






# Project 3-Phase 3

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

Deadline – April 25th, 11.59PM

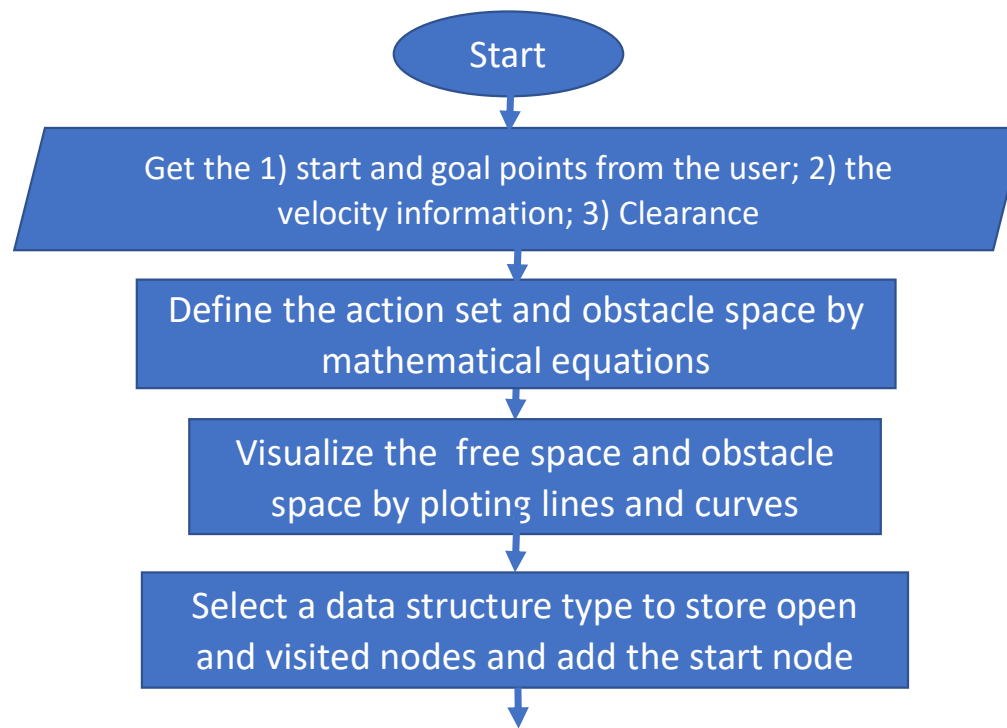
# Project3

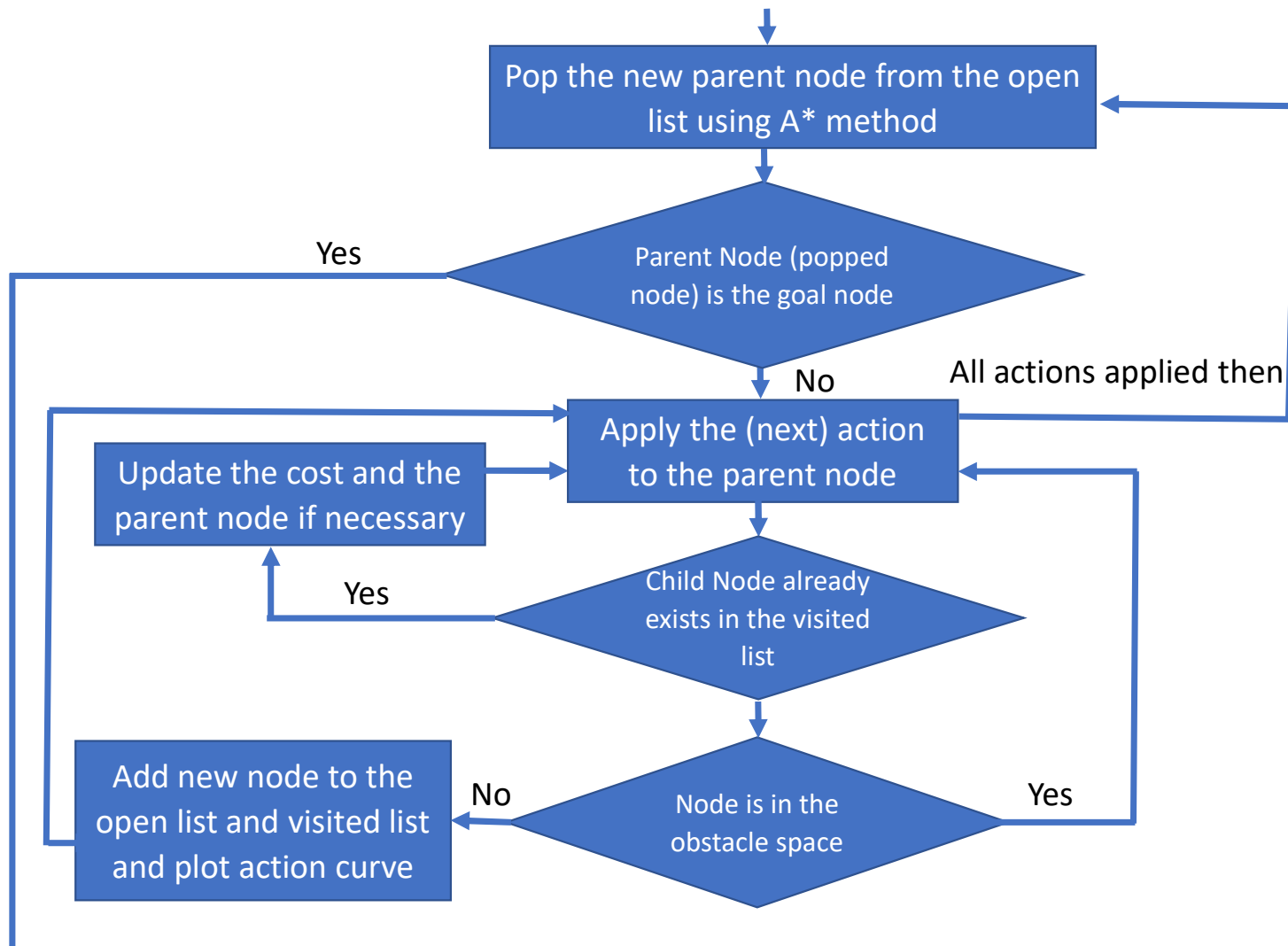
					
	Project 1 8-puzzle problem	Project 2 Point robot	Project 3 Mobile robots	Project 4 Manipulators	Project 5 Final project
<b>Step 1</b>	Define actions set & Generating tree	Define actions set & Generating tree	Define actions set & Generating tree	Learn V-rep and Move it/Gazebo	Define actions set & Generating tree
<b>Step 2</b>	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
<b>Step 3</b>			Implement: 1) nonholonomic cons. 2) Obstacle avoidance	Use these modules: 1) Inverse kinematics 2) Collision detection	Suggest incremental innovations for published papers
<b>Step 5</b>				Implement project 3 or 4 on a real robot	Write a paper in IEEE format
<b>Dates</b>	MM/DD	MM/DD	MM/DD	MM/DD	MM/DD
<b>Deliverable</b>	- 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

