IJAREEIE

e-ISSN: 2278-8875
p-ISSN: 2320-3765

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

Volume 10, Issue 6, June 2021

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

Impact Factor: 7.282

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering (IJAREEIE)

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 7.282|

|| Volume 10, Issue 6, June 2021 ||

| DOI:10.15662/IJAREEIE.2021.1006047 |

Autonomous Mobile Robot for Warehouse Navigation and Path Planning Using ROS and Gazebo Simulator

Dr Vaijayanti Deshpande[1], Mohit Sawwalakhe[2], Prajoth Shetty[3], Yash Sawant[4]

Professor, Department of Mechatronics Engineering, Symbiosis Skills and Professional University, Pune, Maharashtra, India.[1]

Student, B.Tech, Department of Mechatronics Engineering, Symbiosis Skills and Professional University, Pune, Maharashtra, India[2,3,4]

ABSTRACT: This paper proposes the idea to study and simulate a robot for the application of autonomous floor & inventory management respectively. The project consists of a Multi wheeled mobile bot, equipped with an array of sensors that will be used for automated path planning and perception in a partially controlled environment. In other words, we need to determine a collision-free path for a robot between start and end positions, so that it can navigate through obstacles cluttered in a workspace respectively. The main motive of autonomous mobile robots is to emphasize path-planning using the Simultaneous Localization and Mapping (SLAM) algorithm that would produce an optimal path for a robot to navigate in an environment. We use a combination of Robotic Operating System (ROS) & Gazebo simulation with effective use of different ROS packages and their integration for making the Bot (simulated model). We use the Amcl, Gmapping ROS-package for sensor fusion, Navigation, and path planning.

KEYWORDS: Autonomous Mobile Robot, Simultaneous Localization and Mapping, Robotic operating system (ROS), Gazebo, Simulation, Inventory.

I. INTRODUCTION

a) Autonomous Mobile Robot:

Autonomous Mobile Robot (AMR) is any robot that can understand and move in its environment without the direct supervision of an operator or on a predetermined fixed path. It helps them perform their tasks in the most effective way and path, avoiding fixed obstacles (buildings, shelves, workstations, etc.) and variable obstacles (such as people, forklifts, and debris). Autonomous mobile robots find the most efficient route to achieve each task and are designed to work collaboratively with operators such as picking and sorting operations, whereas AGVs typically do not. These robots cannot always be programmed to execute predefined actions respectively. [1]

Benefits of AMR [3]-

i. Warehouse Management System (WMS), Order Management System (OMS), Transportation Management System (TMS), and Yard Management Software (YMS).
ii. Remote monitoring is available around the clock.
iii. Systems that are simple to configure.
iv. Zones with automatic charging.
v. Robotic payloads ranging from 250 to 1500 kg.

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 7.282|

|| Volume 10, Issue 6, June 2021 ||

| DOI:10.15662/IJAREEIE.2021.1006047 |

b) Robotic Operating System:

ROS is an open-source, meta-operating system for developing robotics software. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management, etc. The framework is formulated as a collection of small programs that can rapidly pass messages to one another throughout the grid.

This model was chosen to encourage the reuse & rapid prototyping of robotics software outside the particular robot and environment that drove its creation. It also provides tools and libraries for obtaining, building, writing, and running code across diversified computer networks. Being an Open-Source commodity, the creation of generic modules that are applicable to broad classes of robot hardware and software pipelines, facilitating code sharing and reuse, has worked well with the huge robotics community worldwide [2].

c)ROS-Gazebo:

Gazebo is a simulator of multiple robots for an outdoor environment. Step by step, you can simulate robots, sensors and groups of objects, but it is in the continuation world. It produces both comments from realistic sensors and physically all the interaction between objects (includes the exact simulation of rigid physics). Gazebo presents a standard player interface in addition to its own native interface. The written controller for the stage simulator can usually be used from a point of view without any change (and secondary scanning).

