Autonomous Navigation of Disaster Environments Using SLAM and Nav2

1st Musab Kasbati
Computer Science
Habib University
Karachi, Pakistan
mk07811@st.habib.edu.pk

2nd Bilal Qureshi
Electrical Engineering
Habib University
Karachi, Pakistan
mq08079@st.habib.edu.pk

3rd Syed Meesam
Electrical Engineering
Habib University
Karachi, Pakistan
sz08464@st.habib.edu.pk

4th Zaid Bin Khalid
Electrical Engineering
Habib University
Karachi, Pakistan
zk08128@st.habib.edu.pk

***Abstract*—This paper presents the design and implementation of an autonomous differential-drive robotic system capable of navigating complex disaster-stricken environments using SLAM (Simultaneous Localization and Mapping) and Nav2 toolboxes in ROS2 and Gazebo environments. The robot autonomously detects obstacles, generates an updated map of its environment, and is able to navigate to any given goal point on the map. The primary objective is to enable effective navigation and victim assistance in unpredictable environments, leveraging lidar data and advanced path planning algorithms. The simulated results demonstrated the robot's ability to navigate an unfamiliar environment with high accuracy and efficiency.**

***Index Terms*—SLAM, Nav2, ROS2, Gazebo, Autonomous Navigation, Disaster Recovery Robotics**

I. Introduction

Disaster-stricken environments, such as post-earthquake zones, collapsed buildings, and flooded areas, often present high-risk scenarios where human intervention is challenging and hazardous. When emergencies occur, unfamiliar environments are difficult and dangerous for first responders to search and rescue, often leading to secondary casualties[18]. When the victims are located in an indoor environment, emergency medical teams have to spend time navigating the disaster-stricken zone and locating the victims in need of help. This time is crucial as a few minutes difference significantly affects the casualty rate[16]. In such cases, autonomous mobile robots serve as invaluable tools for search and rescue operations. These robots can efficiently navigate through unpredictable environments, identify survivors, and relay critical information to rescue teams, reducing response time and minimizing risks to human rescuers. The primary challenge lies in enabling autonomous robots to operate effectively in unstructured, dynamic environments. Traditional navigation methods are insufficient when dealing with obstacles such as debris, irregular terrains, and moving objects. Additionally, robots must not only traverse unknown regions but also update their understanding of the environment in real-time. This requires advanced techniques for Simultaneous Localization and Mapping (SLAM), path planning, and obstacle avoidance to ensure reliable performance in critical scenarios. This project focuses on designing and simulating an autonomous differential-drive robotic system capable of navigating complex disaster environments. We employ the Robot Operating System 2 (ROS2) and Gazebo for simulating realistic scenarios. The robot leverages the SLAM Toolbox to construct and update maps of the environment while localizing itself within the space. For navigation, the Nav2 Framework is utilized to generate feasible paths, avoid obstacles, and control the robot's motion using real-time sensor data, specifically Lidar-based laser scanning. The simulated environment replicates post-disaster conditions with static obstacles, such as rubble/debris, to test the robustness of the system. Through this project, we aim to achieve the following objectives: Accurate Environment Mapping: Implementing SLAM algorithms to generate real-time maps of unknown environments while handling sensor noise and inaccuracies. Path Planning and Navigation: Developing efficient path planning strategies using Nav2 components like A* and MPPI controllers to ensure smooth, obstacle-free navigation. Obstacle Avoidance: Adapting the robot's behavior to avoid obstacles while maintaining progress toward goal points.

II. Related Work

Simultaneous Localization and Mapping (SLAM) has emerged as a vital technology in disaster response, offering real-time mapping and localization capabilities in unstructured and dynamic environments. [14] highlighted the role of SLAM systems in improving search and rescue efforts, damage assessment, and resource allocation during disasters. SLAM's ability to operate autonomously using a combination of sensors, such as cameras, LiDAR, and IMUs, enables it to create detailed maps while tracking the agent's pose within these maps. However, the authors noted that the integration of SLAM in complex and resource-constrained disaster zones remains a challenge, as traditional SLAM algorithms face issues with sensor noise and dynamic environmental changes [14].

[15] emphasized the role of UAV-based SLAM systems for disaster assessment and relief. By leveraging advanced vision techniques, such as image stabilization, target recognition, and 3D reconstruction, UAVs equipped with SLAM technology can map disaster areas effectively while detecting survivors and hazardous zones. Zhang discussed the importance of improving real-time performance and robustness in SLAM systems, particularly when navigating unpredictable

post-disaster conditions. This study highlights the integration of deep learning-based visual techniques to enhance target detection and navigation accuracy [15].

Indoor disaster environments pose unique challenges for first responders, where GPS systems are unreliable, and complex building layouts can impede navigation. [16] proposed a visual SLAM-based indoor localization system tailored for emergency situations. Their work achieved a 94% success rate in accurately localizing individuals and guiding responders to critical areas. The study emphasized the importance of thresholding feature extraction and addressing lighting variability for accurate localization. The authors demonstrated the applicability of SLAM in reducing response times during emergencies, particularly for locating victims in dense urban buildings where traditional GPS-based methods fail (16).