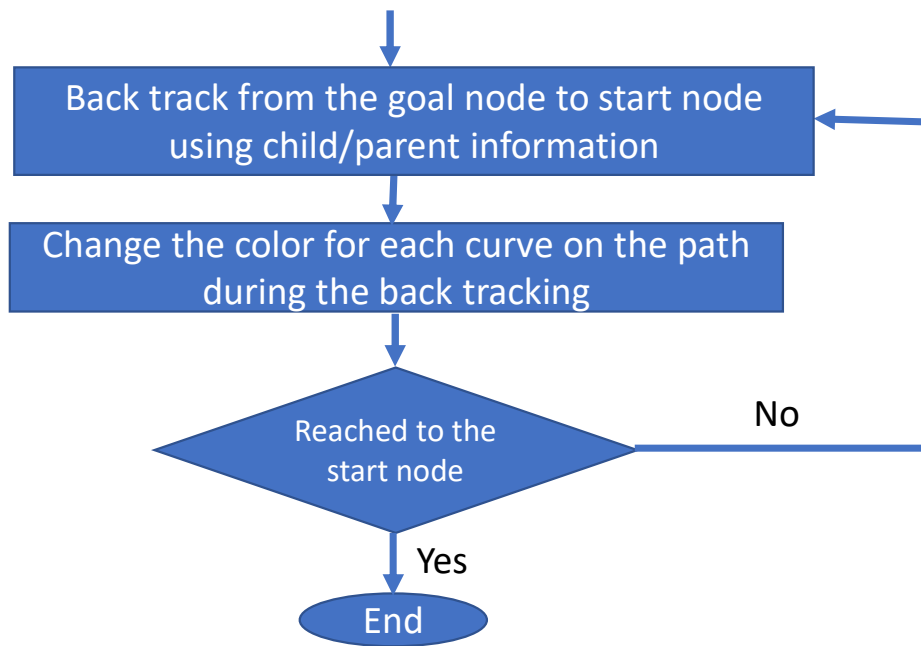
\*Optional

# 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 set of action space .







# 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

Note: To simplify path explored, final orientation input is not required.

# 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)

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

The parameters should be referred from the python file.



# 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 revolution per minute (RPM) and the second element corresponds to the right wheel RPM.

Example: `actions= [[50,50],[50,0],[0,50],[50,100],[100,50],[100,100],[0,100],[100,0]]`

## 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

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

# 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

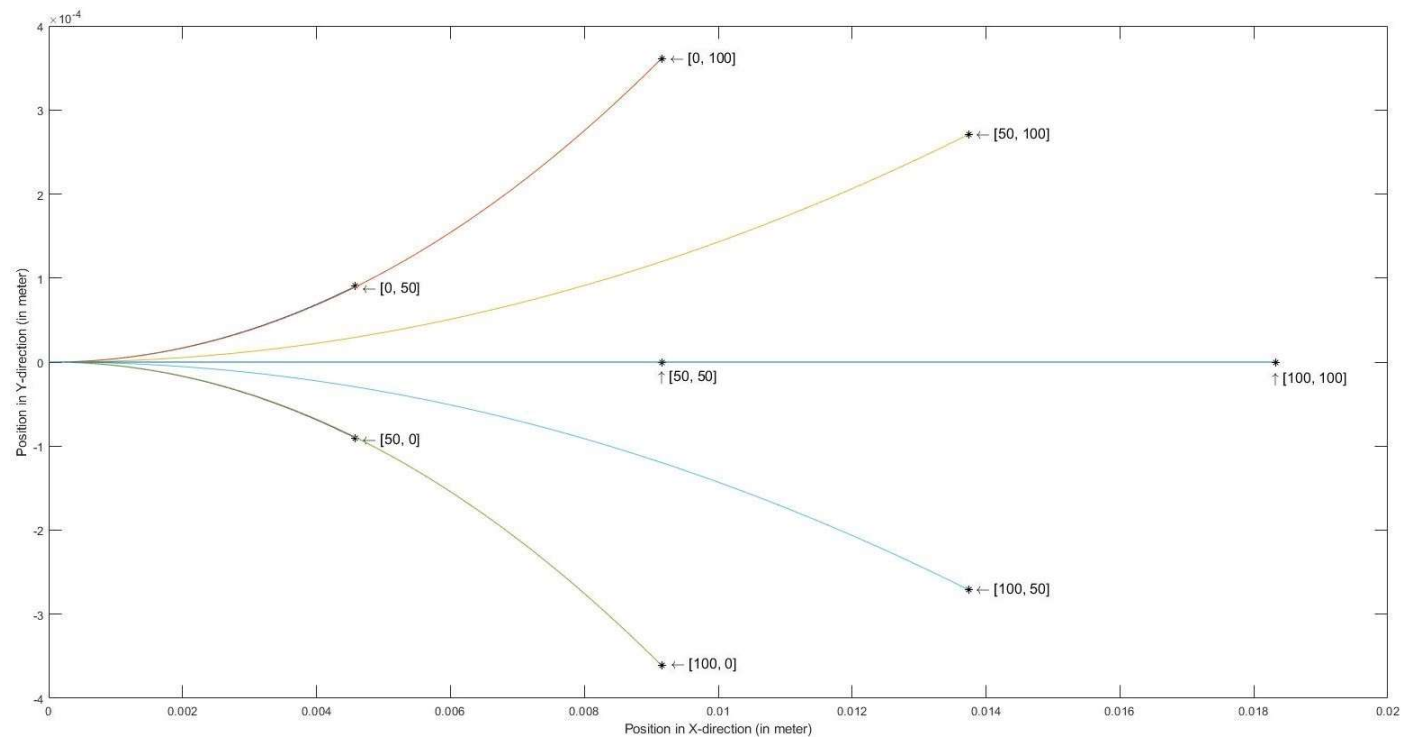
$$dx = \frac{r}{2} (u_l + u_r) \cos \theta dt$$

