Optimal Path Planning for Autonomous Collaborative Delivery Robots with Dynamic Obstacle Avoidance

Akshay Laddha
Robotics Engineering Department
Worcester Polytechnic Institute
Worcester, MA, USA
aladdha@wpi.edu

Anagha Ramaswamy
Robotics Engineering Department
Worcester Polytechnic Institute
Worcester, MA, USA
aramaswamy@wpi.edu

Kunal Nandanwar
Robotics Engineering Department
Worcester Polytechnic Institute
Worcester, MA, USA
kgnandanwar@wpi.edu

***Abstract*—Autonomous delivery robots have gained massive importance in the last decade and have been successfully been implemented by several companies, and science-fiction movies. The delivery task were automated with the purpose to generate labor cost savings by reducing human labor along with an increase in accuracy.**

In this paper, we will be implementing motion planning algorithm on a group of autonomous mobile robots for food delivery.

I. Introduction

In retrospect, a robot delivering a particular product was thought of as an entertainment product in science-fiction movies. The German assessment of robotics and automation focused on the labor market where the delivery task were automated with the purpose to generate labor cost savings by reducing human labor along with an increase in accuracy. In his book, Flamig talks about the the goods delivery and freight industry automation indicating that small delivery robots are capable of mapping the urban centers but may still require some sort of human involvement from the receiver side. Discussing the potential of Autonomous delivery robots, reduction in the fuel use and emission is considered to be a strong case for bulky cargo vehicles.

Currently, due to an exponential rise in technology over the years, automation has grown beyond production of generalized industrial products to engaging its value in the service sector. Autonomous delivery robots have become the talk of the town and are on the path of becoming a reality. The advantages of Delivery robots range from cheaper delivery cost, rapid customer service to personnel safety and the correctness of delivering the right package to the relevant person. These autonomous delivery robots are stuffed with obstacle avoidance and range detection sensors and are made to map the environment with the aid of several machine vision algorithms and radar technology.

II. Background

B2B and product delivery are booming, and several start-ups have already begun pilot projects to deliver goods and groceries. A head-start in this area was given by Starship Technologies, an Estonian company who collaborated with Co-op and Tesla to begin its autonomous delivery service in the United Kingdom and consequently, they became the first and largest robot delivery service in British Town center. Beginning Janaury of 2019, Starship Technologies partnered with Sodexo to launch its robot food delivery services which happened to be the largest implementation of autonomous robot food delivery services on a university campus. Figure 1 shows the fleet of the Starship technologies robots [2].

Fig. 1: **Starship Delivery Robots:** Fleet of Starship technologies robots [1]

Starship Technologies food delivery robots are electric powered attaining a maximum speed of 6 kmph and are preferred for shorter range deliveries. If the autonomous operation fails, the robot can be remote controlled. To understand and navigate the terrain, the delivery robots make use of edge detection and mapping techniques. In terms of sensor suite, the delivery robot consist of a camera, an IMU (Accelerometer, Magnetometer and Gyroscope), GPS in combination with a radar technology.

Following this, big tech companies like Amazon, Nuro, AutoX to name a few have joined this revolutionary movement of delivery bot. With its electric powered Scout Autonomous delivery Vehicle, Amazon has began delivering packages to Amazon Prime customers at the Amazon HQ of Washington.

Fig. 2: **Nuro Delivery Robot:** Driverless bot from Nuro [1]

Nuro has gone one step ahead by going with the open-road route instead of the sidewalk route. Named as the R2, the driverless vehicle places fresh food in high temperature sections with no conventional controls. Figure 2 shows the driverless bot delivery from Nuro which has received a green light to operate [2].

In this paper, we aim to focus and delve deeper into this domain by exploring relevant algorithms and implementing them on autonomous swarm robots with evasive movement capabilities.

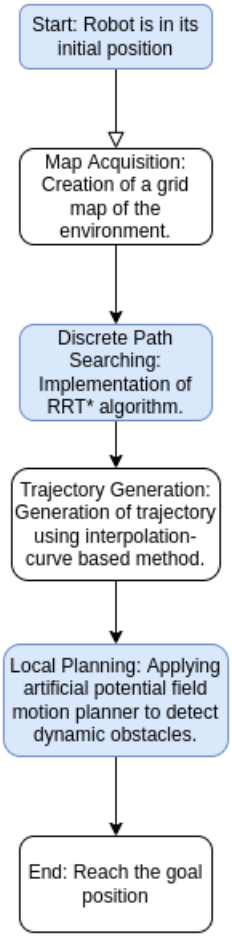

Fig. 3: **Methodology**: Step by step implementation of the project

III. Proposed Methods

To develop a system of autonomous mobile robots, the most fundamental and essential aspect is that motion planning. The architecture of motion planning methods can be divided into four parts: map acquisition, discrete path searching, trajectory generation and local planning. Figure 3 summarizes our methodology of our project.

A. Map Acquisition

The map representation of the environment is the initial step required to plan a motion planning algorithm. A few common maps include OctoMap, Point Cloud Map and Grid Map [4]. We will be implementing our algorithm on a 2D space initially, by using a grid map, and later try to apply the same algorithm in a 3D environment. For the 3D map, we plan to use a simulating software that supports a swarm of robots, such as ARGoS or Gazebo.

B. Discrete Path Searching

The next crucial step is to find a suitable path from the initial to the final point. From a broad perspective, the Discrete Path Searching can be divided into two sub-parts.

The first category is that of Graph-search-based algorithms. The map can be considered as a graph for the implementation of the algorithms. They are applied mainly on low dimensional spaces. Few examples include:

- Depth First Search (DFS)
- Breadth First Search (BFS)
- Dijsktra's algorithm
- A* algorithm and its variants

Algorithm 1 The RRT* Algorithm

$V \leftarrow \{x_{init}\}$; $E \leftarrow \varnothing$; $T \leftarrow (V, E)$;
for $i = 1 \rightarrow N$ **do**
  $x_{rand} \leftarrow Sample(i)$;
  $X_{near} \leftarrow Near(V, x_{rand})$;
  if $X_{near} = \varnothing$ **then**
    $X_{near} \leftarrow Nearest(V, x_{rand})$;
  end if
  $x_{parent} \leftarrow FindBestParent(X_{near}, x_{rand})$;
  if $x_{parent} \neq$ NULL **then**
    $V \leftarrow V \cup \{x_{rand}\}$; $E \leftarrow E \cup \{(x_{parent}, x_{rand})\}$;
    $E \leftarrow Rewire(E, X_{near}, x_{rand})$;
  end if
end for
return $T = (V, E)$;

Fig. 4: **RRT* Algorithm**: Pseudocode of the RRT* algorithm [3]