Similarly, [18] introduced a radio-frequency-based localization framework for building emergency response. By combining SLAM with ad-hoc sensor infrastructure, their approach achieved room-level and coordinate-level accuracies, enabling first responders to navigate complex building layouts safely. The study underscored SLAM's role in addressing the situational awareness needs of responders by dynamically updating maps of hazardous indoor environments [18].

Integration of SLAM with Path Planning and Autonomous Navigation

While SLAM technology focuses on creating accurate maps and maintaining localization, its integration with path planning algorithms enables autonomous navigation through disaster zones. [4] presented a dynamic fire rescue system that combines real-time path optimization, visual SLAM, and wireless sensor networks. The framework provides firefighters with optimized navigation routes in real time, helping them locate trapped occupants efficiently. The study emphasized the critical role of SLAM in creating accurate 3D maps of fire-affected buildings, allowing responders to adapt their routes dynamically based on environmental conditions [4].

Moreover, [21] proposed a cloud-based drone system for disaster response that employs SLAM to generate 3D maps and locate survivors. Their system utilizes a network of specialized drones, where mapping data generated by high-end pathfinder drones is shared with lower-cost drones tasked with rescue and supply drop operations. This approach reduces costs while maintaining the accuracy and robustness required for autonomous navigation in post-disaster environments [21].

SLAM for Situational Awareness in Dynamic Environments

[20] explored the use of SLAM for enhancing situational awareness during indoor first responder operations. The proposed system creates real-time maps and tracks the location of responders, enabling remote coordinators to maintain a common operational picture (COP). This situational awareness enhances decision-making by providing a comprehensive view of the operation, including potential obstacles and blocked pathways. Smit's work highlights the challenges of tracking loss in narrow or dynamic spaces, which remain key areas for improvement in SLAM systems [20].

While existing studies have explored the application of SLAM in disaster response, challenges such as dynamic environmental changes, sensor reliability, and real-time performance persist. Our work builds on these foundations by integrating SLAM Toolbox with ROS2's Nav2 framework for autonomous navigation in unstructured disaster environments. By combining real-time mapping, path planning, and obstacle avoidance, our project aims to address the limitations of previous works while providing a practical and scalable solution for disaster response scenarios.

III. System Architecture

The robot is a differential-drive system equipped with a Lidar scanner for environmental perception. We assume that the robot receives a relative position of the target and its goal is to navigate towards the target, determining a safe path for a manned rescue team. The high-level architecture consists of the following:

- LiDAR Measurements: The robot uses the attached LiDAR sensor to observe the environment. Note that it initially has no knowledge of the map so it needs some way to get information from the environment.
- SLAM Module: Through the measurements obtained from the LiDAR, as well as information deduced through odometry, the robot attempts to build a map while recording its position within this map. This information is needed for the robot to plan out its motion towards the target
- Navigation Stack (Nav2): Given the map and the robot's position within it, Nav2 handles the global planning towards the target, keeping in mind walls and blocked paths, local path planning, to avoid nearby obstacles, path smoothing, to simplify and robot movement making it faster.

The robot operates in a simulated indoor disaster scenario (an office environment post-earthquake). A victim (person) is spawned at a random location, and the robot navigates toward the person using noise triangulation or predefined coordinates.

IV. System Details

A. Robot Design

We designed a simple differential drive robot with minimal footprint and height to allow it to navigate between and under obstacles, and through complex environments. The robot is equipped with a lidar Sensor at the top. The robot design can be seen in Figure 1

B. LiDAR Integration

Sensors are an important aspect of every robot application, the sensors are there as an input device, which perceives the data from the environment and then feeds that data to the robot so that it can use that data to make multiple decisions which also includes planning and performing the kinematics.
There are two ways a robot can navigate in an environment:

1) The robot is being provided with the entire global map initially, so that it can plan its path before and then start

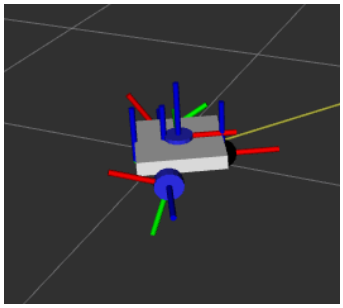

Fig. 1. Robot Model

navigating based on that. In most scenarios it would require the sensors for local planning and correcting its pose which can be caused due to some errors.

2) The second method is depending entirely on the sensors in a scenario where the map is unknown, as only the goal point is known in this situation then the robot entirely depends on local planning using the data from the sensors and in some scenarios also mapping the environment for future use.

For our robot application, the latter scenario is more relevant, as we are navigating in a disastrous situation, where only the goal point is known, the robot depends on a sensor for navigating and mapping.

We are using a 2d lidar sensor placed on top of our robot and we have placed the origin of the sensor on top of the origin of our robot. As we will be using this sensor for mapping, we are scanning the area of $-\pi$ to π around the robot so that it has a sense of all the area around it. For that purpose the resolution is set to 1 laser beam per degree such that we have 360 beams of the laser which are individually analyzing the data and creating a virtual map for the robot based on the object they detect. The frequency has been adjusted to 10 Hz, which means the data is updated after every 0.1 seconds, this update rate is enough for our robot as it is not moving at such high speeds.

As all of this has been done in xacro file for gazebo, we have used

```
filename = "libgazebo_ros_ray_sensor.so"
```

this is a plugin of Gazebo, and it takes the ray sensor, and make it communicate with ROS, so that we can use the data for our application.

