_r)\sin\theta \, dt$$

$$d\theta = \frac{r}{L}(u_r - \ u_l)\, dt$$

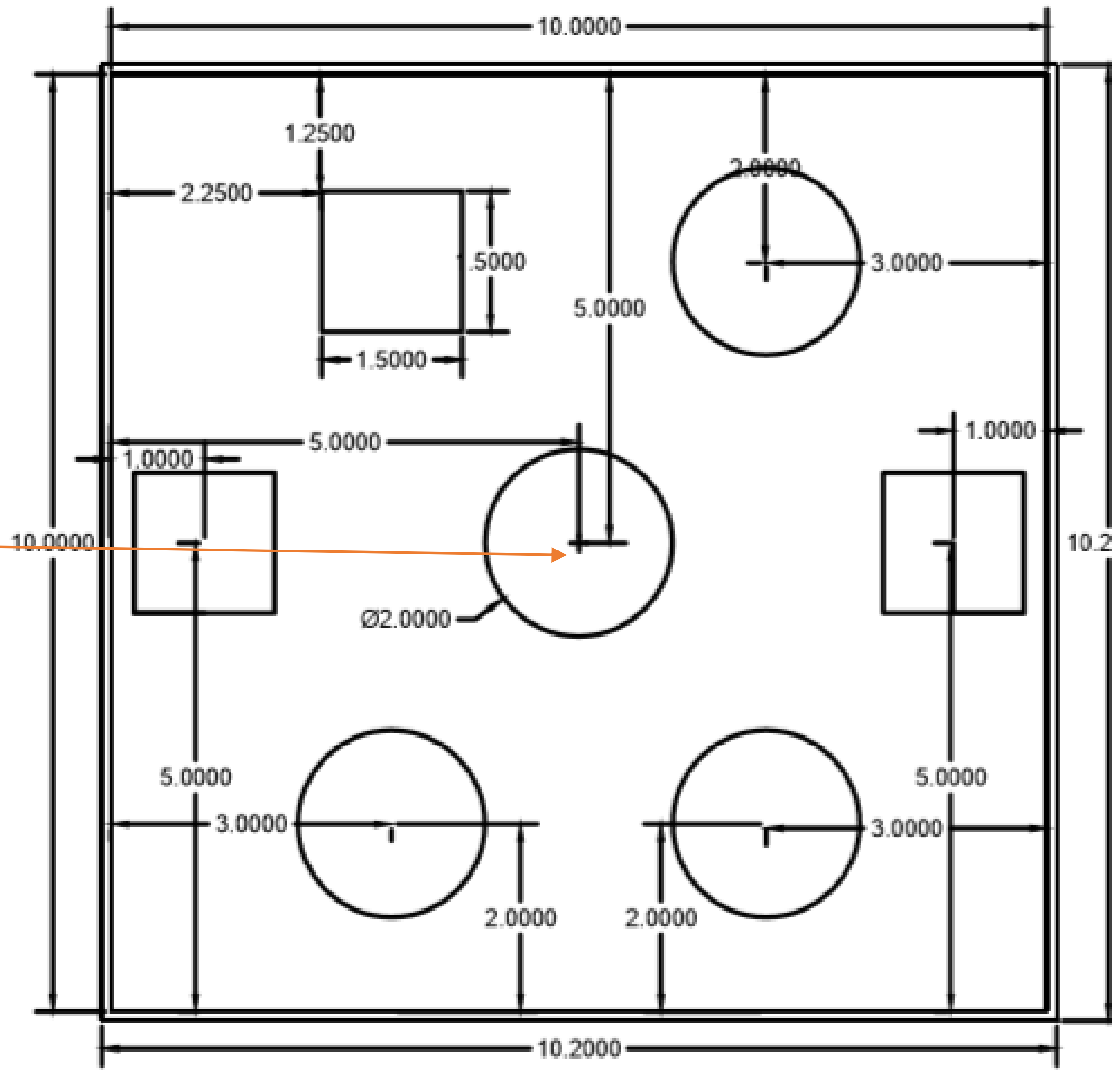
Differential Drive Constraints (Continued..)



- The figure shows various curvatures obtained by changing left and right wheel velocities.

