# Project AutoMapper
## 18500 – ECE Capstone Final Report

Aditya Ranade, Electrical and Computer Engineering, Carnegie Mellon University
Akshat Jain, Electrical and Computer Engineering, Carnegie Mellon University
Wenxin Xiao, Electrical and Computer Engineering, Carnegie Mellon University

### I. Abstract

Our project aims to automate the entire search and reconnaissance phase of disaster recovery operations. Our solution uses an iRobot Create 2 along with a LIDAR and Thermal Camera to autonomously explore any indoor environment, generate a map, and locate all people within it. We envision our project to be a prototype of the first stage to reaching the goal of fully automating such search missions. The fundamental functionality such as obstacle avoidance, complete exploration, and human detection that we implemented can be extended to a more robust robot base able to navigate through rough/uneven environments as well as be used in conjunction with other robots to simultaneously conduct such missions.

### II. Introduction

The goal of our project is to automate the search phase of search and rescue operations in indoor environments. During search, first responders require two essential pieces of information – the map of the environment and the location of people within it. This is crucial as it allows first responders to conduct situational assessment as well as plan an efficient recovery path to rescue victims. We developed a solution which uses an iRobot Create 2 mounted with a LIDAR Scanner, Thermal Camera and a Raspberry Pi Microcontroller to autonomously explore any indoor unknown environment in its entirety and place all people within it.

Disaster recovery operations are often a race against time - to reach as many victims as possible in the shortest amount of time since the expectancy of life decreases exponentially after a certain period of time. As a result, the primary motivation of our project is to reduce the time taken by the robot to finish the search process as well as locate most victims.

We use the technique of Frontier Exploration[1] to autonomously explore an environment and use a Thermal Camera to detect people in the vicinity.

### III. Challenges

Below we highlight four key challenges a mobile robot faces in such operations.

- *Mapping/Localization* - In order for the robot to navigate and find victims, it needs to be aware of its location in relation to the map of the environment.

- *Exploration* – The robot must be able to answer one key question – How can we know that we have explored the entire environment? This becomes even more imperative when human-teleoperation is not possible. The robot must have a heuristic to explore and decide when exploration is complete.

- *Victim Detection* - In order for a successful rescue phase, the robot needs to locate potential victims and overlay them on the map it generates.

- *Mobility* - Disaster recovery environments are often characterized by hard-to-traverse terrains and obstacles. After some consideration, we determined that we would have to use a 3D depth scanner and much more sophisticated navigation algorithms in order to traverse the environment. As a result, we designated this challenge to be out of scope for our project.

### IV. Assumptions

Since the scope of our project is quite wide, we made a few assumptions about the environment we are operating in to narrow it down.

1. *Flat Terrain* – We assume the terrain we are traversing to be flat and smooth, allowing us to use a 2D depth scanner. Moreover, all obstacles have to cross the height of the LIDAR in order to be detected.

2. *Static Environment* – Mapping is very susceptible to variations in the environment. So, we assume that the environment remains static for the duration of exploration.

### V. Design Requirements

In order to satisfy the aforementioned challenges, the robot needs to meet the following specifications.

*1.* ***Hardware***

a) The robot should be able to traverse the test

environment at a minimum of 2ft/s and be able to rotate in place.

b) The robot should have a sensor capable of capturing depth information from the environment.
c) The robot should have an attached embedded microcontroller to which instructions can be sent via pc. The robot should respond to instructions with the appropriate movement in the correct direction.
d) The robot should have some sort of camera to capture information about the environment and the embedded system should be able to stream the captured information to the pc.

***2. SLAM***

a) The robot must localize itself within the environment to a precision of 1 ft in distance and 5 degrees in orientation.
b) The robot must be able to generate a Grid Map of the environment using depth data from sensors.
c) At every iteration, the robot must take no longer than 5 seconds to generate a map and localize itself.

***3. Exploration***

a) The robot must be able to determine where to move in order to gain as much new information about the environment as possible.
b) The robot must be able to generate a complete map of the environment with at least 90% coverage.
c) The robot must avoid all collisions into obstacles/walls.
d) The robot must be able to traverse an environment of size 1500 sq ft. within 12 minutes.
e) The robot must be able to determine when it has completed exploring the entire environment.

***4. Human Detection***

a) The robot must be able to use sensor data to detect human presence.
b) The robot must mark the position of humans to a precision of 2 ft in distance.
c) The robot must be able to detect at least 90% of all potential victims.

Meeting the SLAM specifications will allow the robot to know where it is in relation to the environment. This is a crucial starting point for the critical parts of our project - Exploration and Human Detection, since both of these require a local map of the environment and the pose of the robot to be known.

One of the biggest risk factors for our project is Exploration - the ability of the robot to explore an entire unknown environment from any starting point. We need to be able to implement a robust algorithm that allows the robot to know where to go and how to reach there in order to achieve complete coverage. Specification 3.d requires that we reduce computation time and increase the time the robot spends exploring. This requires efficient path planning and goal searching.