The next category is that of Sampling-based algorithms. These are more suitable for path planning scenarios that involve high dimensional spaces. They have probabilistic completeness. The two fundamental sampling-based algorithms are:

- Probabilistic Road Map
- Rapid-exploring random tree (RRT)

TABLE I: **Timeline and Division of Tasks:** The table shows the tentative deadlines for each task

From	***To***	***Task***	***Responsible***
-	Jan 31	Project Proposal	All
Feb 1	Feb 14	Setting up the environment on Pygame	Anagha
Feb 1	Feb 14	Setting up the environment on Carla	Kunal
Feb 1	Feb 14	Setting up the environment on V-REP	Akshay
Feb 1	Feb 14	Setting up the environment on Gazebo	Akshay, Anagha
Feb 15	Feb 18	Robot modeling for Gazebo and V-REP	All
Feb 19	Feb 21	Project Status Update	All
Feb 22	Feb 28	Finalize the Environment	All
Mar 1	Mar 10	Implement the Motion Planning Algorithm on Single Robot	Kunal
Mar 11	Mar 18	Static & Dynamic Obstacle Avoidance	Anagha
Mar 19	Mar 28	Implement the Motion Planning Algorithm on a Swarm of Robots	Akshay
Apr 1	Apr 20	Testing and Debugging	All
Apr 21	May 1	Final Project Report	All

RRT planning algorithm creates a graph and finds the path. But the path is not always optimal. Overcoming this limitation of RRT is the RRT* algorithm. Figure 4 demonstrates the pseudocode of the RRT* algorithm.

We will be implementing RRT* algorithm to our delivery robots.

C. Trajectory Generation

A suitable trajectory is generated after considering the geometric constraints of the environment by the path searching algorithms. To optimize further for our delivery robot, a time constraint is added. For this, we will be using the interpolation-curve-based method when implemented on a 3D environment. [5]

D. Local Planning

There could be several uncertainties while deploying the robot that include dynamic obstacles and pedestrians. To overcome these, we will be using a local motion planner - artificial potential field.

IV. Goals

- *Environment mapping*: The robot will be working with a map of the environment that contains the static obstacles.
- *Implementation of the motion planning algorithm*: We will then be applying the algorithm to the robot so that it can traverse from the initial point to the final using an optimal path.
- *Obstacle avoidance*: The robot should also be able to avoid the obstacles on its path. This includes dynamic obstacle avoidance.
- *Implementation on a swarm of robots*: After a single robot is successfully able to perform the tasks, the same will be implemented on a swarm of robots. The simulation of the same will be performed.

V. Timeline

Table I demonstrates the timeline and work distribution for the project.

VI. Tools

There is a lot of simulation-based testing that can help robot deployment on-site faster.

A. Stage

Stage is a lightweight 2.5D simulator with seamless ROS connection, built primarily for wheeled mobile robots. Stage includes ROS APIs for commanding velocities, returning the robot's ground truth stance, and a 2D-LiDAR plugin for simulating laser scans. Stage is an excellent choice for multi-agent systems due to its small number of perception-oriented capabilities and low computing needs.

Stage has limitations in imitating cameras, 3D LiDARs, and non-wheeled robots.

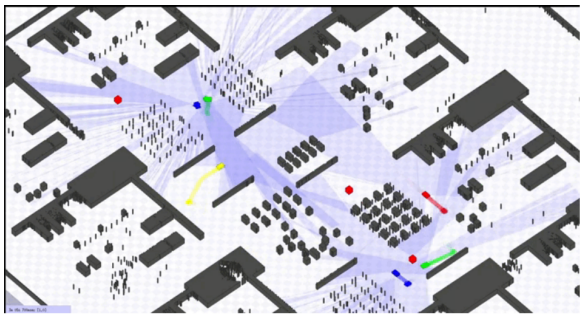

Fig. 5: **Environment in Stage:** Autonomous Mobile Robot Fleet simulation in Stage

B. Gazebo

Gazebo is a 3D physics engine and renderer that can handle nearly all robot kinds, including land, air, and water robots. Gazebo allows users to employ a variety of sensor plugins, including RGB-D cameras, 3D LiDARs, GPS, IMU, and more. In addition to its functionality, Gazebo has a sizable user/developer community and offers simple solutions to the majority of problems one might encounter.

When dealing with non-wheeled outdoor robots, a gazebo can be employed. It also features a more advanced physics

engine than Stage, making it feel more ”real.” The difficulty stems from the increased processing load that comes with adding more robots, as well as the limitations of using camera data—Gazebo worlds aren't a particularly realistic approximation of reality, making it tough to evaluate vision-based algorithms.

C. Unity

In the realm of robotics simulations, Unity has recently acquired prominence. Unity's functional advantages include high-fidelity graphics, a physics engine, and the ability to create any form of robot.

Unity also comes with a developer community and software tools already in place. However, because most Unity programming is focused on gaming, most of the accessible assets (environment, objects, and so on) aren't ready-made representations of real-world scenarios in which a robot would find itself. Because the Unity-Robotics community is still expanding, few robot platform providers have produced official Unity-compatible models of their robots. The biggest difficulty with Unity is that it requires a lot of processing power to get the most out of the visual benefits.

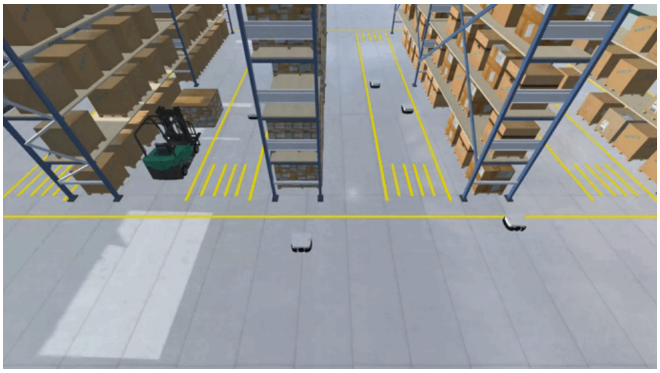

Fig. 6: **Environment in Unity:** Autonomous Mobile Robot Fleet simulation in Unity

D. Others

CARLA and LGSVL are two simulators designed with self-driving automobiles in mind. Off-road applications and inside locations like as factories and warehouses are not well suited to them. AirSim is an intriguing alternative to Unity-based robotics simulation. Webots is a robotics simulator that has been open source for the past four years, making it more accessible than ever before. Another popular robotics simulator worth trying out is CoppeliaSim (previously known as V-Rep).

VII. Technical Set Up

A. Environment Set Up

Since there was a bit of uncertainty in the potential of the various environments to launch multiple robots, we planned to implement the basic environment setup on four different ones. Depending on their capabilities, we will finalize the environment. The task of setting up the environments are described below.