We have created a topic of *LaserScan* which contains all the data from the lidar. *LaserScan* is the message type for the 2d lidar sensor in ROS and it is inside the *sensormsg*. *LaserScan* gives details of the range reading, which it receives from the lidar sensor while the robot is navigating in the Gazebo simulation and perceiving the environment. To use this data, we need a lidar node which will publish the message of *LaserScan*.

```
<output_type>
    sensor_msgs/LaserScan
</output_type>
```

Using the above command we are telling the *LaserScan* is the topic which we need to publish and then using the following command,

```
<argument>~/out:=scan</argument>
```

We are publishing the *LaserScan* to another topic, *scan*. The topic *scan* is then used in SLAM and for navigation using nav2 to move around the environment and simultaneously create the map.

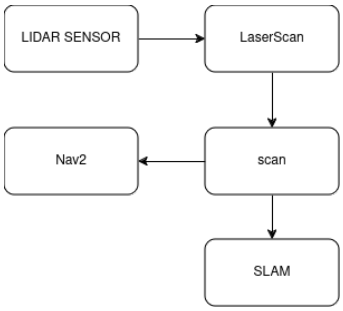

Fig. 2. Flow Diagram

C. SLAM

Simultaneous Localization and Mapping (SLAM) enables a robot to concurrently construct a map of its environment while determining its own position within that environment. For instance, consider a scenario where a robot enters an unfamiliar room. How does the robot identify its location? How does it perceive the layout of the room? These fundamental challenges are addressed by the SLAM algorithm.

The concept of SLAM can be intuitively understood using an analogy of navigating a dark room with a flashlight. Imagine standing in a room with limited visibility and a flashlight to illuminate nearby walls. By measuring the distances between yourself and the walls while keeping track of your movements, you can gradually construct a mental map of the room. Simultaneously, it is crucial to maintain awareness of your position relative to the room to ensure accurate mapping.

In practice, SLAM utilizes sensors such as LiDAR to gather environmental data, such as the distances between the robot and nearby obstacles or walls. It may also detect and record unique features within the environment, such as doors, windows, or corners, to refine position estimates. The SLAM process involves several critical tasks, including calculating the robot's position, constructing and updating the map, and correcting any inconsistencies. There are various approaches to SLAM, including visual, LiDAR-based, graph-based, and filter-based methods. Each of these have their own benefit in terms of accuracy and computation costs.

For our purpose, we make use of an online asynchronous graph-based SLAM. Online indicates that the map is generated in real-time, while asynchronous allows us to make use of readings from multiple sensors despite temporal differences.

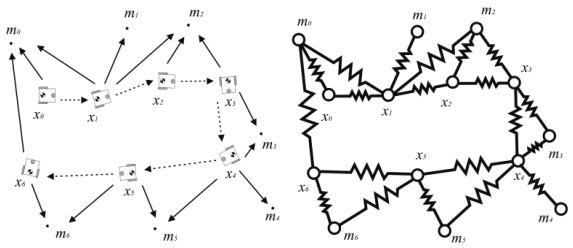

Fig. 3. Pose Graph. Soft connections between states and landmarks can also be modeled as springs. [3]

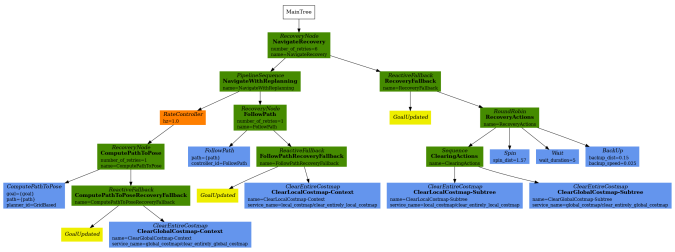

Fig. 4. *navigate_to_pose_w_replanning_and_recovery* behavior tree [4]

While we do not leverage the asynchronous capability of the algorithm we deploy, it gives us the option to extend the system with more sensors for more robust mapping.

Graph-based SLAM works by modeling relations between robot poses and landmark observations at different times. In Figure 3, the robot positions at different timesteps are indicated by the nodes marked $x_1, x_2, \ldots x_6$, while landmarks are indicated by the nodes marked $m_0, m_1, \ldots m_6$. The connection between these vertices represents the distances which can be estimated by odometry information and LiDAR sensor readings. However, we note that these connections can only be estimated. We can leverage this by modelling the connections as soft connections. For every possible state of the connection, we can determine a likelihood. The likelihoods across all possible connections can model a constrained equation where the goal is to maximize the overall likelihood of our structure. Thus the original simultaneous mapping and localisation problem:

$$x^*_{1:t}, m^* = \arg\max_{x_{1:t}, m} p(x_{1:t}, m \mid z_{1:t}, u_{1:t})$$

translates into the following under the assumption of Gaussian noise [1]:

$$\log p(x^*_{1:t}, m^*) = \text{const} +$$
$$\sum_t [x_t - g(x_{t-1}, u_t)]^T R_t^- 1 [x_t - g(x_{t-1}, u_t)] +$$
$$\sum_t [z_t - h(x_t, m)]^T Q_t^- 1 [z_t - h(x_t, m)]$$

This transforms our SLAM problem into a quadratic form.