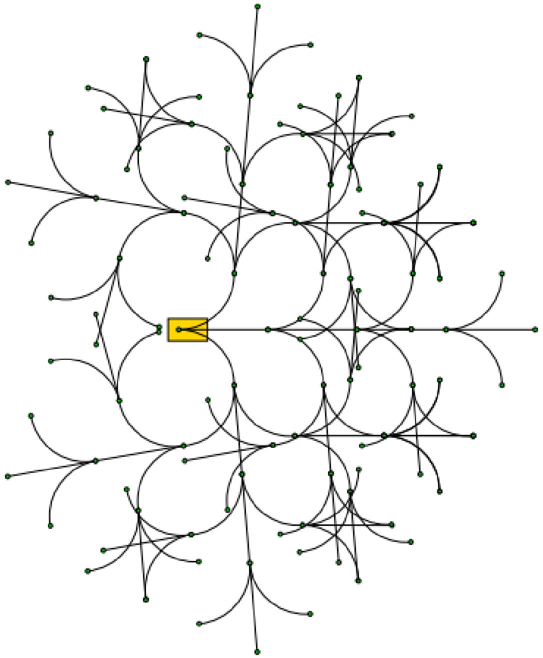
Step 2) Modify the map to consider the geometry of the rigid robot

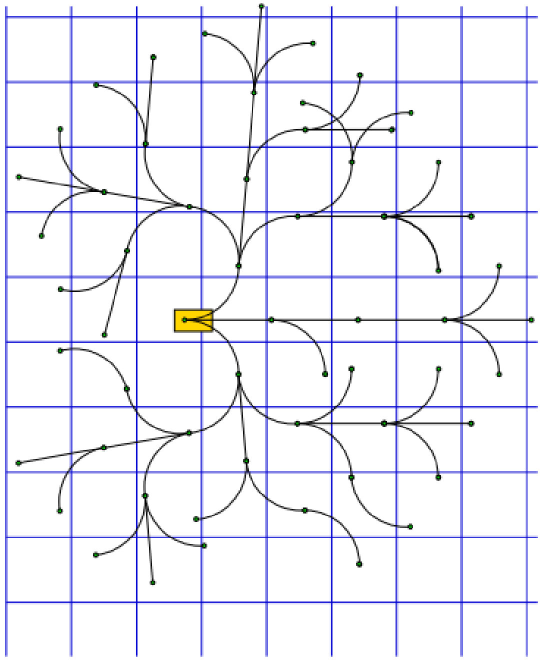
- Dimensions of the robot is available in the datasheet
- The center circle is (0,0).



Step 3) Generate the tree using non-holonomic constraints

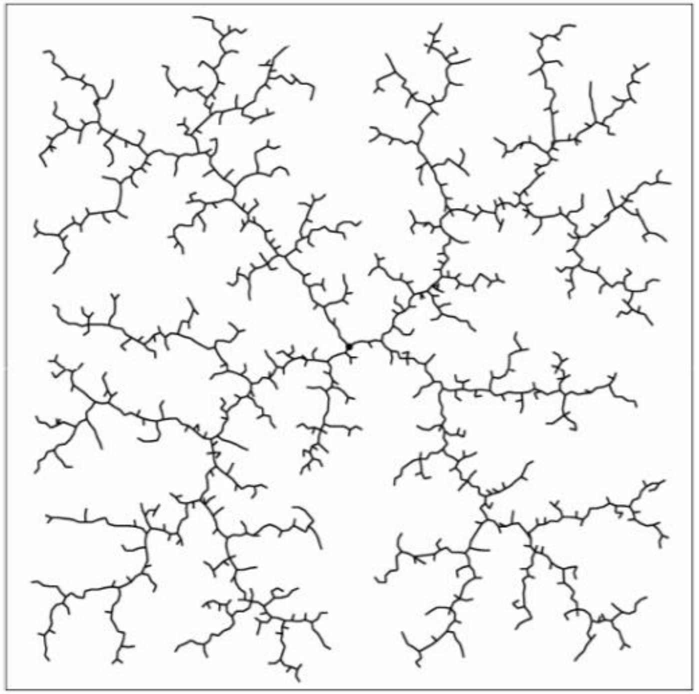
- Consider the configuration space as a 3 dimensional space.
- Follow the same step from Project 3- Phase 2 , step 4 to check for duplicate nodes (consider threshold as per your own need, but make sure the 8-action space is not violated).





Step 4) Display the tree in the configuration space

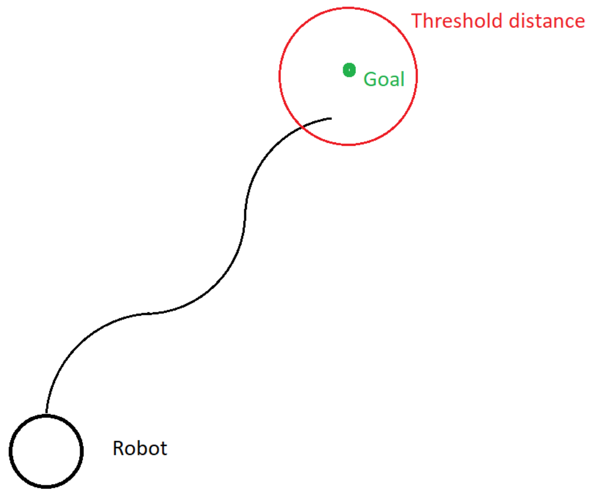
- Use curve that address the non-holonomic constraints to connect the new node to previous nodes and display it on the Map.



