ENPM661 - Spring 2023

Project 03 - Phase 2

Implementation of the A* algorithm on a Differential Drive (non-holonomic) TurtleBot3 robot

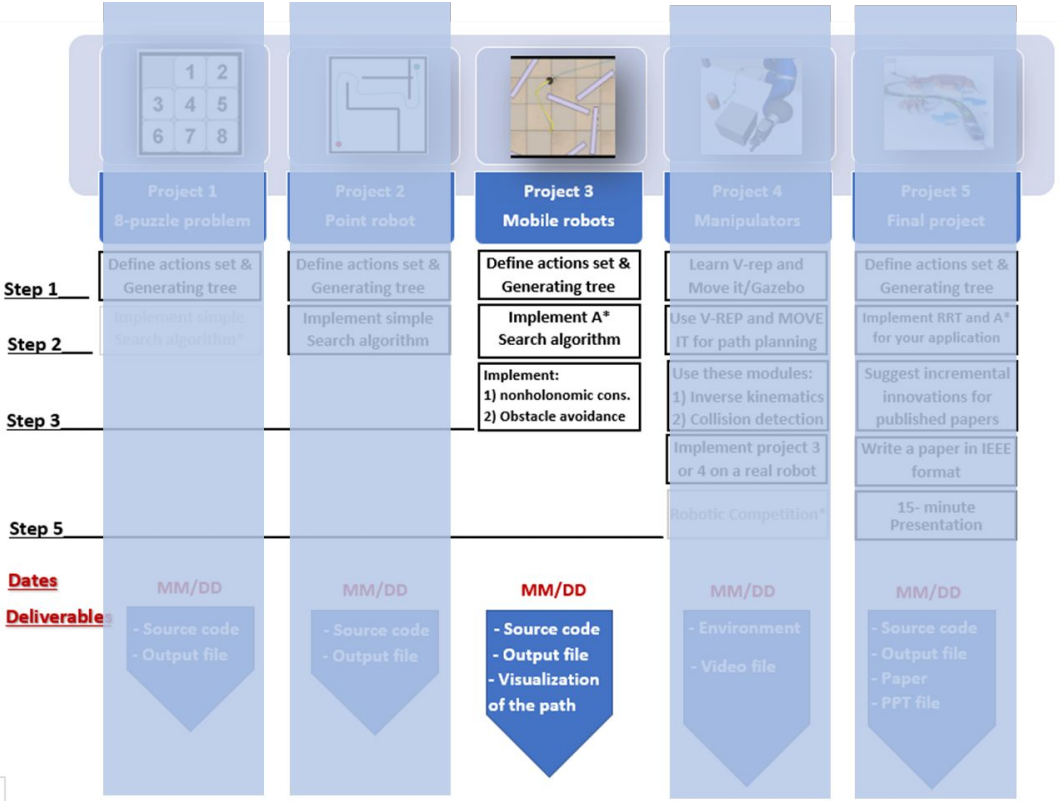
Note: This is a group project (max two students*)

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

Due Date: April 07, 11:59 PM

Points/Weightage: 10

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

Project 03 Phase 2: Description

- Navigate a differential drive robot (TurtleBot3 **Burger**) in the given map environment from a given start point to a given goal point.
    - 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.
- Consider differential drive constraints while implementing the A* algorithm, with 8 set of action space.

Project 03 Phase 2: User input

- Your code must take 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 (2-element vector): (X_g, Y_g)
    - To simplify the path explored, final orientation input is not required.
  - Wheel RPMs (2-element vector) => (RPM1, RPM2)
    - RPM: Revolutions per Minute
    - 2 possible values for the wheel RPMs
  - Clearance (in mm)

Project 03 Phase 2: Parameters

- Your code must define the following parameters of the TurtleBot 3 Burger robot.
- These parameters are NOT user defined but rather are constants which can be defined in your Python code.
    - Robot Wheel Radius (R)
        - From the robot's datasheet/documentation
    - Robot Radius (r)
        - From the robot's datasheet/documentation
        - This needs to be added to the clearance value to create the obstacle map.
    - Wheel Distance (L)
        - From the robot's datasheet/documentation

Project 03 Phase 2: Action set

- Let the two RPMs provided by the user be `RPM1` and `RPM2`. Then the action space consisting of 8 possible actions for the A* algorithm is:

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

where, the 1st element in each vector above corresponds to the Left Wheel's RPM and the 2nd element in each vector above corresponds to the Right Wheel's RPM.