The problem of minimizing quadratics is called the least squares problem. Many optimizers already exist for this problem. We leverage the one implemented in SLAM Toolbox by default, i.e. the Ceres Solver, Sparse Normal Cholesky. [2]. The resulting map and pose information is used in navigation.

D. Navigation

We make use of the Nav2 stack for robot navigation. Nav2 manages robot control using a behavior tree. In particular, we make use of *navigate_to_pose_w_replanning_and_recovery*

The behavior tree, illustrated in Fig. 4, can be roughly understood as follows. Control flows from left to right, and output from lower layers is sent to upper layers. Initially, when the navigation command is sent to Nav2, it first tries to compute a path to the goal using its available planner and the map information. If a path cannot be found, we remove some information from our global costmap (which is derived from the map) to eliminate erroneous map readings that may indicate blocked paths where none exist. If, after clearing the costmap and rebuilding the map, the path is still blocked, this suggests that the path is indeed blocked and not merely a sensor error. However, if the path is not blocked, a new path will be computed. Then, as per the sequence node "NavigateWithReplanning," the robot will attempt to follow the path using its available motion controller. If there are no obstructions and it is able to do so, the robot will proceed toward the goal. However, if it is unable to proceed, it will clear the local costmap to update it and identify previously misidentified obstacles and paths. If the robot is still blocked, it may be stuck on an unobservable obstacle, and the "NavigateWithReplanning" node will return a failure. In this case, the robot enters recovery control, where it typically clears the local and global costmaps, rotates to obtain a better scan of the map, waits for the map to be processed, and then moves backward in case it is caught on an obstacle. This navigation and recovery procedure can be repeated up to six times before the control determines that it is unable to reach the goal from its current position and outputs a failure.

This overall control flow allows for navigation in complex environments. We note that during some phases, the robot performs complex tasks that require specialized algorithms. Specifically, we need algorithms for path planning, path smoothing, and motion control. Nav2 offers a suite of algorithms to choose from, and we detail the algorithms that our robot employs below.

1) Path Planning: For path planning, we employ the Theta* Planner built into Nav2. The version implemented in Nav2 is actually the Lazy Theta* P variant, which is highly optimized. However, the basic variant of Theta* is detailed in [5]. To understand the idea behind this algorithm we must first understand the A* path-finding algorithm. A* is a popular path-finding algorithm for grid environments. It assigns heuristics to different points in the grid based off their euclidean distance to a goal (or some other metric). Then the algorithm proceeds to explore nodes from the starting point, prioritising those paths that have a minimal heuristic distance from the goal by

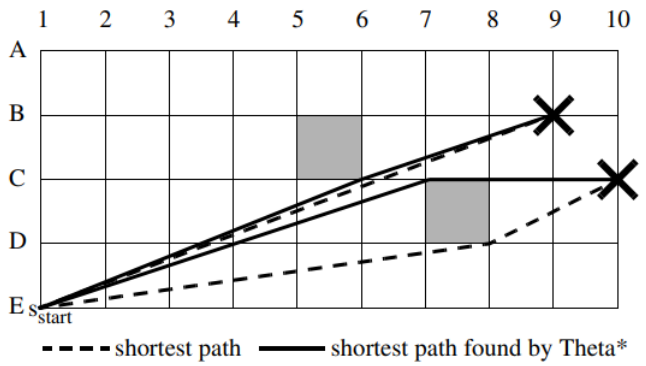

Fig. 5. Result of Basic Theta*

considering the actual cost to a node along with the heuristic cost from the node to the target. This is computationally efficient as it avoids computing paths which only take us away from the goal.

As our map is represented as a grid, A* can be used for path planning here. However, since our actual environment is continuous and we only discretize it for the sake of grid-based mapping, following the base A* path leads to an unnecessarily jagged and long path. One key insight to optimize this is that we can create connections between non-neighbouring nodes if the line in between is unoccupied. Theta* does exactly this. In other words, if there is a line of sight between a node and its grandparent, we can skip the parent of that node. This can simplify unnecessarily jagged paths and reduce overall path length. This approach can still further be optimized by allowing nodes to be parents to nodes that are not amongst its descendants and is one of the key ideas used to further optimize Theta*. We outline the algorithm for basic Theta* in Algorithm 1. Some results of Theta* can be seen in 5. We also note another benefit of the Theta* planner, namely, its any-angle path-finding. This allows the robot to consider more complex motions, offering it the flexibility of movement to move through the tighter regions that our robot will be faced with, which makes this path-planning algorithm well-suited for our task.

Another key concept in path planning is the cost map. Whilst we want the path to our goal to be as short as possible, we do not want it to be easy for the robot to navigate. Thus, we want to keep the robot away from obstacles. For this purpose, we derive a costmap which serves as a critic favoring paths that keep distance from walls and obstructions. Simply put, it gives a high cost to blocked cells on the map, as well as a cost to cells close to blocked cells. The cost given to these nearby non-blocked cells decreases exponentially with their distance to the blocked cell, thus discouraging paths that go too near walls.

This concludes the details of our path planner, however, we must also briefly consider why we used this particular planner.