The main objective is to work precisely with the robot and its particulars, so ROS entails the description of the kinematics of the robot. Thus, trajectories/path planning, navigation, and more can be planned and executed. The ROS approach is to describe a robot by specifying its properties in URDF (Universal Robot Description Format) files. URDF supports XML and xacro (XML macro) languages. Xacro code is used because of ease in implementation, maintenance and has better readability altogether. To use a URDF file in Gazebo, some additional simulation-specific markers must be added for better functionality with Gazebo. [4]

II. METHODOLOGY

Our approach to studying AMR application in floor management is by seeking the specific needs to simulate and understand a mobile bot by determining the ROS-gazebo framework as an environment & understanding the implementation of the ROS packages used for our distinct application. We also cater to different ROS-based packages and features used in observing the simulation results.

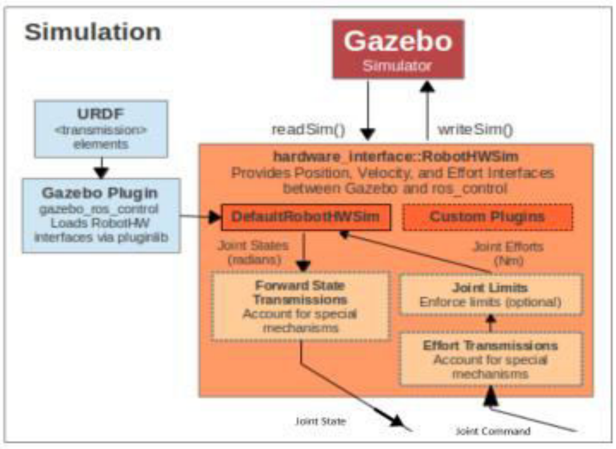
Simulation
Gazebo
Simulator
URDF
<transmission> elements
Gazebo Plugin
gazebo_ros_control
Loads RobotHW interfaces via pluginlib
readSim()
writeSim()
hardware_interface::RobotHWSim
Provides Position, Velocity, and Effort Interfaces between Gazebo and ros_control
DefaultRobotHWSim
Custom Plugins
Joint States (radians)
Joint Efforts (Nm)
Forward State Transmissions
Account for special mechanisms
Joint Limits
Enforce limits (optional)
Effort Transmissions
Account for special mechanisms
Joint State
Joint Command

Fig .1. Framework of Robot Simulation .Gazebo.

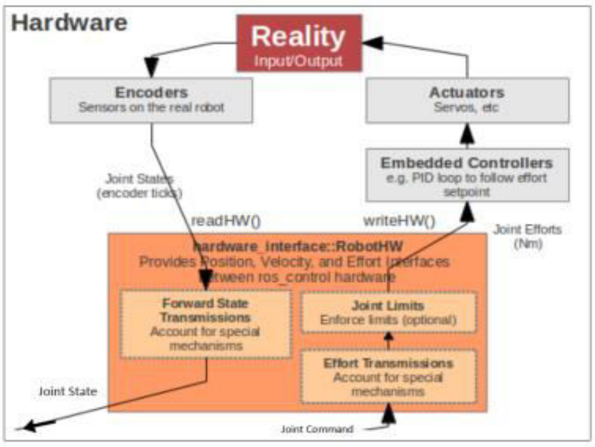
Hardware
Reality
Input/Output
Encoders
Sensors on the real robot
Actuators
Servos, etc
Embedded Controllers
e.g. PID loop to follow effort setpoint
Joint States (encoder ticks)
readHW()
writeHW()
Joint Efforts (Nm)
hardware_interface::RobotHW
Provides Position, Velocity, and Effort interfaces between ros_control hardware
Forward State Transmissions
Account for special mechanisms
Joint Limits
Enforce limits (optional)
Effort Transmissions
Account for special mechanisms
Joint State
Joint Command

Fig. 2. Framework of Robot Hardware

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 7.282|

|| Volume 10, Issue 6, June 2021 ||

| DOI:10.15662/IJAREEIE.2021.1006047 |

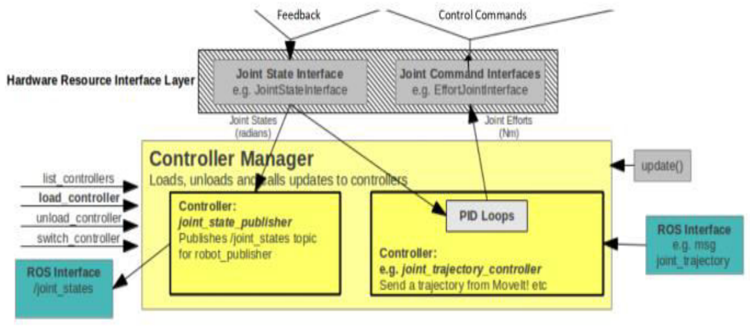

Fig.3. ROS Controller Manager

First, understand the level of the ROS file system and ROS computational graphs, understand the sensor inferences, and later in ROS to get the necessary SLAM (Simultaneous Localization and Mapping) and optional simulation results. Understand the packages used in the Gazebo environment. The structure of the