1) CARLA: Post software installation, we implemented an already existing world in CARLA. We imported python files to create traffic and moving vehicles that would work as the dynamic obstacles. We successfully navigated the vehicle in the map. Moreover, we created the aerial view to map the static and dynamic obstacles at the same time giving us the information about the permitted path.

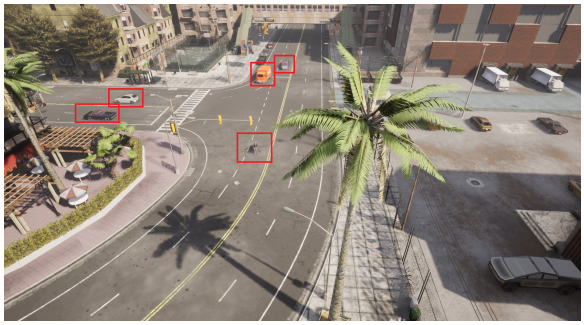

Fig. 7: **CARLA environment with static & dynamic obstacles:** CARLA environment with static and dynamic obstacles

2) V-REP: For the V-REP simulation, a custom world with series of obstacles is created and a Pioneer P3DX robot equipped with laser sensor is placed within the simulation world.

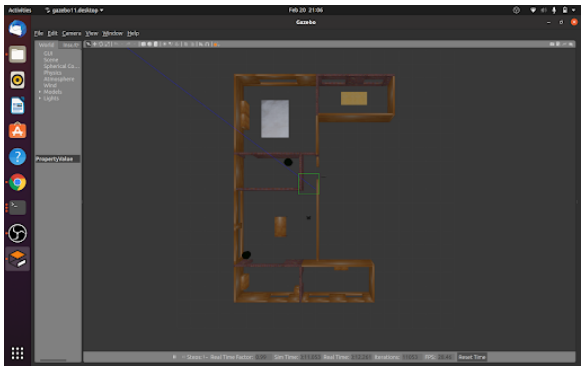

Fig. 8: **Gazebo Environment 1:** The simulation of the existing environment on Gazebo

3) Gazebo: Gazebo was installed on the system and we launched the world. First, we implemented an already existing world on Gazebo. Later, we modified the same to create our own world for our application. We also created a CAD model of our robot and then created a URDF version of the same so as to launch it on Gazebo. An image of the two environments with the robot is shown in figure 8 and figure 9

- Localization - In Gazebo, we plan to create a ROS package that implements an extended Kalman filter which can be further used for sensor fusion. The sensor data that can be fused together derives from the -
  - Inertial Measurement Unit - An IMU generally consist of a 3-DOF gyroscope (stability purposes) and a

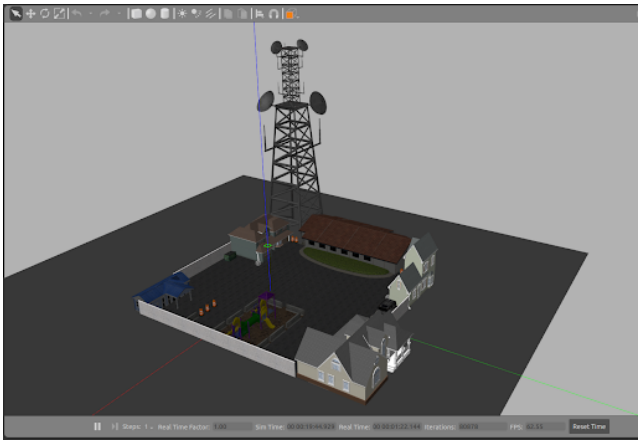

Fig. 9: **Gazebo Environment 2:** The simulation of the environment on Gazebo

3-DOF accelerometer and can sometimes contain a 3-DOF magnetometer allowing to measure velocity and the positioning.

- Rotary Encoders (wheel odometry) - These are attached to the robots actuated wheels in order to measure the position and velocity of the wheels

4) PyGame: The 2D map of the environment was created with the static obstacles. A start point and end point was given to the robot [6]. The image of the environment is shown in figure 10.

B. Mapping and Obstacle Avoidance

For visualizing the environment, we plan to add vision sensors such as a video camera and stereo camera to the robot. Along with that, we also plan to add the 3D scanning and 3D mapping capabilities of a LiDAR for obstacle detection and location determination. The feedback from the LiDAR will be used to adjust the speed and the path. Ultrasonic Sensor and Radar will also be attached on the robot to assist obstacle detection.

C. Preliminary Results

The environment setup was first done simultaneously on PyGame, V-REP and Carla. The implementation of RRT algorithm on a single robot was done successfully on the PyGame environment that had static obstacles as shown figure 11.

Further, we were able to setup our environment on Gazebo and launch our model of the delivery robot as shown in figure 12.

Figure 14 demonstrates CARLA environment with static obstacles only. On the other hand, figure 15 contains static and dynamic obstacles. Dynamic obstacles include humans and moving vehicles and are highlighted in red box for better visualization.

We also tried implementing the aerial view as shown in figure 13. The aerial view shows our vehicle in green marker and the obstacles in blue and orange markers. Also, relevant details(like speed, speed limit etc) is also displayed in the text on the left of the image.

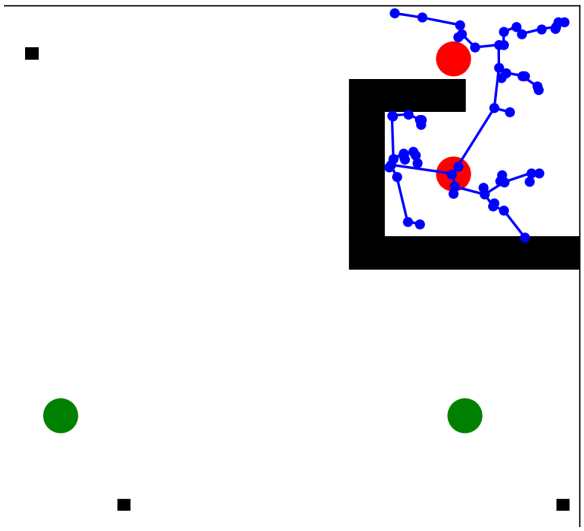

Fig. 10: **PyGame Environment:** The 2D simulation of our environment on PyGame

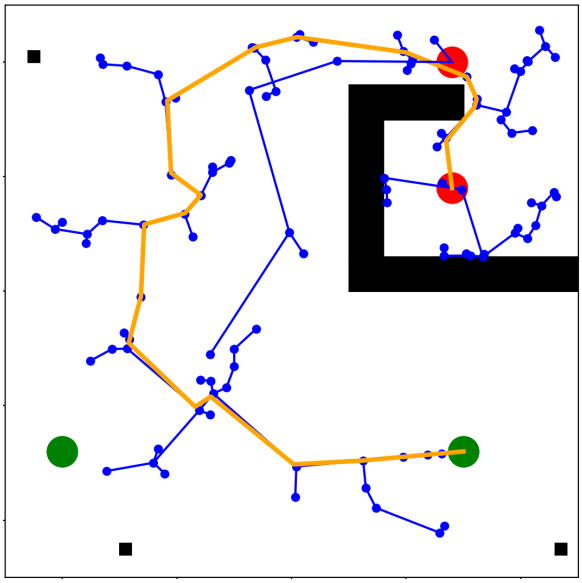

Fig. 11: **Implementation on Pygame:** Implementation of the RRT algorithm on PyGame is shown