2) Path Smoothing: For path smoothing we use the Savitzky-Golay smoother/filter [6]. A key feature of this

Algorithm 1 Theta* (adapted from [5])

1: **Main()**
2: $g(s_{start}) := 0$
3: $parent(s_{start}) := s_{start}$
4: $open := \emptyset$
5: $open.Insert(s_{start}, g(s_{start}) + h(s_{start}))$
6: $closed := \emptyset$
7: **while** $open \neq \emptyset$ **do**
8: $s := open.Pop()$
9: **if** $s = s_{goal}$ **then**
10: **return** "path found"
11: **end if**
12: $closed := closed \cup \{s\}$
13: UpdateBounds(s)
14: **for all** $s' \in succ(s)$ **do**
15: **if** $s' \notin closed$ **then**
16: **if** $s' \notin open$ **then**
17: $g(s') := \infty$
18: $parent(s') := NULL$
19: UpdateVertex(s, s')
20: **end if**
21: **end if**
22: **end for**
23: **end while**
24: **return** "no path found"
25:
26: **UpdateVertex(s, s')**
27: **if** lineofsight($parent(s), s'$) **then**
28: **/* Theta* optimization */**
29: **if** $g(parent(s)) + c(parent(s), s') < g(s')$ **then**
30: $g(s') := g(parent(s)) + c(parent(s), s')$
31: $parent(s') := parent(s)$
32: **if** $s' \in open$ **then**
33: open.Remove(s')
34: open.Insert($s', g(s') + h(s')$)
35: **end if**
36: **end if**
37: **else**
38: **/* default A* */**
39: **if** $g(s) + c(s, s') < g(s')$ **then**
40: $g(s') := g(s) + c(s, s')$
41: $parent(s') := s$
42: **if** $s' \in open$ **then**
43: open.Remove(s')
44: open.Insert($s', g(s') + h(s')$)
45: **end if**
46: **end if**
47: **end if**=0

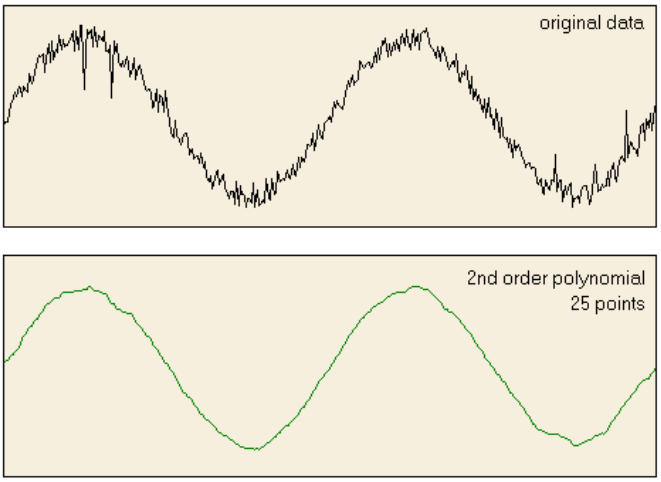

Fig. 6. Savitzky-Golay filtering applied to a noisy signal considering a 25 point window and order 2 polynomials

smoother is that it captures the characteristic of the motion, allowing for the robot to follow close to the planned path while simplifying unnecessarily rough movements. Following the path closely is necessary to prevent from running into unaccounted for obstructions. The way this filter works is that for each point, it remaps it by considering a window of points around it and taking a weighted average of them. The simplest idea would be to use equal weights for averaging, however, this can lose out on important characteristics of the path, for example reducing peaks and troughs. The solution employed by Savitzky and Golay is to fit the points to a polynomial with lower degree than the number of points in the window. This effectively translates into a least squares problem which has many existing optimizers as discussed earlier. An example of Savitzky-Golay filtering is shown in Figure 6.

3) Controller: For controlling robot motion we use a MPPI (Model Predictive Path Integral) Controller. This controller generates new local trajectories by considering the previously determined optimal trajectory for the previous timestep and then adding noise onto it to generate many new potential trajectories. Each trajectory is scored by how well they specify certain metrics/critics which can be tuned for any application. These critics prioritise a smooth path tracking and active collision avoidance behavior for example: goal angle, goal distance, obstacle map, path align, prefer forward, anti-twirling. For our particular use, we mainly used the default configuration, however, we enabled the anti-twirling critic which was disabled by default.

E. Model Predictive Path Integral (MPPI) Control

Model Predictive Path Integral (MPPI) Control is a sampling-based stochastic optimal control algorithm that generates optimal control inputs by evaluating trajectories under uncertainty. It operates by sampling multiple control trajectories, evaluating their associated costs, and applying a weighted average to iteratively improve the control sequence. Unlike traditional approaches, MPPI avoids solving backward partial differential equations and instead approximates the solution through Monte Carlo sampling.

The MPPI control formulation begins with a discrete-time representation of the system dynamics:

$$x_{t+1} = x_t + \big(f(x_t,t) + G(x_t,t)u_t\big)\Delta t + B(x_t,t)\epsilon\sqrt{\Delta t}, \quad (1)$$

where x_t represents the state at time t, u_t is the control input, $f(x_t,t)$ and $G(x_t,t)$ are the drift and control transition matrices, $B(x_t,t)$ represents the noise covariance, and ϵ is Gaussian noise.

