DOCUMENTATION

# **Simulation of Autonomous rover**

# 

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **Sl.No** | **Content** | **Page No** |
| 1 | Project Outline | 3 |
| 2 | Scope | 3 |
| 3 | Cost | 3 |
| 4 | Installation | 4 |
| 5 | Gazebo | 4 |
| 6 | Sample maze generation | 5 |
| 7 | Rover creation | 8 |
| 8 | Use-case description | 11 |
| 9 | Commands for starting automatic rover | 12 |
| 10 | What we learned | 13 |
| 11 | Summary and Conclusion | 14 |
| 12 | Feature Enhancement | 14 |

**Project Outline:**

“Rover can explore places where humans cannot”.

Simulation made lives easier by reducing the implementation costs. Making the rover explore the maze by detecting and avoiding the obstacles. Rover will be able to generate its own path and generate a map of the maze. The rover should be able to take pictures of the maze. Sends the data continuously to the receiver and keeps the receiver updated about the Map data. Our vision for the project is to simulate a rover that can autonomously travel through a static maze which is an anonymous 2-d plane by detecting the obstacles, generating its own map, and sending the feedback.

**Scope:**

This project was developed in a span of 6 sprints. Each sprint has its own set of deliverables and all the sprint goals were met at the end of each sprint. Initially, new user stories are created in the product backlog, then each of them is taken into sprint depending on the sprint goals. Each team member worked on different user stories and the other team member tested the other's code at the end of the sprint.

The sprint process goes like this, at the beginning of the sprint, each team member analyses the task assigned in order to implement the user stories. Then the implementation part is taken care of. Finally, testing takes place at the completion of the user story. Each sprint lasted for 2 weeks and three scrum calls has taken place per week, the status of the project was discussed during the scrum calls. After successful completion of the user stories, each story will be closed with proper comments.

**Cost:**

For the implementation of this project, all the software used was free-to-use the software.

**Engineering Tasks:**

The project design was decided at the beginning of the project. Project requirements were analyzed and updated every sprint. At regular intervals of time, we have connected with the product owner ( DAVID ) and discussed the gaps in the project. Based on the vision and goal of the project, operational environment (Linux and Ros2), and added business value of a simulated autonomous rover, we have arrived at the requirements elicitation by having multiple brainstorming sessions within the team.

**Equipment used:**

Since it is a simulation project, Development and testing require manpower but no external hardware apart from the laptop is required. A laptop with the following specifications is preferred for the execution of this project, intel i5 or ryzen3500, 8GB ram, 100Gb storage.

**Materials used:**

No materials were used in the development of the project.

**Risk and Measures:**

Since this is a simulation project, there is no risk involved

Measures to be followed in order to have a successful project are:

The data collected using the camera has to be stored in a proper manner so that it can be accessed later and protect the data so that it won't fall into wrong hands.

**Installation:**

The following are the steps for the installation of the required software for rover simulation.

**Virtual Machine and Ubuntu installation:**

Install Oracle VM RAM size - 8192MB Choose the option to create a virtual hard drive. Hard Drive file type - VDI(VirtualBox Disk Image) Storage on a physical hard drive - Fixed size. Virtual hard drive size - 50- 100Gb Create virtual machine and wait until creation is accomplished. Load ubuntu 20.0.4 .iso image. Ubuntu->setting->Controller: IDE->CD/DVD drive-> IDE Secondary Master. Ubuntu->setting->System->Processor-> CPU-> 4 Ubuntu->setting->System->Execution Cap-> 100 Ubuntu-Power ON -> Install Ubuntu Configure language and username and password. Ready to explore ubuntu!

**ROS2 foxy-Debian:**

Set Locale: locale # check for UTF-8

sudo apt update && sudo apt install locales sudo locale-gen en\_US en\_US.UTF-8 sudo update-locale LC\_ALL=en\_US.UTF-8 LANG=en\_US.UTF-8 export LANG=en\_US.UTF-8

locale # verify settings

Setup sources: sudo apt update && sudo apt install curl gnupg2 lsb-release sudo curl -sSL <https://raw.githubusercontent.com/ros/rosdistro/master/ros.key> -o/usr/share/keyrings/ros-archive-keyring.gpg

echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/ros-archive-keyring.gpg] <http://packages.ros.org/ros2/ubuntu> $(source /etc/os-release && echo $UBUNTU\_CODENAME) main" | sudo tee /etc/apt/sources.list.d/ros2.list > /dev/null