On V-REP, we created a custom world and imported the Pioneer 3DX robot in the world. The purpose is to build the map of the environment as the robot moves from room to room and reads data from laser sensor. The laser sensor only returns points if an object is detected indicating that if the object is below the maximum distance associated with this laser sensor, we receive a point or else no information or point is included in the map as shown in figure 16 and 17. While integrating V-REP with ROS, we faced a couple of glitches and moreover with less community support, we have put a hold on VREP.

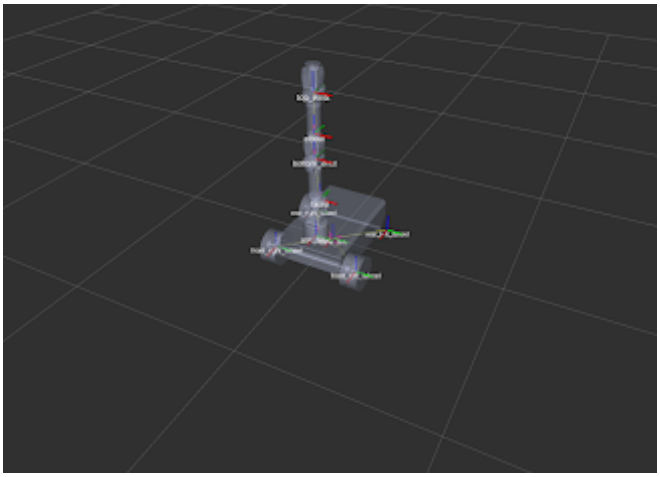

Fig. 12: **Delivery Robot:** Model of delivery robot that we plan to use on Gazebo

VIII. Challenges

During environment setup of V-REP, we faced issues with its compatibility with Python. The software provided the update with python last month and hence there isn't enough documentation to refer to.

The present challenges with Carla is multi-vehicle integration and generation of a 2D grid map. We have successfully implemented the 2D view of the simulation using aerial view. We aim to generate a 2D grid map out of the same. Since Carla requires a lot of RAM, the simulation works seamless and perfect in Windows but a bit sluggish in Ubuntu(due to RAM allotment issues).

Communication between the various robots is also another challenge that we have to address.

IX. Results

A. PyGame

The environment was setup on Python with an obstacle field. It consisted of both static and dynamic obstacles. An Artificial Potential Field was set up on the environment so as to facilitate the robot so as to avoid obstacles. After implementing it on a single robot, the same setup was used to navigate the path with multiple robots.

The start point of the robot is shown as a red dot and the goal is shown as a green dot. The robots perform RRT path planning algorithm, and find the respective paths to the goal position. After path calculation, the artificial potential field is implemented which helped the robots avoid the obstacles as shown in 19. The implementation of dynamic obstacles are also done successfully. The three cubic obstacles shown in the environment are set to move one the robots start moving. Their movement effects the artificial potential field. With the help of both the path planning algorithm and the implementation of artificial potential field, the two robots are able to navigate through the required environment by following the path successfully as shown in 18.

The same setup can be used to navigate multiple robots.

1) Artificial Potential Field: The robot and the obstacles are considered as a positive bodies and the goal is considered a negatively charged body. This way, the goal will always attract the bot and the obstacles will repel the bot. The robot will aim to move to a lower energy configuration. The energy is minimized by following the negative gradient of the potential energy function.

The potential field can be divided into two parts -

- Attractive Potential Field - helps move towards the goal
- Repulsive Potential Field - helps avoid obstacles

The combined potential, that is the sum of both attractive and repulsive potential field, is calculated so as to assign the artificial potential field.

2) Multiple Robot Navigation: Implementation is done by having multiple robot classes in the code. Each robot class corresponds to the features of a robot. The procedure followed by multiple robots is same as that of a single robot system. The only difference being that the other robots will be considered as an obstacle to the robot. It navigates without colliding with the other robots.

B. Single-Robot system in Gazebo

In order to set-up a working environment in Gazebo and be familiarized with the integration of sensors and SLAM procedure, a single-robot delivery architecture was tested.

For the initial stage, a mobile manipulator was modelled by creating the config launch meshes and urdf folders within a ROS package. The meshes folder contained the STL files required to build the robot and were imported in the urdf file to model the mobile robot. The robotic arm was modelled using Rviz with the configuration parameters set up for the controller and the movement of the arm was created using Moveit software by generating a collision matrix,and choosing RRTStar as the group default planner. The movement was created by defining two different robot poses - one in stationary mode where the links are at home and other at goal destination.

Later, a map of the entire world was chalked out by the mobile manipulator using two packages -

1) Hector-SLAM: Hector-SLAM is a Mapping and Localization based algorithm which makes use of hector mapping method which in turn is a SLAM approach that can be used without odometry as well as on platforms that exhibit roll/pitch motion (of the sensor, the platform or both). It leverages the high update rate of modern LIDAR systems like the Hokuyo UTM-30LX and provides 2D pose estimates at scan rate of the sensors (40Hz for the UTM-30LX).

2) Adaptive Monte Carlo Localization: : Adaptive Monte Carlo Localization is a probabilistic localization system for a robot moving in 2D. It implements the adaptive (or KLD-sampling) Monte Carlo localization approach (as described by Dieter Fox), which uses a particle filter to track the pose of a robot against a known map.