The goal of MPPI is to minimize the cost functional:

$$S(\tau) = \phi(x_T) + \sum_{t=0}^{T-1} q(x_t,t) + \frac{1}{2}u_t^T R u_t \Delta t, \quad (2)$$

where $S(\tau)$ is the trajectory cost, $\phi(x_T)$ is the terminal cost, $q(x_t,t)$ is the state-dependent running cost, and R is the control cost weight matrix.

1) Path Integral Formulation: Using an exponential transformation of the value function and applying the Feynman-Kac lemma, the optimal control input is obtained as a weighted sum of random control perturbations:

$$u_t = \hat{u}_t + \frac{\sum_{k=1}^{K} \exp\left(-\frac{1}{\lambda}S(\tau_k)\right)\delta u_{t,k}}{\sum_{k=1}^{K} \exp\left(-\frac{1}{\lambda}S(\tau_k)\right)}, \quad (3)$$

where K is the number of sampled trajectories, $S(\tau_k)$ is the cost of the k-th trajectory, λ is the temperature parameter controlling exploration, and $\delta u_{t,k}$ is the noise added to the control input.

The control update law can be interpreted as a reward-weighted average of the sampled control perturbations, where trajectories with lower costs are assigned higher weights.

2) Algorithmic Implementation: The MPPI algorithm iteratively samples trajectories, evaluates their costs, and updates the control sequence. The implementation can be summarized as follows:

Algorithm 2 Model Predictive Path Integral Control [22]

Given: K (number of samples), T (time horizon), initial control sequence $\{u_0, u_1, \ldots, u_{T-1}\}$
Initialize λ, R, dynamics f, G, B, and cost function $S(\tau)$
while task not complete **do**
  Generate random control perturbations $\delta u_{t,k}$ for $k = 1, \ldots, K$ and $t = 0, \ldots, T-1$
  for $k = 1, \ldots, K$ **do** {Sample trajectories}
    Simulate dynamics for trajectory τ_k using perturbed control inputs $u_t + \delta u_{t,k}$
    Compute cost $S(\tau_k)$
  end for
  for $t = 0, \ldots, T-1$ **do** {Update control inputs}
    $u_t \leftarrow \hat{u}_t + \frac{\sum_{k=1}^{K} \exp\left(-\frac{1}{\lambda}S(\tau_k)\right)\delta u_{t,k}}{\sum_{k=1}^{K} \exp\left(-\frac{1}{\lambda}S(\tau_k)\right)}$
  end for
  Execute u_0 and shift the control sequence forward
end while=0

V. Experimental Results

The robot was tested in a simulated environment using Gazebo and ROS2. The trajectory, which the robot took and the the path it planned during navigation was tracked using RVIZ. The robot was tested in several different situations which are possible in an actual scenarios during an earthquake, as the robot will not always have an easy way to navigate, and in some situations it might not even have the way to reach the goal.

Different situations have been tested which includes:

- Navigating to a relatively easy goal point where the goal is near and the path is also relatively open and easy to reach as depicted in Fig. 7, the robot easily reached the goal point in the above situation as visible in Fig. 8. Moreover the trajectory tracking in RViz which is shown in Fig.9, this is also shows the robot took a relatively easy and straight path, avoiding unnecessary long path. However due to nature of MPPI controller, the robot might take slightly different path for each iteration of reaching the same goal point.

Fig. 7. Start point in an open situation

Fig. 8. End point in an open situation

- Navigating in a situation where the robot has to move through some tight path to reach the goal as shown in Fig. 10. Our robot eventually made it to the destination as shown in Fig. 11. The robot at this situation took different paths when it was tested was multiple time, however one of the optimal results is being demonstrated in Fig. 12.

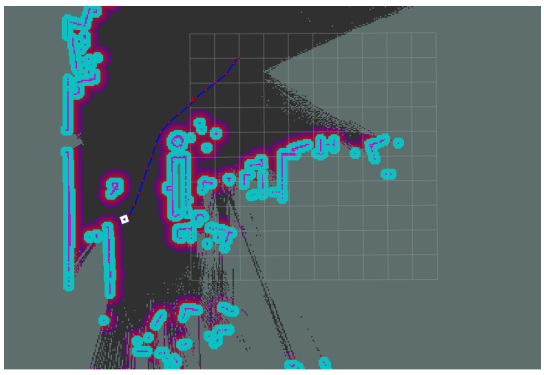

Fig. 9. Trajectory in RViz for open situation

Fig. 10. Start point in tight situation

Fig. 11. End point in tight situation

- There is a possibility that due to the disaster situation, some of the position might be totally unreachable, we have depicted this situation as well by placing the target in a box which is closed from all four sides as shown in Fig. 13. The robot did start to make plan for the goal point as it is unaware at the start that the path is unreachable as

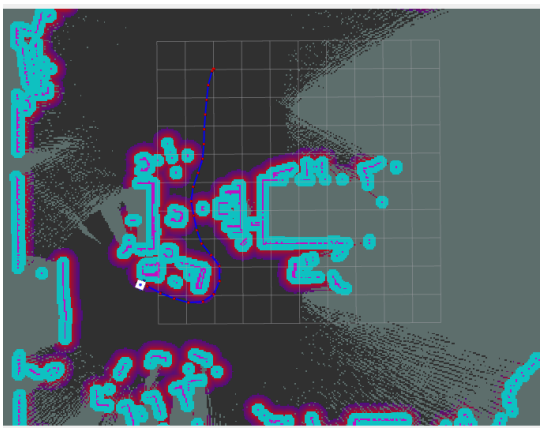

Fig. 12. Trajectory in RViz fr tight situation

shown in Fig. 14, the robot kept moving and it eventually stopped at the point due to the reason that it scanned the area and realized the goal point is unreachable as shown in Fig. 15. Another reason making it harder for the robot to go through the whole space around the box was the tight spaces around it