Gazebo simulation model is shown in Figure 1. It contains robot models and plugins and the Gazebo library. The model accepts data from the Joint Command Interface and retransmits feedback data through the Joint State Interface. The same applies to the hardware model. 2) Connect to the ROS controller using the same interface. The hardware model consists of elements such as embedded controllers, actuators, sensors, etc. which are replaced with components simulated in Gazebo through plugins and libraries. Figure 3 shows the ROS controller. It can process any model and type of data and send control commands accordingly. [10]

1. ROS File System & Computation Graph Level.

i) ROS File System: -The filesystem-level concepts we will mainly cover ROS resources that you encounter on disk, such as:

- i. *Packages*: Packages are the main unit for organizing software in ROS. A package may contain ROS runtime processes (*nodes*), a ROS-dependent library, datasets, configuration files, or anything else that is usefully organized together.
- ii. *Metapackages*: Metapackages are specialized Packages that only serve to represent a group of related other packages.
- iii. *Package Manifests*: Manifests (package.xml) provide metadata about a package, including its name, version, description, license information, dependencies, and other meta information like exported packages.
- iv. *Message(msg)types*: Message descriptions, stored in my_package/msg/MyMessageType.msg, define the data structures for messages sent in ROS.
- v. *Service(srv)types:* Service descriptions, stored in my_package/srv/MyServiceType.srv, define the request and response data structures forservice.[12]

ii)ROS Computation Graph Level:

The Computation Graph is the peer-to-peer network of ROS processes that are processing data together. The basic Computation Graph concepts of ROS are nodes, Master, Parameter Server, messages, services, topics, and bags, all of which provide data to the Graph in different ways.

- i. *Nodes*: Nodes are processes that perform computation. ROS is designed to be modular at a fine-grained scale; a robot control system usually comprises many nodes.
- ii. *Master:* The ROS Master provides name registration and lookup to the rest of the Computation Graph.
- iii. Parameter Server: The Parameter Server allows data to be stored by key in a central location. It is currently part of the Master.

iv. *Messages:* Nodes communicate with each other by passing messages. A message is simply a data structure, comprising typed fields.
v. Topics: Messages are routed via a transport system with publish/subscribe semantics.
vi. Services: The publish/subscribe model is a very flexible communication paradigm, but its many-to-many, one-way transport is not appropriate for request/reply interactions, which are often required in a distributed system.
vii. Bags: Bags are a format for saving and playing back ROS message data.

2.Packages Used in ROS Environment

i) Gazeb_ros_package-The gazebo ros pkgs package is a series of ROS packages that offer the interfaces needed to simulate a robot in the Gazebo 3D rigid body simulator for robots. It uses ROS messaging, services, and dynamic reconfiguration to interface with the platform. The following are some of the characteristics of gazebo ros pkgs: Make use of existing ROS plugins for Gazebo. Improves out-of-the-box compatibility for ros control controllers. There are numerous simulated sensors that are ready to use. [14]

ii) AMCL-package -Amcl is a two-dimensional probabilistic localization system for a robot.It uses a particle filter to track a robot's pose against a known map, as described by Dieter Fox's adaptive (or KLD-sampling) Monte Carlo localization approach. [14]

iii)Rviz- package -Rviz is a ROS 3D visualization tool that allows you to visualize the environment and create visuals for your robot. [14]

iv)Gmap-package -This package contains a ROS wrapper for OpenSlam's G-mapping. The g-mapping package provides laser-based SLAM (Simultaneous Localization and Mapping), as a ROS node called slam_gmapping. Using slam_gmapping, you can create a 2-D occupancy grid map (like a building floorplan) from the laser and pose data collected by a mobile robot.[14]

v) AWS- robomaker :AWS RoboMaker includes a sample robot application to get you started right away. The sample application contains robot application code and simulation application code. The sample simulation application includes pre-built worlds, including rooms in the room, retail stores, and resources needed to create user experiences.[11]

