MICROMOUSE BOT

PROJECT REPORT

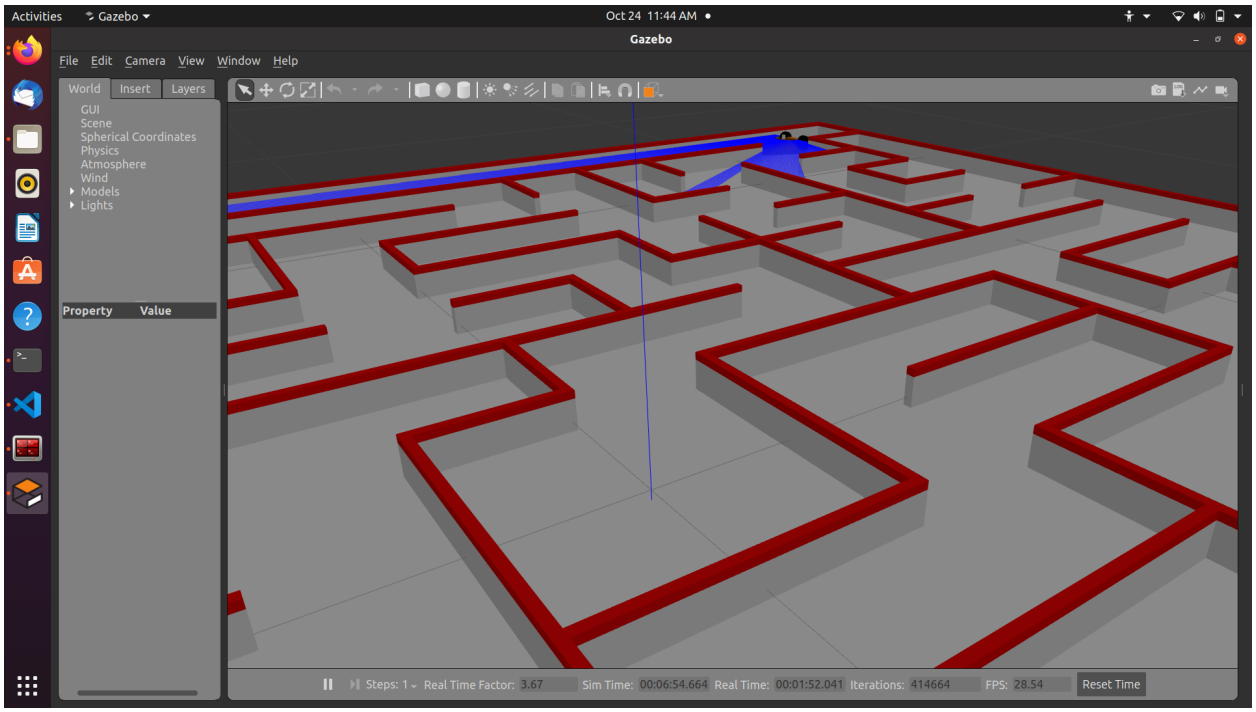
EKLAVYA MENTORSHIP PROGRAMME
At
SOCIETY OF ROBOTICS AND AUTOMATION,
VEERMATA JIJABAI TECHNOLOGICAL INSTITUTE
MUMBAI

SEPT 2021

1. PROJECT OVERVIEW:

1.1 DESCRIPTION OF USE CASE AND PROJECT:

A micromouse is a small, autonomous self-contained bot which can get to the centre of a maze in the shortest possible time.
A micromouse essentially comprises of a drive motor or motors to move it; a steering and turning method and sensors to detect the presence or absence of maze walls which has to be specified in the URDF and sensors or control logic to oversee the action of the rest and keep the vehicle 'on track' or to solve the maze.