The aforementioned packages are included in the ROS Navigation stack. The mapping was performed manually using rqt robot steering package of ROS. The map of the environment

Fig. 13: **Aerial View for corresponding vehicle:** The aerial view shows our vehicle in green marker, obstacles in blue and orange markers and relevant details(like speed, speed limit etc) in the text on the left of the image

Fig. 14: **CARLA environment with static obstacles:** CARLA environment containing static obstacles only

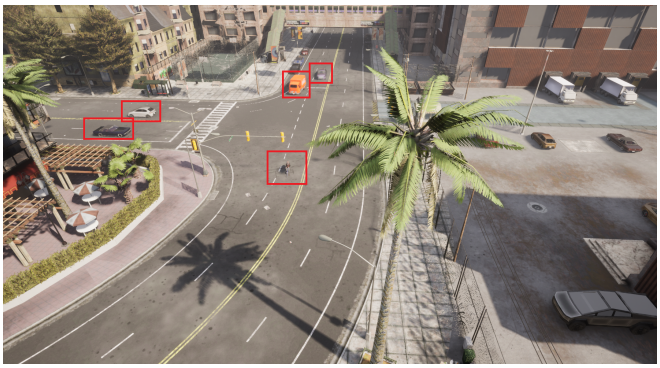

Fig. 15: **CARLA environment with static and dynamic obstacles:** CARLA environment with traffic(humans and moving vehicles marked in red box) that would act as the dynamic obstacles

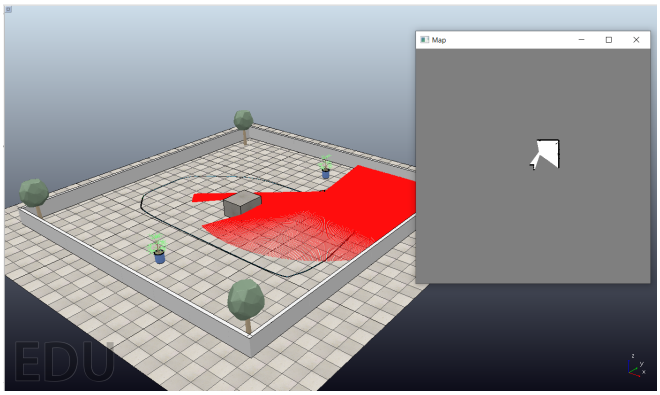

Fig. 16: **V-REP custom environment with static obstacles:** Pioneer 3DX bot using SLAM to map the environment containing static obstacles only and an occupancy grid map being created

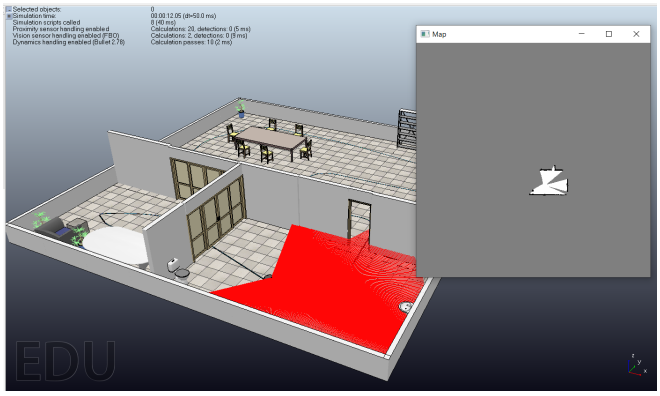

Fig. 17: **V-REP imported environment to perform SLAM:** Pioneer 3DX bot placed in an imported world to perform SLAM

was saved using the map server package from ROS which saves the SLAM performed map data to yaml and pgm format.

The delivery system was implemented in two ways -

1) Prepared a cpp code mentioning the four co-ordinates of interest within the map and based on the user input, the robot

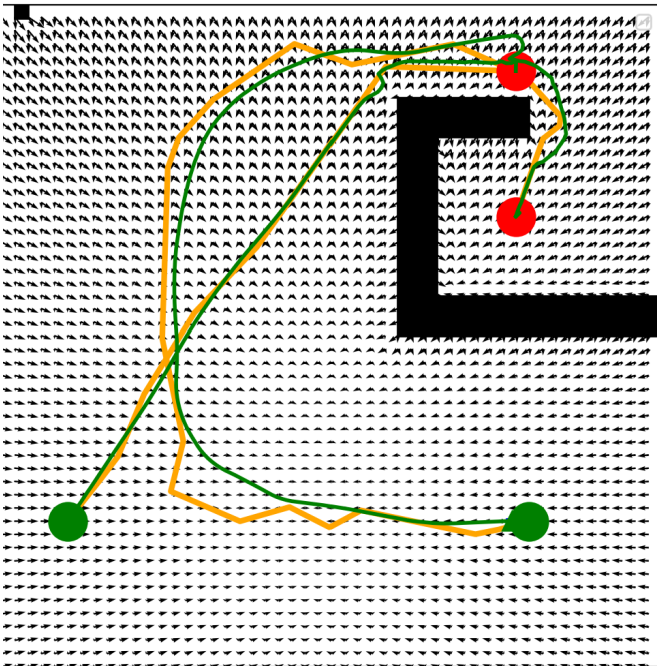

Fig. 18: **Implementation on PyGame:** The two robots reach the goal position by following RRT algorithm and avoiding obstacles

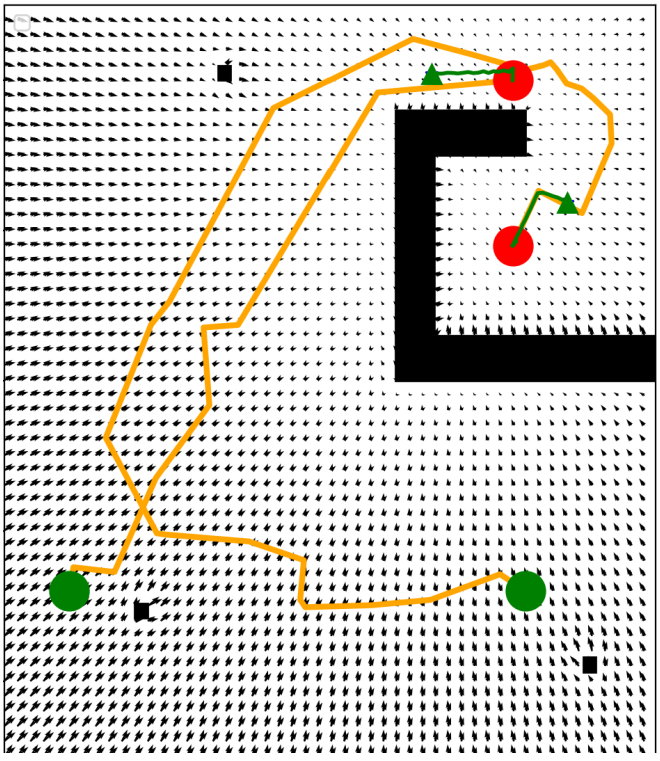

Fig. 19: **Implementation on PyGame:** The two robots are navigating to reach the goal position by following RRT algorithm and avoiding obstacles

will traverse to the location.