**Ros2 packages**:

sudo apt update sudo apt install ros-foxy-desktop sudo apt install ros-foxy-ros-base

**Gazebo**: sudo apt update sudo apt install gazebo

**Reference:**

Installation: <https://brb.nci.nih.gov/seqtools/installUbuntu.html> <https://docs.ros.org/en/foxy/Installation/Ubuntu-Install-Debians.html> <https://navigation.ros.org/tutorials/docs/navigation2_with_slam.html> <http://gazebosim.org/tutorials?tut=ros2_installing&cat=connect_ros>

**Exploring Mazes and creation using Gazebo**

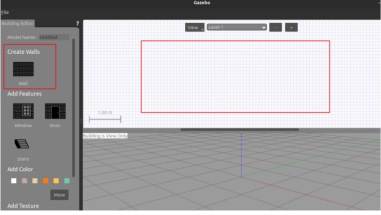
**Gazebo: -**

The gazebo is a sophisticated robot simulator that calculates physics, produces sensor data, and provides user-friendly interfaces. It is used by both businesses and academics.

**Sample Maze Creation:-**

In the Gazebo, we can see an Edit button on the left side after opening the Gazebo. We can select Building editor.

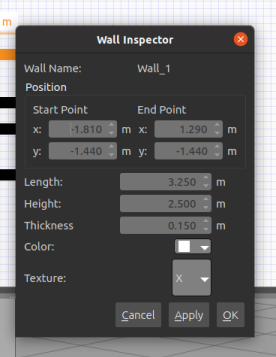
It opens a screen with a simulation screen where that shows a matrix on the screen. It is a white 2d matrix where we can design our two- dimensional maze.



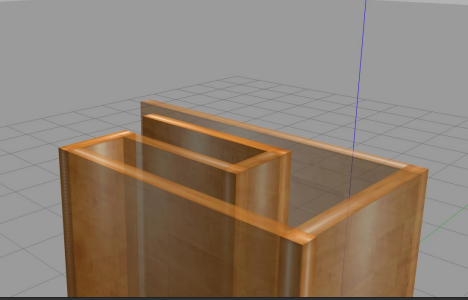
As we are creating a 2D maze we are including only walls in the gazebo. We can use Create Walls feature to add walls

Draw the required 2D maze in the given space with the required dimension and specification. The maze must have at least one entry point and one exit point and can have multiple walls.

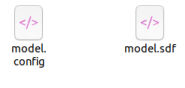
The dimension can be adjusted by double-clicking on the wall. It opens a wall inspector tab where you can set all the dimensions.



Once the drawing is complete you can see its appearance on the gazebo simulation screen. You find the position of the maze in the 3D plane here and can change its position after seeing it in the world file. The maze can be textured by using the texture option on the left tab (bricks, Woods etcetera ).



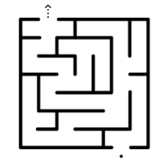
For saving the maze select file and save as (filename). The model is saved in a folder with a given name and it has two files mode.sdf and model. config.



**Types of Mazes that can be used**

**Standard maze (Single entry point single exit point)**

The maze has a single entry point and a single exit point and there are multiple walls that create multiple paths.



**Single entry point multiple exits points**

The maze has a single entry point and multiple exit points and there are multiple walls in between which create multiple paths.

A picture containing diagram

Description automatically generated

**Circular Maze**

The maze has a single entry point and Circular walls. The rover must move in circular paths to reach the centre of the maze which is the destination point.

Logo

Description automatically generated

**Creation of Rover model using Gazebo**

**Gazebo: -**

The gazebo is a sophisticated robot simulator that calculates physics, produces sensor data, and provides user-friendly interfaces. It is used by both businesses and academics.

**Rover Creation:-**

Create a folder for the rover which contains a model.sdf file and model. config file.

Create a closed box by making necessary changes in the model.sdf file and set the dimensions of the rover according to the requirements.

The pose (i.e. position and orientation) of the geometric center of this box is set at a position of (x = 0 meters, y = 0 meters, z = 0.1 meters) and an orientation of (roll = 0 radians, pitch = 0 radians, yaw = 0 radians). The pose of the chassis (as well as the post of any link in a sdf file) is defined relative to the model coordinate frame.