Fig. 13. Start point in unreachable situation

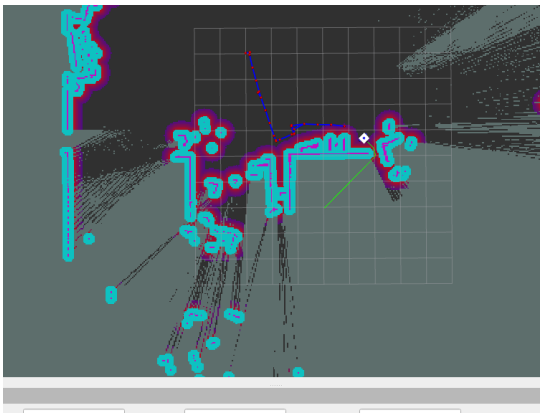

Fig. 14. Planning for unreachable situation

Fig. 15. Robot going to halt

- The last scenario which we tested was was a situation when the robot is very far away from the target person as in Fig. 16. For this there were several iterations done and the robot was able to reach to the target in most of the cases as in Fig. 18.The strategy of the robot was to chose the shortest path but eventually had to change the path as it kept moving forward and kept finding the blocked areas, due to this reason in a few scenarios for the urge to take the shortest path, robot got suck at center where there were a lot of obstacles, unable to then find a way out. However in most of the cases it did eventually found its way out and successfully reached its goal, a trajectory for such path is shown in Fig. 18

Fig. 16. Start point when target is far

All of these experiment are done using MPPI controller, Theta* Planner and Savitzky-Golay smoother as described in the paper.

VI. Assumptions and Limitations

A. Assumptions

In this project, several assumptions have been made to simplify the scope and focus of the study:

Fig. 17. Reaching Goal point

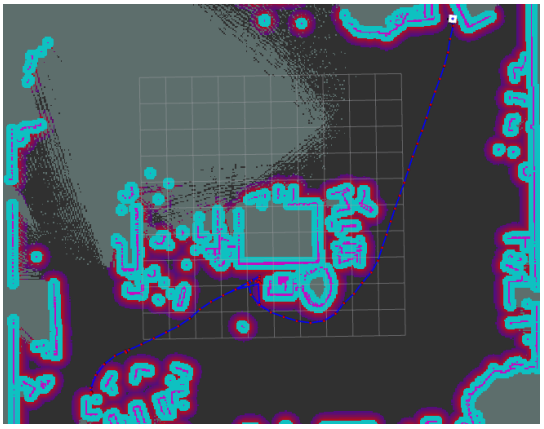

Fig. 18. Rviz for trajectory tracking in a long distance

- Path Existence: It is assumed that, for any given scenario, a path always exists between the robot's initial position and the designated goal point. This path is assumed to be free of obstacles, ensuring that the robot does not require any movement in the z-direction to reach its destination.
- Single Goal Point: The project operates under the premise that there will always be a singular goal point within the map, simplifying the path-planning and navigation process.

B. Limitations

While the project demonstrates promising results, there are certain limitations that must be acknowledged:

- Absence of Field Testing: The system has not yet been deployed on actual hardware for real-world validation, and all tests have been conducted in simulated environments.
- Limited Test Scenarios: While several test cases have been performed within the scope of the project, additional considerations, such as handling multiple victims, remain unaddressed for real-world deployment.
- Static Environments: The robot has not been tested in dynamic environments with moving objects, which is a critical consideration for real-world disaster response applications. The person in need of help could be moving, which the simulated model does not currently account for.
- Visibility Constraints: The system's performance has been evaluated only for static objects, without accounting for challenging visibility conditions such as smoke or fire, which are common in disaster scenarios.
- Comparative Performance: The robot's performance across various maps has not been quantitatively compared, leaving gaps in understanding its adaptability to different layouts with varying complexities.

VII. Conclusion and Future Work

This project successfully demonstrates an autonomous robotic system capable of navigating complex and disaster-stricken environments with precision. By leveraging Simultaneous Localization and Mapping (SLAM) and the Nav2 framework within ROS2, the system achieves robust mapping, path planning, and obstacle avoidance in unstructured environments. The integration of these tools highlights the system's ability to create real-time, detailed maps while ensuring optimal navigation to a predefined goal point.

The project addresses critical challenges in disaster recovery robotics, such as autonomous movement in dynamically changing environments, and underscores the importance of scalable solutions for mapping and navigation. The modular design of the system allows for adaptability to various real-world scenarios, ensuring its relevance across diverse applications like search and rescue, hazard assessment, and resource allocation.