2) The second method is utilizing the estimation of robot pose and navigation feature goal method of Rviz and setting a pose estimate within the map somewhere in line with the currently on the map and the goal destination by setting the Nav goal button in Rviz and placing the goal in an empty space on the map

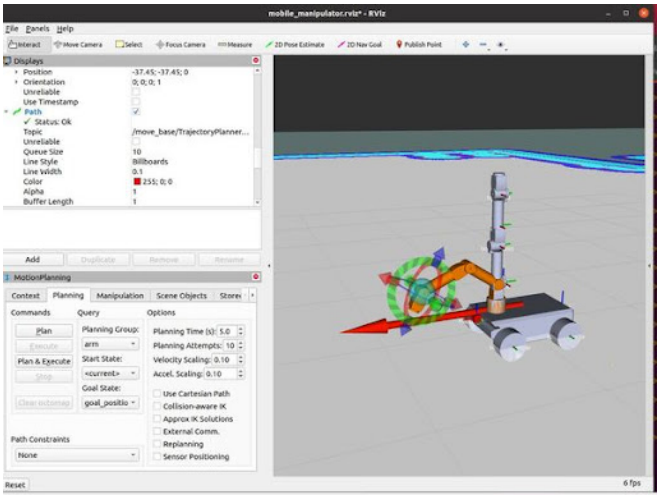

Fig. 20: **Single Robot in Gazebo:** Controlling Robotic Arm using MoveIt & ROS

C. Multi-Robot System in Gazebo

After testing the working of single robot, motion planning algorithm was implemented on multi robot system. Our proposed robotic delivery system consist of a fleet of robots parked at multiple base stations, having unique base ID. Upon receiving the orders from customers, advanced Task Planner algorithms take care of assigning delivery jobs with sufficient charge to fulfill the order. All the required information such as source-destination location parameters are passed to the robot's sensors and path planning algorithms to pick the order and deliver the order to the new destination location.

Upon fulfillment of the order to the destination, The robot's destination location becomes the new source location to process the new order. These operations are further continued and planned according to the task planner.

Localization is responsible for using the robot sensors (i.e., the UGV wheel encoders, IMU data, and GPS measurements) to estimate the position of the robot.

Navigation is responsible for sending the UGV wheel velocity commands which move it to the desired destination.

1) Setting up the Robots: For setting up the robot, urdf files for the robot have been created, different sections in the launch file were made and urdf files were imported. Then the position and orientation values were assigned.

The figure 26 shows the fleet of autonomous robots in an empty gazebo world.

2) SLAM: GMapping algorithm is based on particle filter pairing algorithm, Hector-SLAM is based on scan matching algorithm. Hector map just only can afford indoor use, if you are outdoor, the map is too small, and it cannot expand. Gmapping has no such limit, used indoor and outdoor, but a little Inferior to hector map in efficiency.

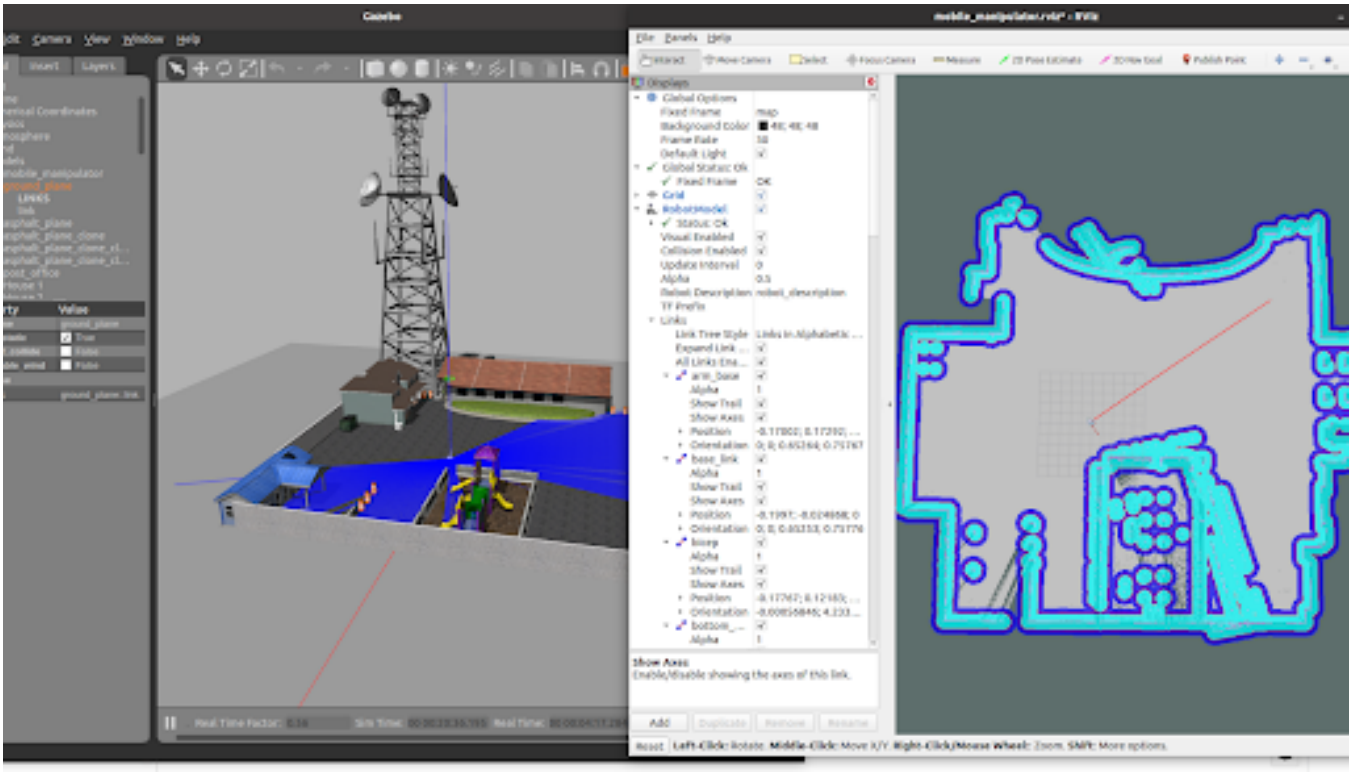

Fig. 21: **Single Robot in Gazebo:**

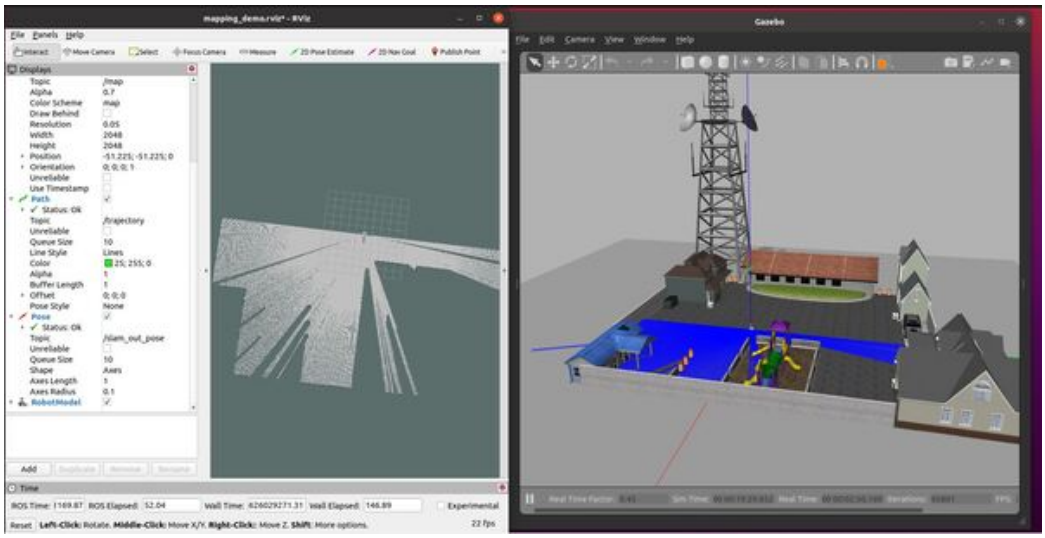

Fig. 22: **Single Robot in Gazebo:** SLAM(a)