1.2 TECHNOLOGY USED:

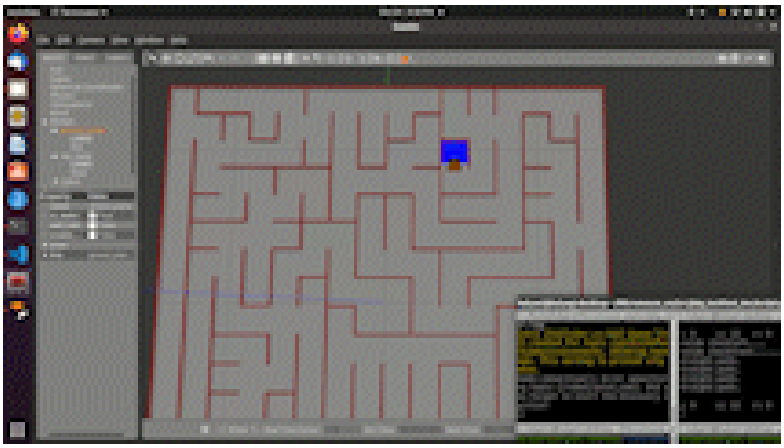
1. **ROS** : The Robot Operating System, or ROS, is a framework for writing robot software. It is a collection of tools, libraries and conventions that aim to specify the task of creating complex and robust robot behaviour across a wide variety of robotic platforms. It is used here to obtain data from the laser scanner at the front of the bot, and to publish linear and angular velocities to the bot based on the laser scanner values obtained. The odometry messages obtained from the bot are used to calculate the euler angles of the bot which aid in its rotation.

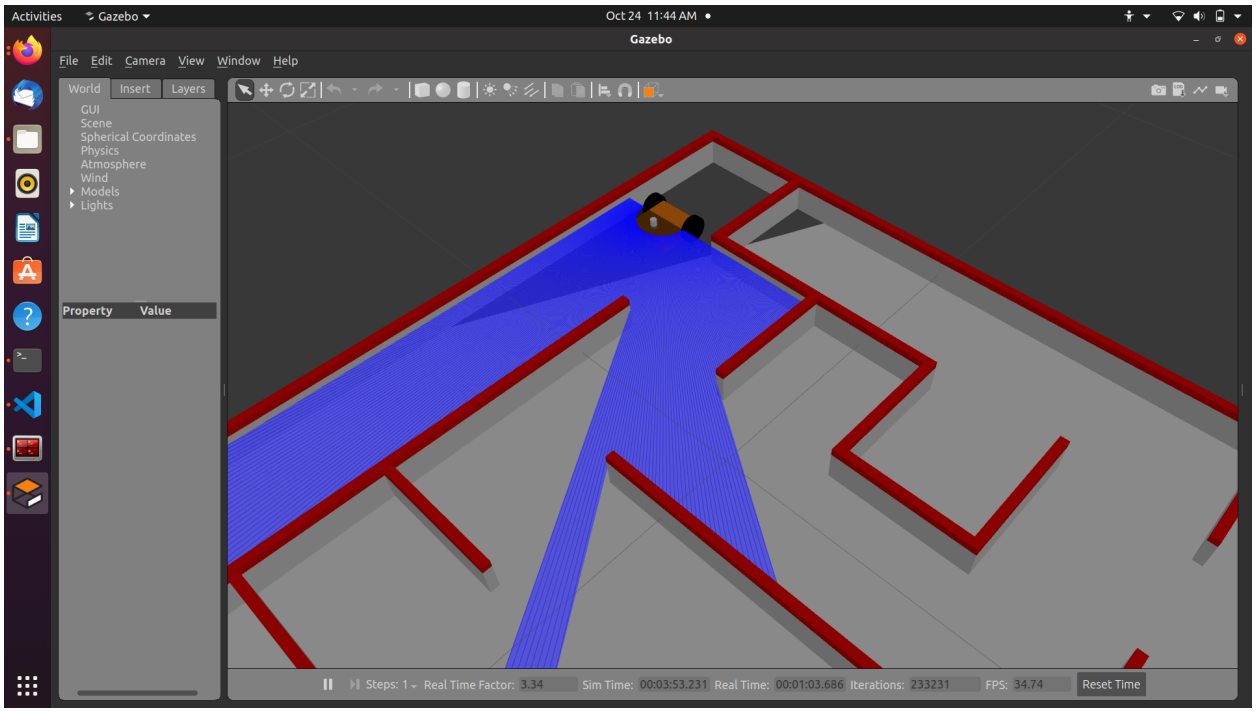
2. **Gazebo** : Gazebo, a powerful simulation software, is used to simulate the maze arena and the micromouse urdf model. Different mazes can be generated as per our preferences from gzmaze package of ROS