A. Future Work

While the current implementation provides a solid foundation, further enhancements are necessary to realize the full potential of the system in real-world disaster response scenarios. Future research directions include:

- Dynamic Obstacle Handling: The integration of advanced path-planning algorithms to account for dynamic objects, such as moving debris or survivors, will enable the system to adapt in real time, improving its robustness in unpredictable environments.
- Victim Identification and Localization: Incorporating computer vision techniques and machine learning-based target detection will allow the robot to identify and localize survivors within the disaster zone. Techniques like deep learning-based visual SLAM and object recognition will enhance the robot's situational awareness and rescue efficiency.
- Multi-Victim Rescue Planning: Extending the system's capabilities to handle multiple victims across large-scale environments will require optimization algorithms for resource allocation and path optimization. This addition would enable the robot to prioritize and navigate to victims efficiently based on their locations and accessibility.

- Real-World Deployment and Field Validation: Conducting extensive field tests on physical robots in realistic disaster settings will validate the system's performance under real-world constraints, such as sensor noise, dynamic lighting, and environmental disturbances like smoke, fire, and debris. These tests will ensure that the system meets practical reliability and robustness requirements.
- Multi-Robot Collaboration: Expanding the system to include multi-robot coordination for collaborative mapping, exploration, and rescue operations will improve coverage and reduce response time in large disaster zones. Integrating cloud-based SLAM frameworks can facilitate seamless communication and data sharing between robots.

B. Final Remarks

The proposed system sets a strong foundation for autonomous disaster response robotics by addressing core challenges in mapping, localization, and navigation. Future enhancements will significantly improve the system's capabilities, enabling it to function effectively in highly dynamic and resource-constrained environments. With further development, this solution has the potential to become a critical tool in improving situational awareness, accelerating search and rescue operations, and ultimately saving lives during disaster recovery missions.

REFERENCES

[1] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*. Berlin, Heidelberg: Springer-Verlag, 2007.

[2] S. Agarwal, K. Mierle, and T. C. S. Team, "Ceres Solver," 10 2023.

[3] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*. The MIT Press, 2nd ed., 2011.

[4] S. Macenski, F. Martín, R. White, and J. Ginés Clavero, "The marathon 2: A navigation system," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[5] K. Daniel, A. Nash, S. Koenig, and A. Felner, "Theta*: Any-angle path planning on grids," *CoRR*, vol. abs/1401.3843, 2014.

[6] A. Savitzky and M. J. E. Golay, "Smoothing and differentiation of data by simplified least squares procedures.," *Analytical Chemistry*, vol. 36, pp. 1627–1639, Jul 1964.

[7] "Ros2 tutorials: Writing a simple publisher and subscriber (python)." https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Py-Publisher-And-Subscriber.html. Accessed: 2024-12-17.

[8] N. Ahmed, "The types of slam algorithms." Medium, 2020. https://medium.com/@nahmed3536/the-types-of-slam-algorithms-356196937e3d.

[9] "Simultaneous localization and mapping (slam)." MathWorks. https://www.mathworks.com/discovery/slam.html.

[10] X. Research, "Paper information." SCIRP, 2022. https://www.scirp.org/journal/paperinformation?paperid=135222.

[11] A. Smith *et al.*, "Advanced robotics in construction," *ScienceDirect*, 2022. Accessed: 2024-12-17.

[12] S. Macenski *et al.*, "The slam toolbox for the navigation2 system," *Journal of Open Source Software*, vol. 5, no. 27, p. 2783, 2020. Accessed: 2024-12-17.

[13] "Mppi algorithm overview." ACDS Lab, 2020. https://acdslab.github.io/mppi-generic-website/docs/mppi.html: :text=MPPI

[14] Z. T. Ibrahim and J. He, "Slam technology on disaster response," *World Journal of Engineering and Technology*, vol. 12, no. 3, pp. 695–714, 2024.

[15] W. Zhang, "Key technologies based on uav slam vision in disaster response," in *AIP Conference Proceedings*, vol. 3144, p. 030029, 2024.

[16] P.-Y. Tseng, J. J. Lin, Y.-C. Chan, and A. Y. Chen, "Real-time indoor localization with visual slam for in-building emergency response," *Automation in Construction*, vol. 140, p. 104319, Aug 2022.

[17] J.-S. Chou, M.-Y. Cheng, Y.-M. Hsieh, I.-T. Yang, and H.-T. Hsu, "Optimal path planning in real time for dynamic building fire rescue operations using wireless sensors and visual guidance," *Automation in Construction*, vol. 99, pp. 1–17, Mar 2019.

[18] N. Li, B. Becerik-Gerber, L. Soibelman, and B. Krishnamachari, "Comparative assessment of an indoor localization framework for building emergency response," *Automation in Construction*, vol. 57, pp. 42–54, Sep 2015.

[19] A. Kleiner, C. Dornhege, and D. Sun, "Mapping disaster areas jointly: Rfid-coordinated slam by humans and robots," in *Proceedings of the IEEE International Workshop on Safety, Security and Rescue Robotics (SSRR)*, KTH Royal Institute of Technology, Sep 2007.

[20] B. Smit, "Creating remote situation awareness of indoor first responder operations using slam," Master's thesis, Utrecht University, 2020. Accessed: 17 December 2024.

[21] C. Alex and A. Vijaychandra, "Autonomous cloud based drone system for disaster response and mitigation," in *2016 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, IEEE, Dec 2016. Accessed: 17 December 2024.

[22] G. Williams, A. Aldrich, and E. A. Theodorou, "Model predictive path integral control: From theory to parallel computation," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 2, pp. 344–357, 2017.