Figure 27 shows visualization of gazebo museum world. Figure 28 demonstrates that robot has created the map of the unknown world using Gmapping package and later used map server package to save the map data in .yml and .pgm format.

3) Final Robot Visualization Map: The final robot visualization map was obtained using the UGV wheel encoders, IMU data, and GPS measurements.

Figure 27 shows the gazebo world with fleet of autonomous robots deployed, Figure 29 captures IMU and laser scan data visualization for the corresponding gazebo world.

Figure 30 shows the robot heading towards the goal location.

X. Conclusion

To conclude, we have successfully implemented a single and multi robot delivery system in 2D using PyGame and in 3D using Gazebo, MoveIt and RVIZ.

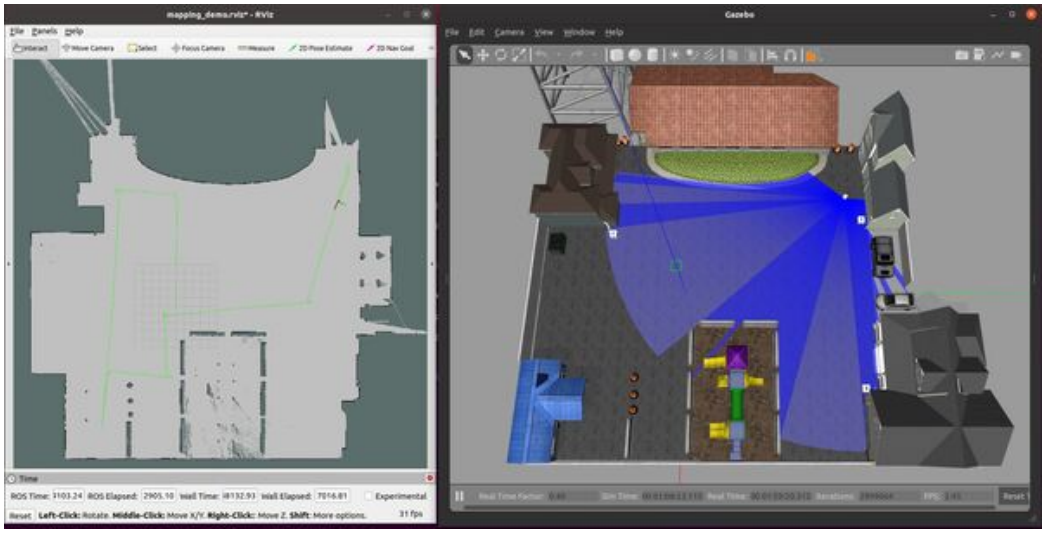

Fig. 23: **Single Robot in Gazebo:** SLAM(b)