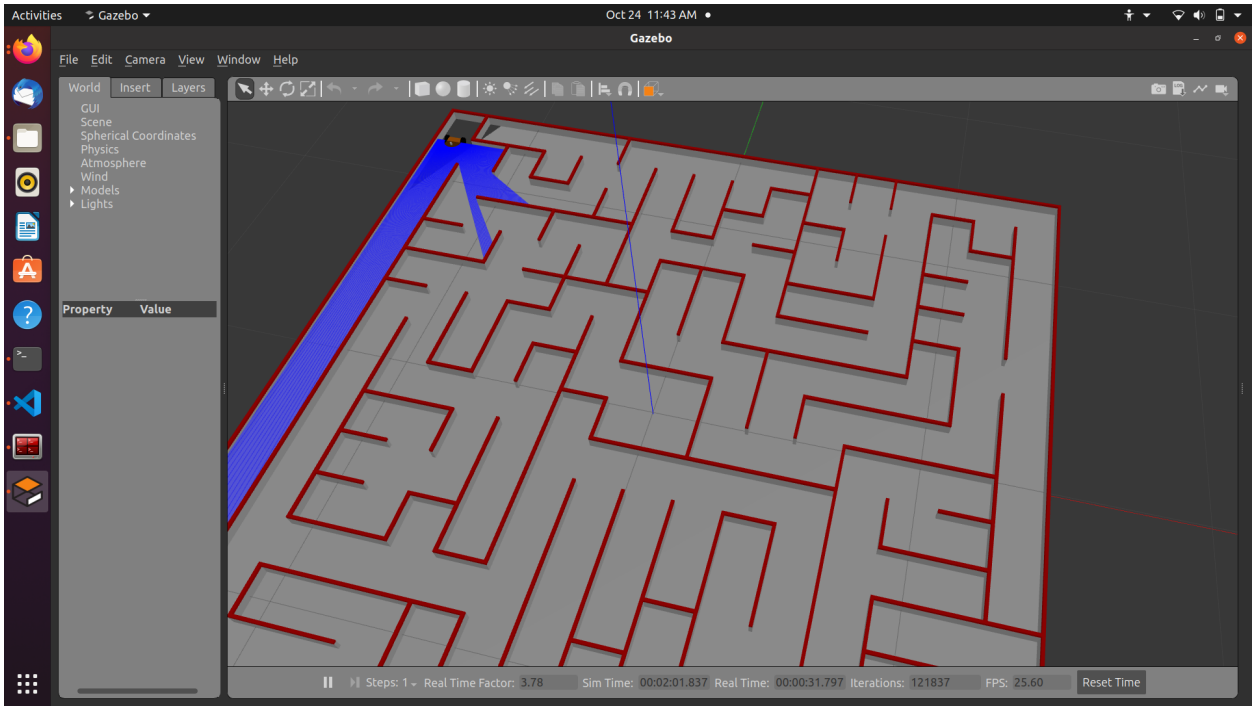
3. **RViz** : Although not necessary, Rviz, a visualiser helps visualize the maze through the micromouse's eyes, as if you were inside the maze. The walls detected by the lasers emitted from the micromouse can be seen in red and it's amazing to watch.