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

Part 01: 2D Implementation

Differential Drive Constraints

- For this project you consider the robot as a non-holonomic robot which means the robot cannot move in the 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 a Differential Drive robot are:

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

From the velocity equations, the distance travelled and angle covered in each time step, *dt* can be calculated as:

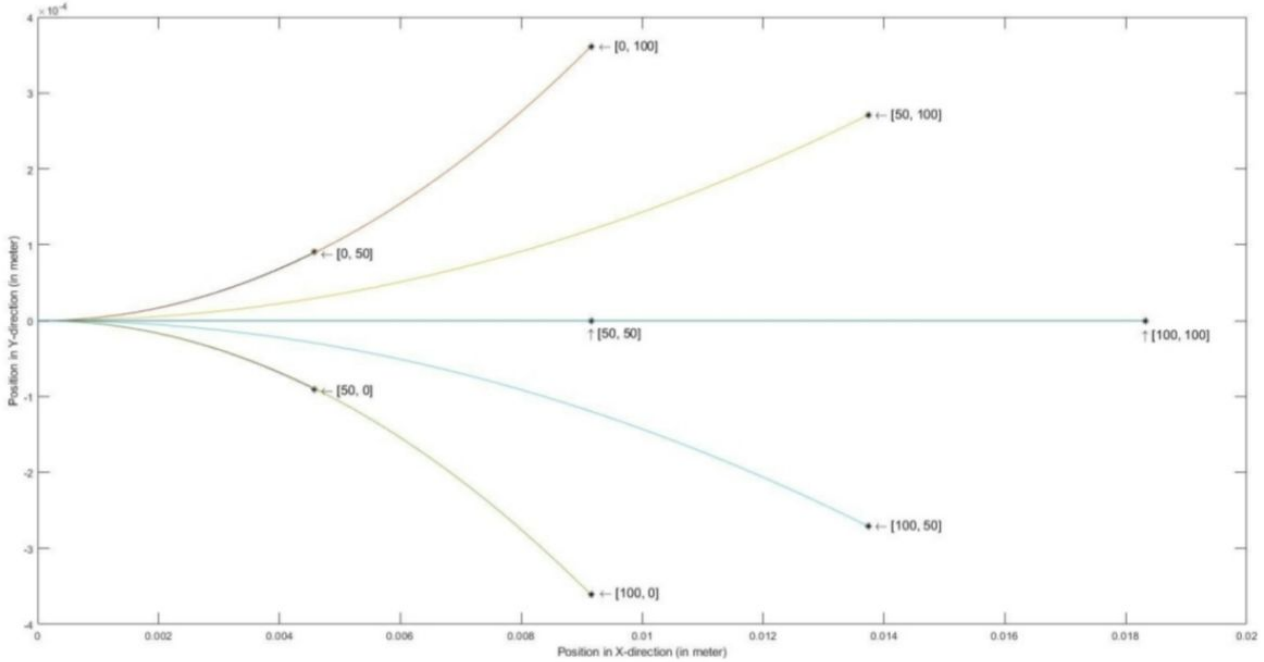
$$dx = \frac{r}{2}(u_l + u_r)\cos\theta\, dt$$
$$dy = \frac{r}{2}(u_l + u_r)\sin\theta\, dt$$
$$d\theta = \frac{r}{L}(u_r - u_l)\, dt$$

where,

- x_dot and y_dot: Velocities in x and y directions, respectively
- u_l and u_r: Left and Right wheel velocities, respectively
- r : Wheel radius
- L: Distance between the two wheels

Refer to the given Python file for the equations.

Differential Drive Constraints



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

Step 01: Function for non-holonomic constraints

- Write a function that will take 2 arguments (Rotational velocities of the Left wheel and Right wheel) and returns the new coordinate of the robot, i.e. (x, y, theta).
    - where x and y are the translational coordinates of the robot and theta shows the orientation of the robot with respect to the x-axis.
- A sample is provided in the python file. You may choose to modify the function or implement your own.

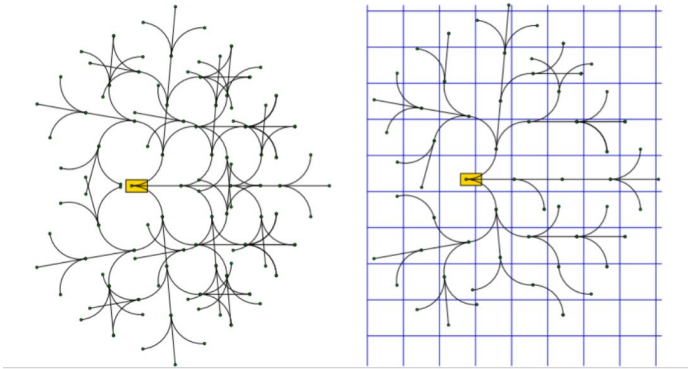
Step 02: Modify the Map to consider the geometry of the Rigid Robot

- Dimensions of the robot are available in the Official Documentation which can be used to define the clearance value.