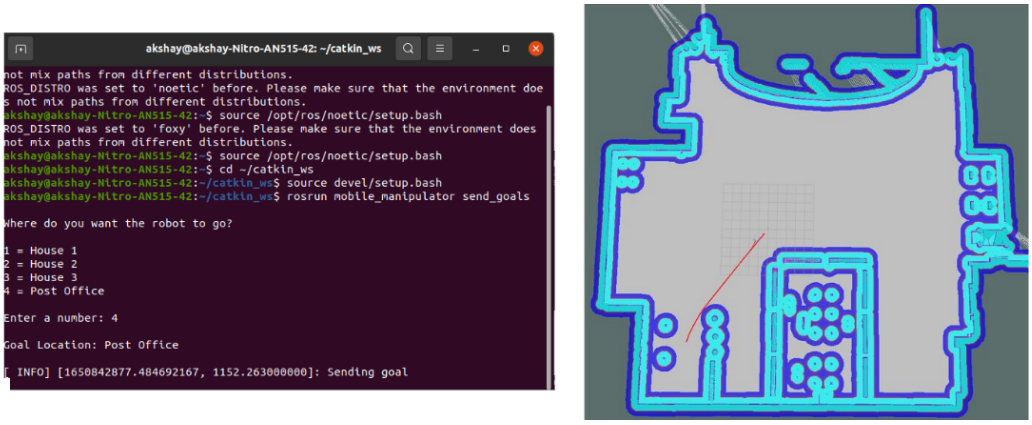

Fig. 24: **Single Robot in Gazebo:** Sending robot to goal location using ROS

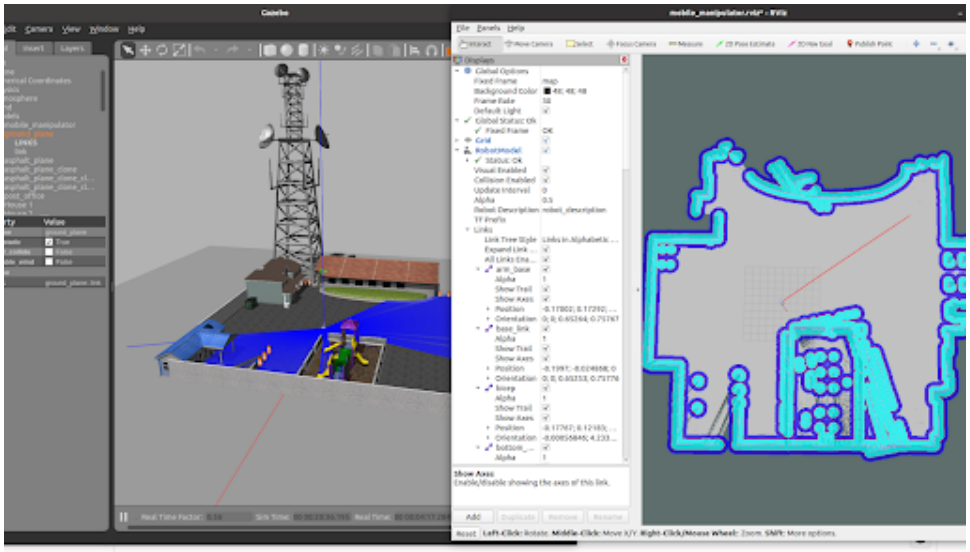

Fig. 25: **Single Robot in Gazebo:** Sending robot to goal location using Rviz

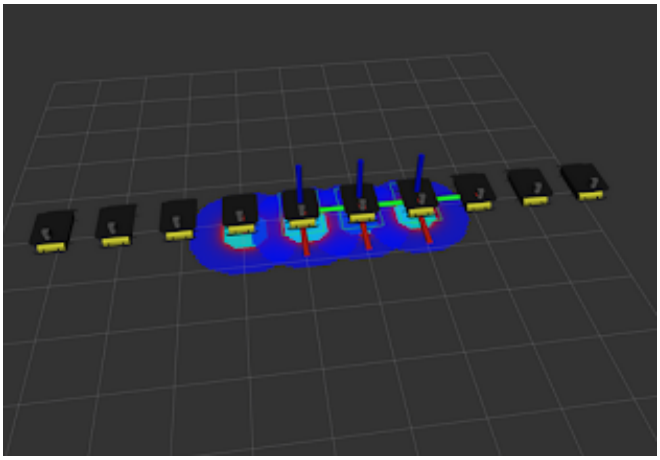

Fig. 26: **Multi Robot Delivery system** Setting up the fleet of robots in Gazebo

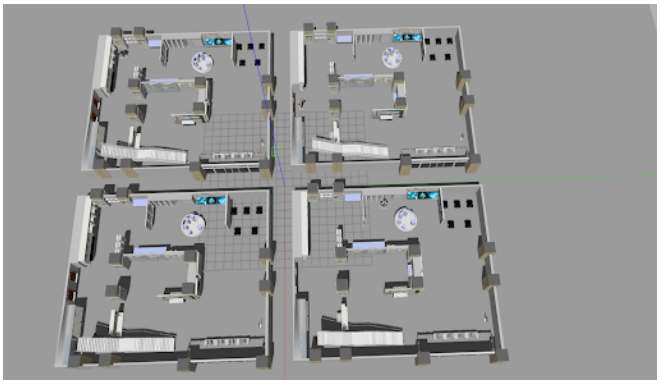

Fig. 27: **Multi Robot Delivery in Gazebo:** Visualization of museum world

A. Risk and Mitigations

- *Risk 1:* Error in Path planning Algorithm due to unknown constraints which might lead to delay in delivery time or failed delivery.
  Mitigation: This can be mitigated by improving the path planning algorithm by feeding the planning data to supervised learning ML algorithms and training the path planner to predict obstacles/possible unknown constraints.
- *Risk 2:* Unexpected shutdown of ROS Master might lead to the shutdown of all the available agents/Robots.
  Mitigation: This risk can be mitigated by migrating the whole project to ROS2 or by integrating a ROS1 bridge with ROS2.
- *Risk 3:* One of the sensors required for path planning might malfunction leading to erratic behavior.
  Mitigation: If such a condition occurs, the robot will be made to override the autonomous navigation signals and a human operator will manually control the robot.

Fig. 28: **Multi Robot Delivery in Gazebo:** SLAM using Gmapping for figure 27

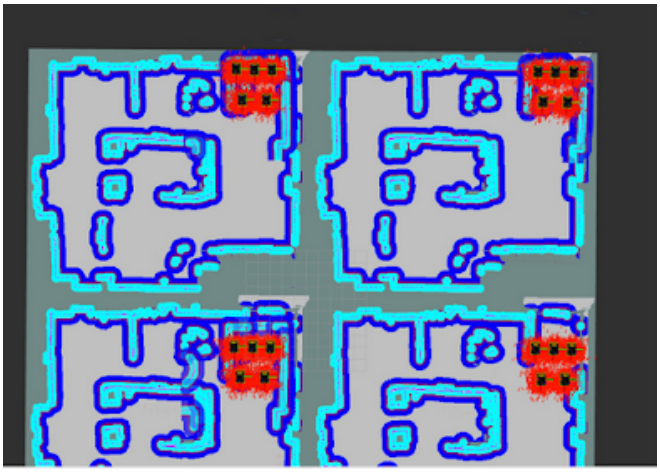

Fig. 29: **Multi Robot Delivery in Gazebo:** IMU and laser scan data visualization for the corresponding gazebo world for figure 27

B. Issues

Due to narrow pathways, the robots are unable to find paths as the cost map generated have high padding. This can be mitigated by using smaller robots or by generating different paths.

Future Work

The significant padding is generated in the cost map and hence the robots are unable to discover paths due to restricted pathways. This can be reduced by using smaller robots or designing alternate pathways.

Acknowledgment

We would like to thank Prof. Flickinger, our instructor for the RBE550: Motion Planning course at the Robotics

Fig. 30: **Simulation of the multi robot system in RVIZ:** The master robot moving towards goal location

Engineering Department of Worcester Polytechnic Institute for supporting our project during the January-May 2022 term. We owe our deep gratitude to him for taking keen interest in our project and guiding us, by providing all the necessary information and timely feedback during this project work.

REFERENCES

[1] https://www.gearbrain.com/autonomous-food-delivery-robots-2646365636.html accessed Jan 29, 2022
[2] https://en.wikipedia.org/wiki?curid=48478069 accessed Jan 29, 2022
[3] Zhang, Xu L, Felix, Ziegler, Christian,Franke. Self-learning RRT* Algorithm for Mobile Robot Motion Planning in Complex Environments. Intelligent Autonomous Systems 13. Springer International Publishing 2016.
[4] He, Z., Wang, J. and Song, C., 2021. A review of mobile robot motion planning methods: from classical motion planning workflows to reinforcement learning-based architectures. arXiv preprint arXiv:2108.13619.
[5] Hachour. Path Planning of Autonomous Mobile Robot. 2008
[6] https://webpages.charlotte.edu/sbadguja/CourseProjectReport_sbadguja.pdf accessed Feb 21, 2022

APPENDIX

- Akshay's Contribution - Akshay worked on creating an environment in V-REP and importing the Pioneer 3dx robot. He programmed the robot to map the environment and focused on integrating VREP and ROS. Also, he performed the robot modeling and designing on Gazebo.
- Anagha's Contribution - Anagha worked on creating the environment on PyGame. She created a start and end point for the robot and implemented RRT algorithm on the same. Further, she worked on creating the environment on Gazebo and launching the robot.
- Kunal's Contribution - Kunal created environment on CARLA. He navigated through the commands and explored how things can be customized. Moreover, he worked on creating aerial view in CARLA and explored various other software that could be integrated with CARLA to create 2D map. He also started working on static and dynamic obstacle avoidance.