1.3 BRIEF IDEA:

Micromouse is an event where small robot mice solve a 16×16 maze. It began in the late 1970s in Japan. The maze is made up of a 16×16 grid of cells, each 180 mm square with walls 50 mm high. The mice are completely autonomous robots that must find their way from a predetermined starting position to the central area of the maze unaided. The mouse needs to keep track of where it is, discover walls as it explores, map out the maze and detect when it has reached the goal. All of this is done in Gazebo for this project.




Activities
Gazebo
Oct 24 11:44 AM
Gazebo
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Atmosphere
Wind
Models
Lights
Property Value
Steps: 1 Real Time Factor: 3.34 Sim Time: 00:03:53.231 Real Time: 00:01:03.686 Iterations: 233231 FPS: 34.74
Reset Time

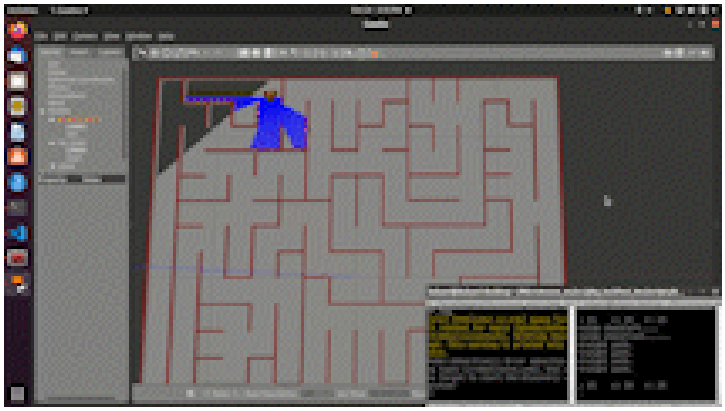


2. WORKFLOW :

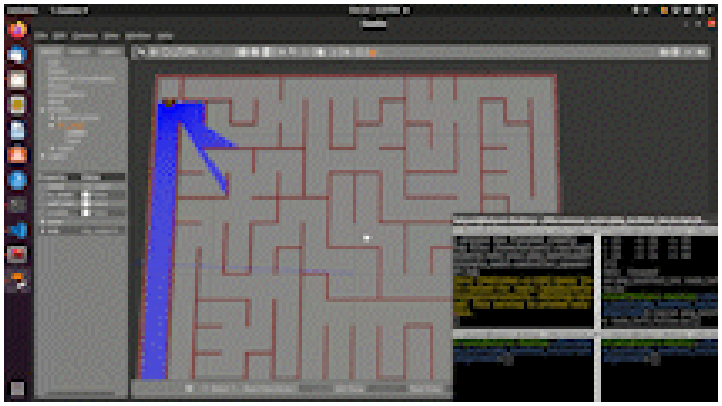
- I already had the urdf model of the bot required and the model of the maze, which I had spawned in Gazebo. So I didn't need to design them on my own.
- Starting with the basics, the robot needs to move forward (maximum permitted speed is 0.4 m/s) and take left, right or U turns as per it needs. So for these linear.x and angular.z velocities are published to the cmd_vel topic of the bot in the form of geometry_msgs/Twist messages.
- Next, it needs to see and take in its surroundings in order to perceive what movements it has to make. For this, their is a laser scanner at the front of the bot that sends out 360 beams in an area spanning -90 to +90 degrees in front of it. Each degree of area in front of the bot contains 2 beams. These beams hit the walls of the maze and get reflected back to the bot; they are received by a receiver which can perceive the distance that the reflecting

surface for that particular beam was. 360 beams means 360 different values are obtained by the receiver in the form of sensor_msgs/LaserScan messages, which are then published to the /my_mm_robot/laser/scan topic; msg.ranges[:] returns a list of values of the distance detected for each beam in the selected range in [ ].

- For determination of rotation, yaw of the bot is obtained from its quaternion values, which in turn are obtained from the /odom topic in the form of Odometry messages. Rotation of the bot is done using the obtained yaw values instead of using time to rotate by 90 or -90 or 180 degrees as required.
- Various algorithms like left or right follow rule, flood-fill, DFS, BFS, Djikstra's algorithm or A * algorithm, etc can be used. For now, I have implemented the left follow algorithm to solve the maze.