4. ROS Navigation Stack

The Navigation Stack is pretty straightforward on a surface level. It collects information from odometry and sensor streams, outputs speed commands and sends them to the mobile base The utilization of the Navigation Stack on any random robot, however, is a bit more complex at the grass-root level. The only essential condition is that the robot must be running on ROS interface should have a tf transform tree in place, and publish sensor data using the correct ROS message type. Further Navigation-Stack has to be designed for metrics such as the shape & dynamics for appropriate functioning of the bot.

Once we have made a simulation model of the robot, we can configure it to operate at autonomous navigation. Several new steps should be taken to achieve this. These steps are determined by the navigation package requirements.

The principle of operation of the navigation package is shown in Figure 4. The autonomous navigation is carried out in the move_base node. [9] This node receives the desired target data from the move_base_simple / target topic. Then move_base reads the odometry, LaserScan, tf-transform library and GetMap data, processes the data and plans the route. Finally, the commands required to control the mobile platform are published in the cmd_vel topic.

The mapping and localization packages [6] must be configured so that the navigation works properly. These two packages are part of the packages in the ROS and can be used immediately. The mapping package is used to create and retrieve map information [7]. The AMCL package offers auto-location algorithms [8]. In addition, some local and global scheduler parameters as well as the global_costmap and local_costmap parameters can be configured. If the entire navigation system is already configured, let's move on to the experiments. A map of the working environment should be drawn up. The map creation process is done manually. The following programs and nodes are started:

- Start the robot model in the simulated world;
- slam_gamapping node, which reads and returns odometry and laser scanner data to create a new card;
- TF node [5] • Node for manual control of the robot, via keyboard or joystick;
- rviz interface for rendering the map created;

After the map is ready, we will save it.
When the map is ready we can start the autonomous navigation mode.
for this we need to start the following programs:
• start the robot model in the simulated world;
• the nodes in Figure 6: map_server, tf, amcl, move_base;
•rviz interface fordisplaying and settingthe desired destinations.
So now we have a functional simulation model of the robot under the control of ROS in autonomous navigation mode. You can open rqt_graph to make sure you have all the nodes you need. An interface opens with all running nodes and topics (Fig. 8). This check is very useful as we can easily determine whether the connections between the nodes are correct and whether any connections are missing.