- The map with obstacles can be setup using any inbuilt function of OpenCV/Matplotlib/Pygame. However, it is recommended to use the half-plane equations method.

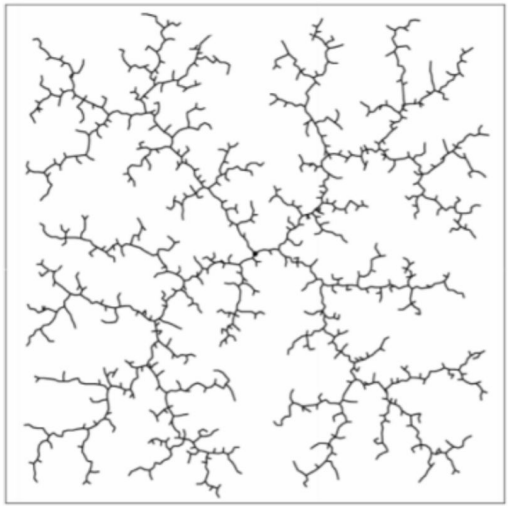
Step 03: Generate the tree using non-holonomic constraints

- 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)



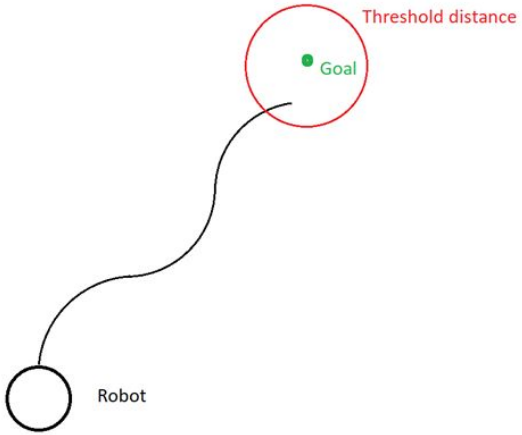
Step 04: Display the tree in the configuration space

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



Step 05: Implement A* search algorithm to search the tree and to find the optimal path

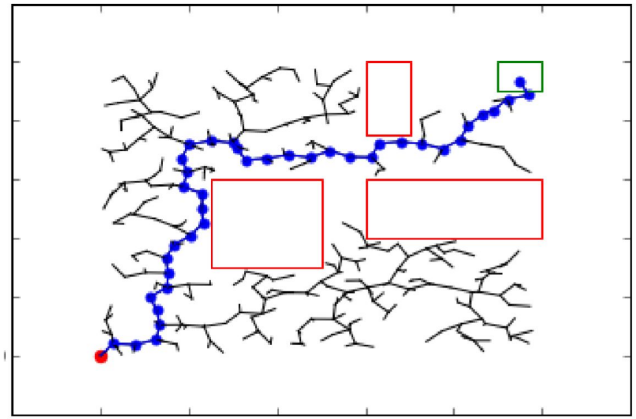
- Consider Euclidean distance as a heuristic function.
- Note:- Define a reasonable threshold value for the distance to the goal point. Do to the limited number of moves the robot cannot reach the exact goal location. So, use a threshold distance to check the goal.



Step 06: Display the optimal path in the Map

- To plot the final path, you can use the `plot_curve` function as shown in the given `Howplotcurves.py` Python file. You may also use other libraries such as `quiver` to help you plot the path lines of the robot.

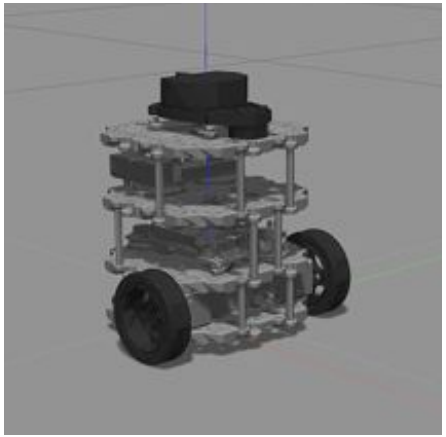
- For Part 01 2D Implementation, reuse the Project 03 Phase 01 map.



Part 02: Gazebo Visualization

Implement Gazebo Visualization

- Simulate the path planning implementation on Gazebo with the TurtleBot 3 Burger robot.
- The Gazebo environment has been provided for the map (`map.world`), which is different from that used in Part 01 2D Implementation. It can be found in the project folder on Canvas under Files -> Projects -> 2023 -> Project 3 -> Phase 2 -> ROS.