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

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

- You can refer python file for the equations.

# Differential Drive Constraints (Continued..)



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

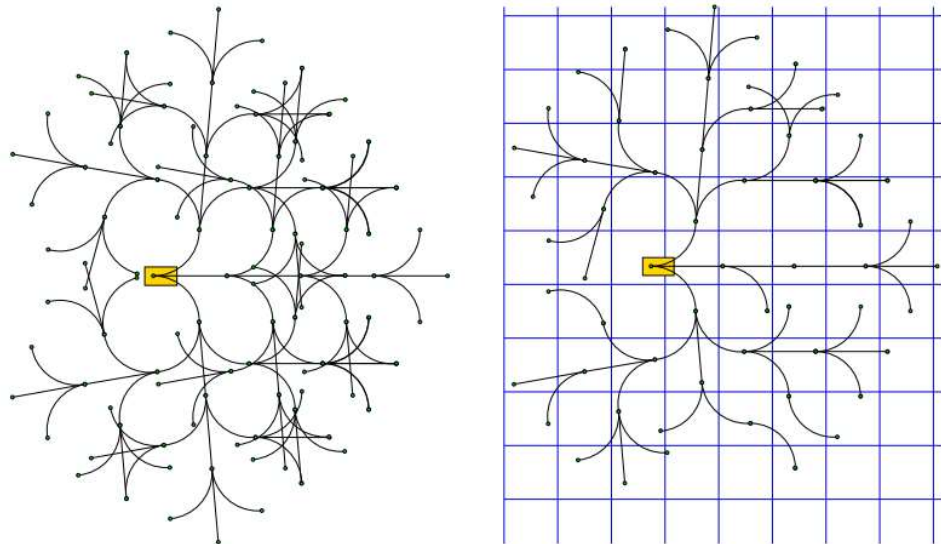
## Step 2) Modify the map to consider the geometry of the rigid robot

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



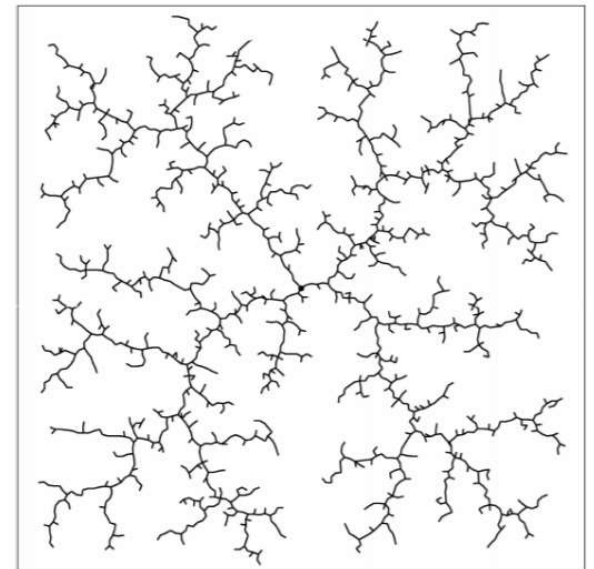
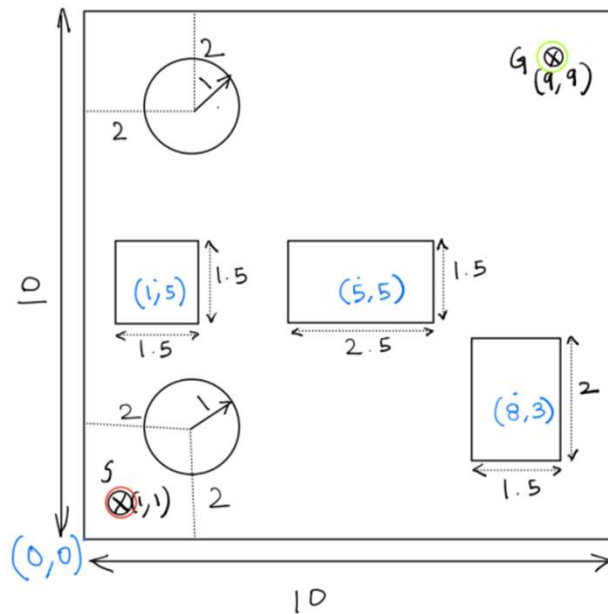
## Step 3) Generate the tree using non-holonomic constraints