- PID too needs to be used to keep the bot at the centre of the path and so that it doesn't collide with the walls gradually and make unnecessary turns and increase the time taken to solve the maze. Without PID tuning the bot may also get stuck in a loop of collisions.



2. ALGORITHMS :

2.1 LEFT FOLLOW ALGORITHM :

The basic concept of this algorithm is that whenever there is a chance to take a left turn, take a left turn. The next preference goes to going straight, when a left turn isn't possible. Lastly if either of these aren't possible, take a right turn. For its implementation, first the bot checks if there is a wall on the left. If not, then it takes a left turn, else it checks if there is a wall in front of it. If there isn't a wall in front, it keeps moving straight ahead, else, checks if there's a wall on the right or not. If there's a wall on the right, it takes a U turn (dead end) , else it turns right.

2.1 RIGHT FOLLOW ALGORITHM :

This algorithm is exactly like the left follow algorithm just that here, the priority order is : Right turn> Going straight> Left turn

2.3 DFS ALGORITHM :

In DFS or Depth First Search Algorithm, the bot checks out one node at a time. A node can be defined as any point in the maze where more than one turning operations can be done, like T shape, Plus, Straight+left and Straight+right nodes.

abcdefghijk

2.3 BFS ALGORITHM :

abcdefghijk

2.4 FLOOD FILL ALGORITHM :

abcdefghijk

3. RESULTS :

My micromouse is able to reach the centre of the maze in ____ mins with the left follow algorithm. With the right follow algorithm, it takes _____mins. Havent yet implemented the above.

4. PROBLEMS FACED :

1. First off, I worked on making a solidworks model of an omniwheeled bot to reduce the complexity arising due to turning in the usual 2 wheeled bot. I couldn't find a usable model of such a bot online, hence I had to design the chassis of the bot on solidworks and import the omniwheel SLDPRT model I found online. After assembling all those parts in an SLDASM, I needed to export it to URDF which was a tedious job since there were 60+ links. So then I went for using xacro and .stl files to simplify the job by referring to the xacro files from the repo _____, and launched the model in Gazebo to get a really weird result. It was quite different from the bot I expected and seemed really hopeless so I decided to scrap the omnibot idea and go with the micromouse model provided by Techfest Micromouse.

2. Next, I worked on the rotations in the techfest model for which I utilised yaw values & target yaw values and published velocities accordingly to the bot. The bot wouldn't turn to yaw=0 values at all and I thought there was some problem with the callback since the callback didn't assign the current yaw values to the yaw variable (which was initialised as zero). I tried many things to figure out what the problem and get around it but I couldn't. So I simply initialised the yaw variable to 1000 and it worked (since yaw can't be greater than +180)
3. Working on the rest; faced minor problems but got around them :)

5. CONCLUSION :

1. This project was mainly ROS based, and it really helped me clear a lot of ROS stuff and strengthen ROS concepts.
2. Gazebo too is required so I got comfortable with Gazebo too because of this project.
3. Even though i couldn't use my own omniwheel solidworks model, i got to learn Solidworks and learn about URDF and XACRO files
4. I couldn't cover so many other other algorithms due to some problems I faced during the project, but I plan on continuing working on them on my own even after Eklavya ends, and ask mentors about any doubts that I face, if it's ok with them.

5. PENDING WORK :

-Implementing all the other algorithms that I mentioned at the start.
-Implementing PID tuning in the bot so it solves the maze more smoothly

-Work on making it remember the correct turns to be taken at every junction/node so that one final run can be made out of the 7 allowed in which it travels to the centre with the shortest possible path, by storing a text in text file.
-Make a final.launch such that all the necessary ROS nodes, RViz and Gazebo Simulation to solve the maze can be launched at once.
-Learn about gzmaze so that a maze of any desired structure can be generated.