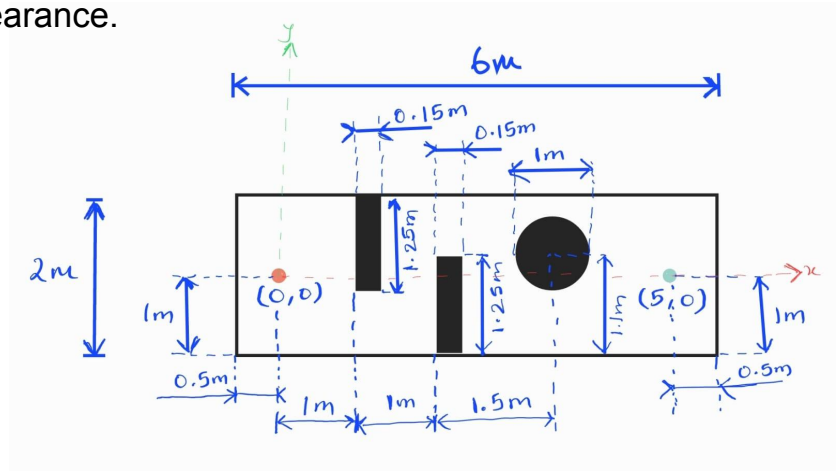


Gazebo Map

- The approximate start point is shown in red and goal point is shown in green in the Figure below.
  - You are free to choose any start point in the space to the left of the 1st rectangle and the goal point anywhere in the space above/below/to the right of the shown black circle.

- The video should show the TurtleBot motion in Gazebo environment for these points. The motion of the TurtleBot should be as a result of the solution generated by your A* algorithm and the corresponding velocity commands published to the appropriate ROS topic.

- Choose the start and goal points by your best judgement along with other user inputs, specifically the wheel RPMs and clearance.



The origin shown here is as represented in the Gazebo world.

Important Information

- To run the simulation in ROS, you should have everything wrapped in only one launch file.
- You are NOT required to implement a controller to make the TurtleBot accurately follow the trajectory. This will be an open-loop controller and hence, it is okay if the robot does not follow the exact waypoints accurately.
- Make note of the coordinate system in Gazebo, and the position of the origin. Input will be based on the coordinates in Gazebo for Part 02: Gazebo Visualization. Quantities in ROS are reported in meters, radians, meter/sec, and rad/sec. Each tile in the Gazebo world is 1m x 1m.
- User input should be from the Terminal and not hardcoded into the code. Decide the clearance and wheel RPMs on your own.
- Helpful Links for ROS/TurtleBot3:
  - ROS Tutorials
    - ENPM661/662 Software Sessions Drive
  - Writing a publisher/subscriber node:
    - http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29
    - ENPM662 ROS Publisher/Subscriber Software Session
  - Initialize Robot Pose:
    - https://answers.ros.org/question/40627/how-do-i-set-the-inital-pose-of-a-robot-in-gazebo/
  - Load custom Gazebo environment:
    - http://gazebosim.org/tutorials?tut=ros_roslaunch
  - TurtleBot3 Installation + Simulation
    - Official Documentation: Installation, Simulation [Choose Noetic]
    - ENPM661 TurtleBot3 Software Session

Project 03 Phase 02: Deliverables

- **proj3p2_firstname#1_firstname#2_firstname#3.zip.** This zip file should contain the following folders & files in the below tree structure.
  - Part01 folder
    - Source Code (.py)
  - Part02 folder
    - ROS Package
      - src
        - <your-node-script>.py
      - launch
        - <your-launch-file>.launch
      - world
        - map.world
  - README file (.md or .txt)
    - Must properly describe how to run the code (for Part01) and launch the ROS Node (for Part02).
      - Mention all the inputs (**give an example set of inputs to run from the terminal**)
    - 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.
    - GitHub Repository Link containing the code(s)
      - Should contain commits (at least 5) with appropriate commit messages, in total, between all members.
      - Initially, set the visibility of the repository to Private and change this to ***Public*** after the deadline date.
    - Simulation Video Links
      - 2 Videos [Reference videos can be found in the project folder]
        - For Part 01: Give a start point (near the bottom left) and end point (near the bottom right).
        - For Part 02: TurtleBot3 Gazebo simulation in the given map with start point (left of the 1st rectangle) and goal point (in the space above/below/to the right of the shown black circle).
      - Do NOT include the .mp4 files in your ZIP file, upload both videos to Google Drive/YouTube (unlisted, if required) and include the links in the README (ensure proper view access)

- **proj3p2_firstname#1_firstname#2_firstname#3.pdf.** This file should contain the source code (Both Part01 and Part02) for plagiarism check. ***This file needs to be submitted ALONG with the ZIP file, and NOT INSIDE the ZIP file***.