- Consider the configuration space as a 3 dimensional space.
- Follow the same step from Project 3- Phase 2 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

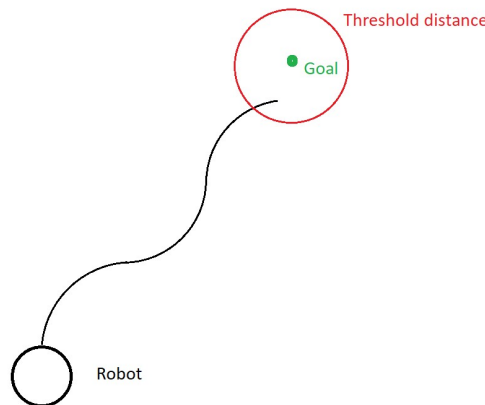
- Use curve that address the non-holonomic constraints to connect the new node to previous nodes and display it on the Map.





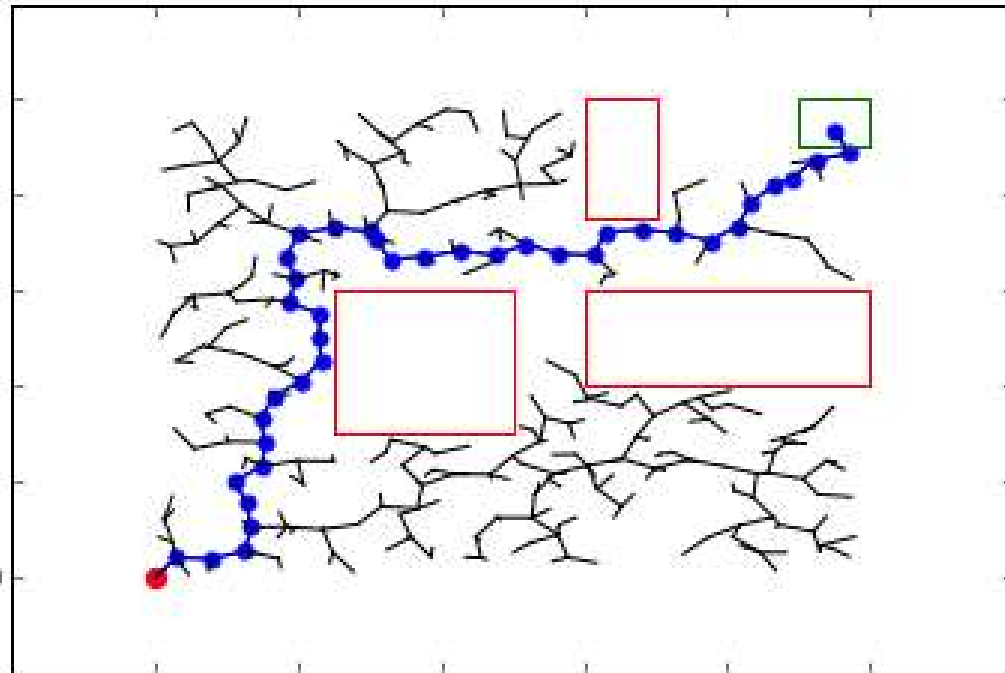
Step 5) implement A\* search algorithm to search the tree and to find the optimal path

- Consider Euclidean distance as a heuristic function.
- Note - You must 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 must 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).