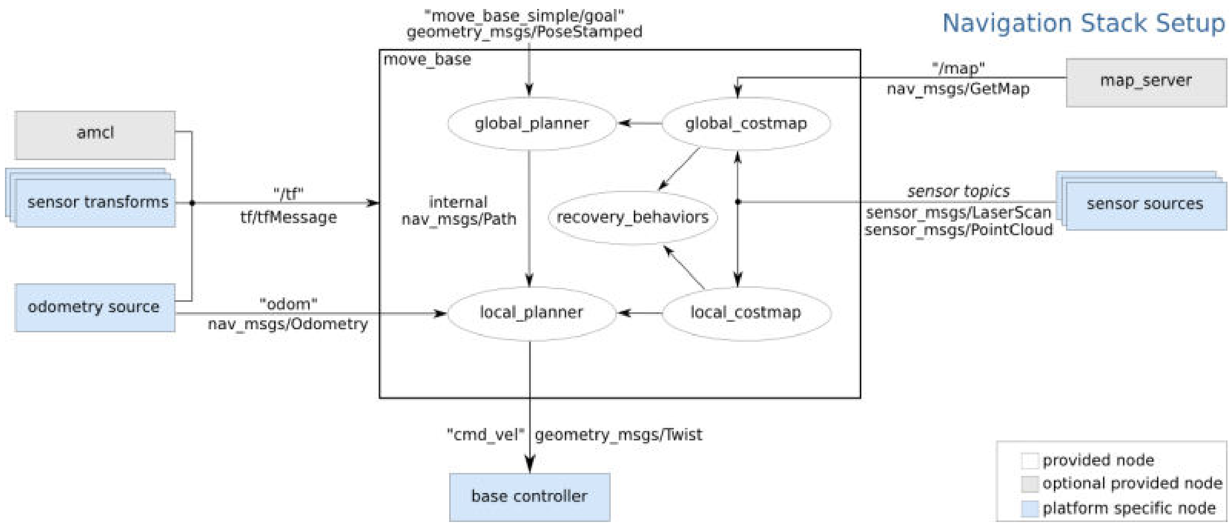

Fig.4. ROS Navigation Stack Configuration

III. SIMULATION ENVIRONMENT AND EXPERIMENTAL RESULTS.

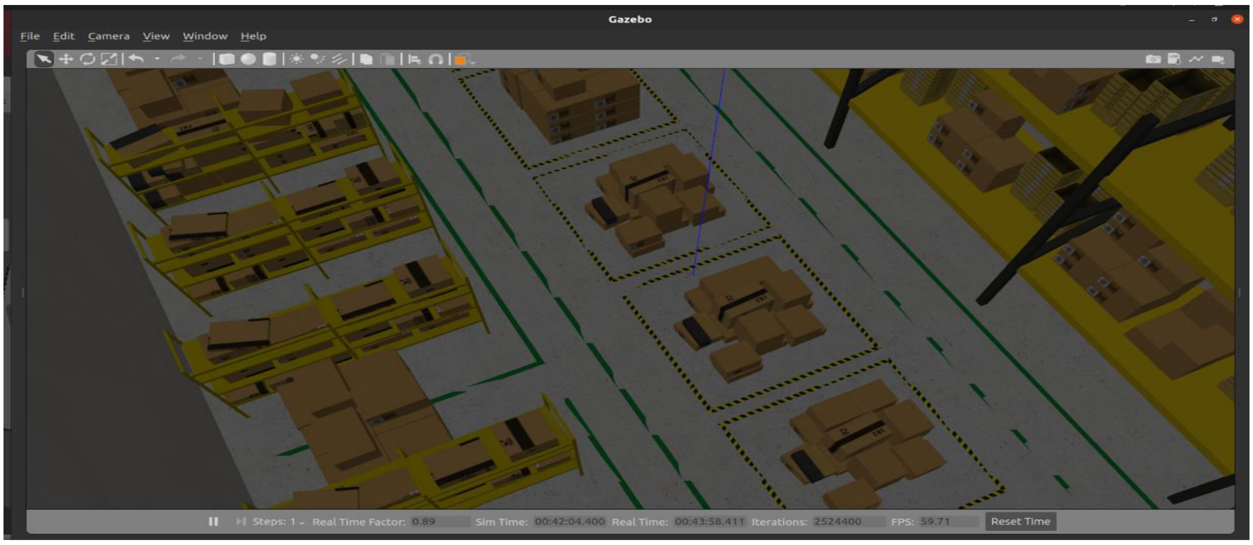

Fig. 5 Gazebo Environment

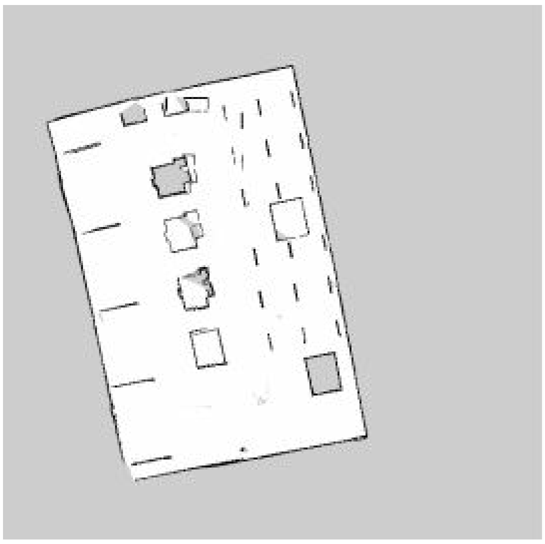

Fig .6. Created Map of the Warehouse Gazebo World

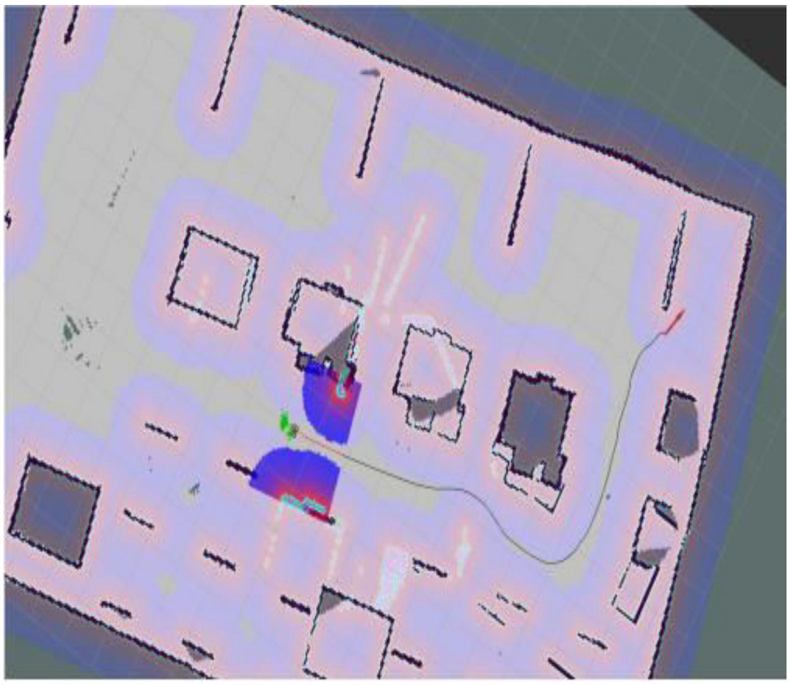