Inside the visual tag, we place the shape that Gazebo’s rendering engine will use to display the robot.

Open the gazebo and start creating the rover. On the left panel, click “My Robot”. You should see a  white box. You can place it wherever you want.

A picture containing text, coffee table

Description automatically generated

Now we can add a caster wheel to the robot. The caster wheel will be modeled as a sphere with a radius of 0.05 meters and without any friction.

The center of the spherical caster wheel will be situated at x = -0.15 meters, y = 0 meters, and z = -0.05  meters in relation to the geometric center of the white box (i.e. the robot chassis).

After creating the sample caster wheel which is placed at the rear end of the vehicle.

A picture containing text, electronics, display, screenshot

Description automatically generated

We need to model the left wheel. We can model this as a cylinder by adjusting the required dimensions according to our requirements.

A picture containing engineering drawing

Description automatically generated

Similar to this we can model the right wheel. We can model this using a cylinder too with the same dimensions.

A picture containing text

Description automatically generated

**Movement of the Rover:**

We can change the state of the rover from true to false. Then you can add a revolute joint to make the rover move. The name of the child is linked to this joint. The parent link is defined. defines the child link's offset from the origin in the frame of the child link.

       The axis of the joint is defined by the parent model frame. This axis is the rotational axis. The two joints in this scenario revolve around the y axis. The normalised axis vector's x, y, and z components are defined. On the right side, the tab increases the force on the rover to move the rover.

A basic Use Case diagram containing your Actors and your main function. Create a short description for your Use Cases (See slide 9)

**Diagram

Description automatically generated**

**Use-case Description:**

Decider, Controller, and receiver are the three actors.

The Controller sets the initial and final coordinates of the rover.

The decider traverses through the maze by detecting the obstacles, generates the map of the maze using SLAM and navigation tools, and sends the map data to the receiver.

The receiver maintains the data received from the decider.

**Commands to start autonomous Rover:**

Please clone the project from the main Branch to the Desktop

**Edit the .bashrc file**

Add the below line to the .bashrc:

**source ~/Desktop/SER515-Spring22-Team-5/project/install/setup.bash**

**Build the colcon packages in the workspace using the below command**

Go to the ~ /Desktop/SER515-Spring22-Team-5/project folder:

Build the package

**colcon build**

**Launch the gazebo world by giving the following command in a new terminal**

**python3 roverGUI.py**

**Launch the controller package in the new terminal by giving the following command**

**ros2 launch maze\_rover\_controller\_pkg controller\_estimator.launch.py**

The rover gets launched in the gazebo and rviz simultaneously in the ros2 environment.

**Lidar**

The Lidar sensor is mounted in front of the rover and the sensor allows the rover to detect the obstacles in front of it.’

**Obstacle avoidance**

The Wall avoidance algorithm allows the rover to avoid the obstacles which are detected by the Lidar sensor.

The Rover avoids any obstacles in its path and will be able to come out of the maze.

A screenshot of a computer

Description automatically generated with medium confidence

**Slam**

The slam generates the map data of its path and sends the map data to the receiver.

A screenshot of a computer

Description automatically generated with medium confidence

**Camera**

TheCamera is integrated onto the top of the rover. The Camera is able to detect the obstacles in front of it.

**Modularity**

If input 2 is given the rover with two wheels and a castor wheel is launched into the gazebo.

If input 4 is given the rover with four wheels is launched into the gazebo.

If input 4 is given the rover with four wheels is launched into the gazebo.

If input 4 is given the rover with four wheels is launched into the gazebo.

It even allows us to select random mazes for the rover from the dropdown

Graphical user interface, text, application

Description automatically generated

Graphical user interface

Description automatically generated

**What we learned:**

We learned how to design a maze and rover in a gazebo. We used python to make the rover avoid obstacles. We have understood the Gazebo and ROS2 functionalities. We made the rover, detect the obstacles in a maze and avoid them.

We were able to capture the images using the camera. We were able to integrate various sensors into the rover. We are able to generate a map of the rover path using SLAM.

**Summary and Conclusion:**

We were able to make the rover avoid obstacles and successfully traverse the maze. We would like to retrieve the data from the rover and store it for future purposes. This project is cost-effective since it was developed in a simulation environment.

**Feature Enhancement:**

In the future, we would like to integrate slam packages into the existing project and generate the map.