# 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.

## Part 2 description

- Simulate the path planning implementation on Gazebo.
  - Gazebo environment has been provided for the map (map.world file)

# Important pointers

- To run the simulation in ROS, you should have everything wrapped in only one launch file.
- Make note of the coordinate system in Gazebo, and the position of the origin. Input will be based on the coordinates in Gazebo.
- User input should be from Terminal.
- 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.

# Helpful links for ROS

- Writing a publisher/subscriber node –

<http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29>

- Initialize robot pose - <https://answers.ros.org/question/40627/how-do-i-set-the-initial-pose-of-a-robot-in-gazebo/>

- Load custom Gazebo environment/ Creating ROS package - [http://gazebosim.org/tutorials?tut=ros\\_roslaunch](http://gazebosim.org/tutorials?tut=ros_roslaunch)

# Submission

- The submission will be in two parts.
- The Zip file should have two different folders for each part.

# Submission Details Part 1

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

proj3\_groupnumber\_simulationSoftware

- code
- simulation video
- readme.txt
- GitHub Link

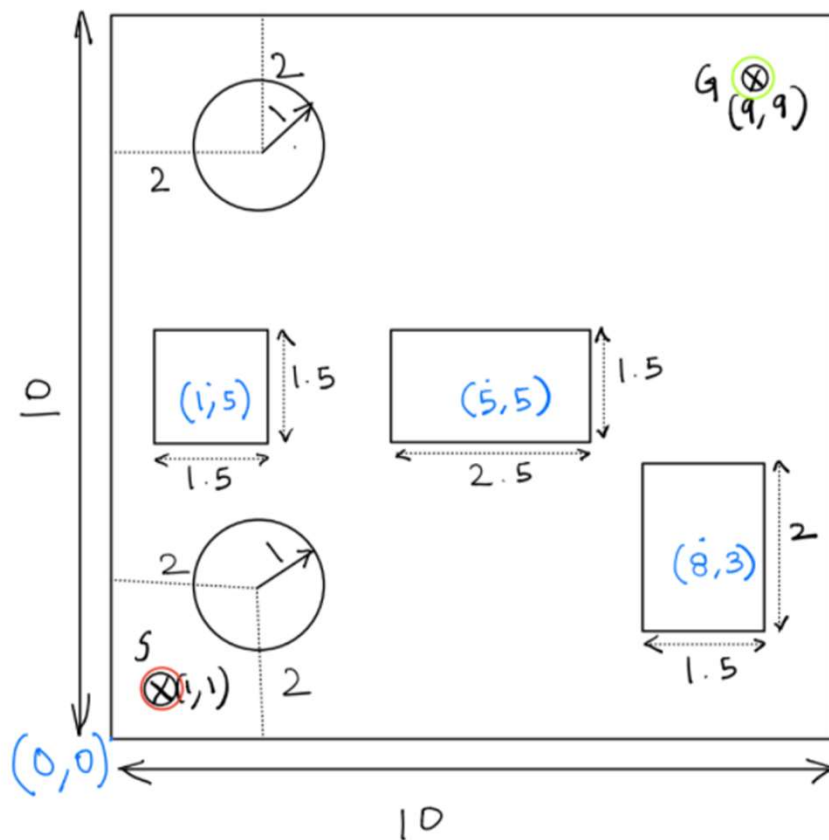


## Submission Details Part 2

- Source code
- GitHub (one repository link only) and ReadMe
- Simulation results (video of the simulation)
  - 'Video1'
  - 'Video2'
- \* Details for the videos are mentioned in the next slides

There is weightage w.r.t to following instruction for folder name, directory structure, one launch file(ROS)/script (VREP) etc.

# Video



The approximate start point is shown in red and goal point is shown in green. The video should show the Turtlebot motion in Gazebo environment for these points.

- Choose the points by your best judgment along with other user inputs.
- You are not required to implement a controller, this will be open-loop and hence, it is okay if the robot does not follow the exact waypoints.