Fig.7. First Simulation of Autonomous Path Planning.

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 7.282|

|| Volume 10, Issue 6, June 2021 ||

| DOI:10.15662/IJAREEIE.2021.1006047 |

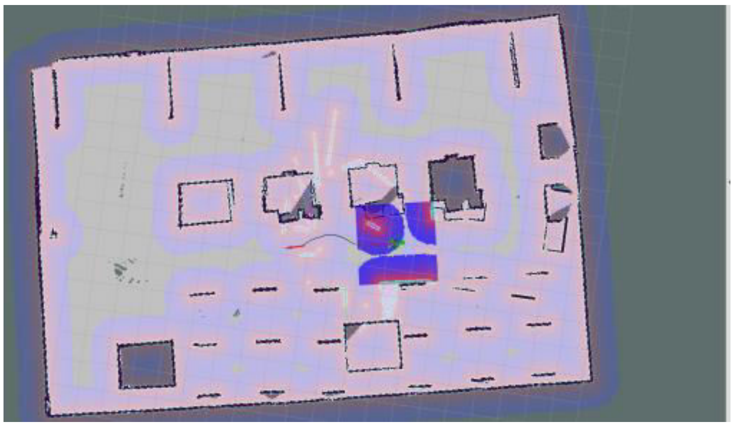
Fig.8. Second Simulation of Autonomous Path

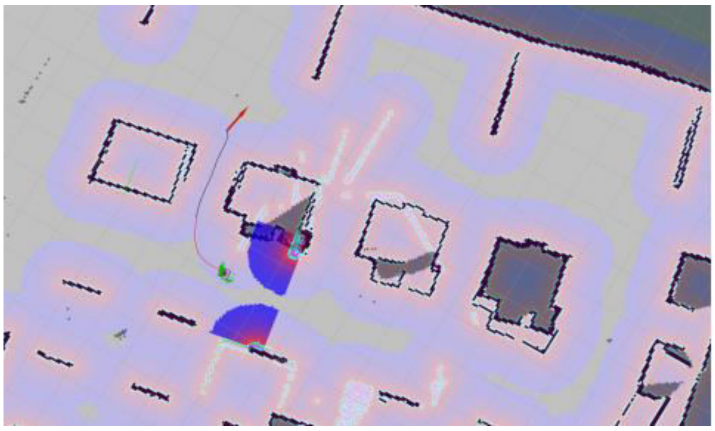
Fig.9. Third Simulation of Autonomous Path Planning.