RI 16-735, Howie Choset with slides from James Kuffner

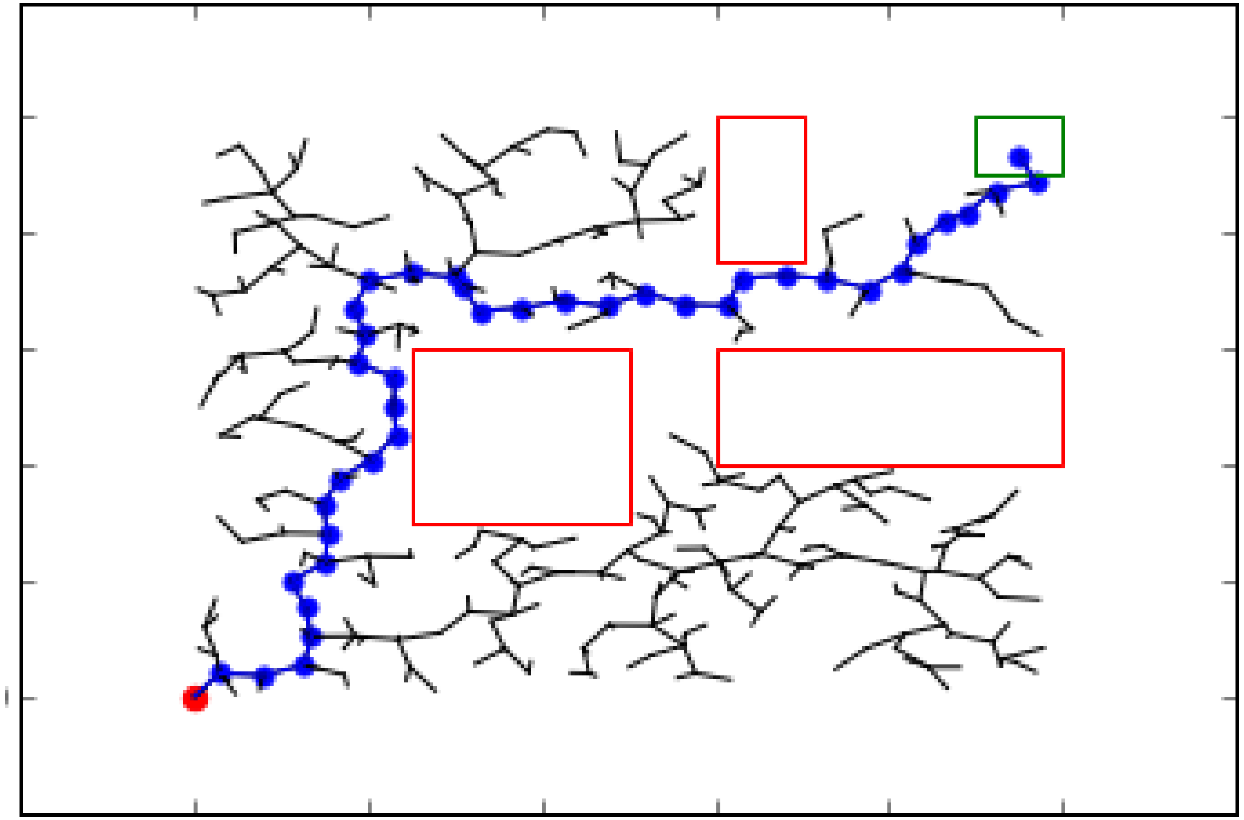
Step 5) implement A* search algorithm to search the tree and to find the optimal path

- 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.



Step 6) Display the optimal path in the map

To plot the final path, you can use ‘**quiver**’ function as shown in **plot_curve** in the shared python file (plot.py).



Submission Details

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

proj3_groupnumber_simulationSoftware

— code

— simulation video

— readme.txt

— GitHub Link

Video Submission – one video

- Give a start point (bottom left) and end point (top right) such that almost all the nodes on the map are being explored.
- Decide the clearance and wheel RPM’s on your own.
- Mention all the user inputs in the ReadMe.