Because only the bare necessities for operating a robot in autonomous navigation mode are met, the robot cannot be expected to move perfectly and always achieve the set coordinates. Experiments can, however, be carried out to ensure that the robot is capable of performing the tasks. There are two types of experiments carried out. The first type of experiment determines whether or not the robot can move independently in a room devoid of obstacles. In the second type of experiment, additional objects were placed in the room, and we tested the robot's ability to detect and avoid obstacles. Both types of experiments yielded positive results. All experiments are successful when the robot is navigated in an empty room with no obstacles added. The system successfully locates the robot, creates a path for it, and navigates it. Every time, the robot arrives at the desired position. Because there is so much open space, the navigation system has no trouble. This, however, changes when additional obstacles are introduced. In these experiments, the navigation system recognizes obstacles and, if necessary, changes the planned path of the robot while it is moving. Figures 7, 8 and 9 depict the steps involved in completing one of the tasks. Because only the essentials for operating a robot in autonomous navigation mode are met, the robot cannot be expected to move perfectly and always achieve the set coordinates.

IV. CONCLUSION AND FUTURE SCOPE

In this paper, we have described an efficient approach that describes preliminary results w.r.t to the application of autonomous mobile robots in smart warehouses with the integration of ROS & SLAM with their prescribed set of algorithms respectively.

This work contributes to the development of a smart warehouse with the incorporation of autonomous mobile bots to uphold the existing supply chain logistics structure as these bots can perform several warehouse and order fulfilment functions, including transporting goods and materials and guiding/helping associates as they perform their tasks. It increases efficiency by opening up floor space allowing for more machines and therefore, moving product through processing at a faster pace. The potential induction of AMRs in the traditional setting will aid in making automation easy to integrate due to ease of deployment, henceforth augmenting & empowering the human labor with flexible capital expenses.

REFERENCES AND CITATIONS

[1] Autonomous Mobile Robot - https://www.conveyco.com/types-and-applications-of-amrs/
[2] Robot Operating System (ROS)https://wiki.ros.org/ROS/Introduction
[3]Benefits of AMR https://om.gen-ural.ru/540
[4]ietrzik, S., and B. Chandrasekaran. "Setting up and Using ROSKinetic and Gazebo for Educational
Robotic Projects and Learning." In Journal of Physics: Conference Series, vol. 1207, no. 1, p. 012019. IOP Publishing, 2019.

| e-ISSN: 2278 – 8875, p-ISSN: 2320 – 3765| www.ijareeie.com | Impact Factor: 7.282|

|| Volume 10, Issue 6, June 2021 ||

| DOI:10.15662/IJAREEIE.2021.1006047 |

[5] Foote, Tully. "tf: The transform library." In 2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA), pp. 1-6. IEEE, 2013.
[6] Guimarães, Rodrigo Longhi, André Schneider de Oliveira, João Alberto Fabro, Thiago Becker, and Vinícius Amilgar Brenner. "ROS navigation: Concepts and tutorial." In Robot Operating System (ROS), pp. 121-160. Springer, Cham, 2016.
[7] da Silva, Bruno MF, Rodrigo S. Xavier, and Luiz MG Gonçalves. "Mapping and Navigation for Indoor Robots under ROS: An Experimental Analysis." (2019).
[8]Talwar, Dhruv, and Seul Jung. "Particle Filter-based Localization of a Mobile Robot by Using a Single Lidar Sensor under SLAM in ROS Environment." 제어로봇시스템학회국제학술대회논문집 (2019): 1112-1115.
[9] ROS Navigation: http://wiki.ros.org/navigation/Tutorials/RobotSetup?action=AttachFile&do=get&target=overview_tf.png
[10] Gazebo ROS Control: http://gazebosim.org/tutorials?tut=ros_control&cat=connect_ros
[11] AWS Robomaker website -https://aws.amazon.com/robomaker/
[12] GitHub -https://github.com/aws-robotics
[13] Ros file system and computational: http://wiki.ros.org/ROS/Concepts.
[14] ROS Packages:
Amcl- http://wiki.ros.org/amcl
Mapserver- http://wiki.ros.org/map_server
Rviz- http://wiki.ros.org/rviz
Gmap- http://wiki.ros.org/gmapping