The other risk factor is human detection. Our specifications for human detection require that the robot is able to find at least 90% of all potential victims. This specification follows directly from the goal of our operation which is to find as many potential victims as possible. The hardware we use for this functionality would need to work in conjunction with the exploration to overlay human positions on the map.

## VI. Hardware Architecture

### *Hardware and Design Tradeoffs*

The following section describes our choices and the tradeoffs we had to make while choosing the four major hardware components of our robot i.e the lidar, Raspberry PI, IMU sensor and thermal camera. We made these decisions considering our requirements, our budget as well as the ease of use.

#### *Mobile Robotic Platform*

The mobile robotic platform is the most important part of our robot as this would house all the sensors and move around the test area. Most robotic platforms met our speed requirements as the robot would move around relatively slowly (~3 km/h). We essentially had to choose between a 4 wheeled square mobile robot or a circular robot like the iRobot Create 2. While the iRobot afforded us some advantages like an extremely easy to use Python API, inbuilt obstacle avoidance and easy turning due to its circular shape, it also cost $200 and would leave no room for slack in our budget in case of any sensor or platform failure. We also needed to implement our own obstacle avoidance algorithm as it would have to work with the Path Planning node in order to plan the path and avoid obstacles at the same time. Our exact requirements and how closely the iRobot Create 2 and the 4 wheeled square mobile robot matched up with them are listed in the table below.

| Requirements | iRobot Create 2 | 4 Wheeled Mobile Robot |
|---|---|---|
| Max Load > 6 kg | 9 kg | 15 kg |
| Space for Lidar | Mount needed. | No mount needed. Can simply screw on top. |
| Space for Thermal Camera | Mount needed. | No mount needed. Can simply screw on. |

| | | |
|---|---|---|
| Space for Raspberry Pi | Yes. Space in cargo bay. | Space for embedded system on platform. |
| Obstacle Avoidance | Inbuilt | Need to use lidar data and write own obstacle avoidance algorithm. |
| Ease of Use | Very easy to use Python API over USB Serial. | No API. Need to control motors individually over USB serial. |
| Turning | Circular. Can easily turn in place. | Need to individually control wheels to turn in place. |
| Price | 200$ | 75$ |

Fig. 1. Tradeoffs between two robot chassis

After going through the tradeoffs for each robot platform, we decided to choose the 4 wheeled mobile robot platform over the iRobot Create 2 primarily because of the cost and the mounts needed for the sensors on the iRobot Create 2. Although it would be much easier to use the iRobot, we decided that it would be safer to go with the 4 wheel mobile robot platform and implement the turning and obstacle avoidance on our own. Unfortunately, the robot we chose came with multiple parts missing and got immediately sold out after that. As a result, we were not able to replace the missing parts and had to return it. We decided to not waste further time by choosing another robot and decided to go with the iRobot Create 2 instead as it came fully assembled and we got a discount code for it which effectively made the cost $130. The robot chassis we chose is shown in Fig. 2.


Fig. 2. iRobot Create

### *Embedded Microcontroller*

We essentially had to choose between a Raspberry Pi and Arduino. Both the Raspberry Pi and Arduino had a GPIO header to interface with all the sensors and the robot platform, both were equally easy to use and worked with ROS. In the end, we went with the Raspberry Pi because all of us had experience working with the Raspberry Pi in previous courses or research. We already have a Raspberry Pi 3A+ so we will not need to buy it. A complete list of all the GPIO ports we would be using along with the sensors we would be using them for is given below. A detailed wiring diagram of the connections we will be making is also given below in Fig 4.

| GPIO Pin(Pin Number) | Sensor/Device |
|---|---|
| **Power(1)** | Lidar, Robot, Thermal Camera |
| **Ground(6,9)** | Lidar, Robot, Thermal Camera |
| **PWM(32)** | Lidar Motor |
| **RX1(8)** | Lidar |
| **TX1(10)** | Lidar |
| **RX2(16)** | Robot Control |
| **TX2(18)** | Robot Control |
| **SDA(3)** | Thermal Camera |
| **SCL(5)** | Thermal Camera |

Fig. 3. GPIO pins

### *IMU*

The decision to include an IMU was made after deciding on the initial mobile robot platform. Since the platform we chose had very big wheels, the wheel's odometry data could not be trusted as the test environments would have smooth surfaces. We decided to go with the Adafruit BNO055 9 DOF IMU with an accelerometer, gyroscope and magnetometer to help with initial State estimation. But, after switching to the iRobot Create 2, we realized that the wheels of the robot were small enough for us to trust the odometry given by the robot. Thus, we decided to exclude the IMU from our final design and just used the odometry data provided by the iRobot for state and position estimation.

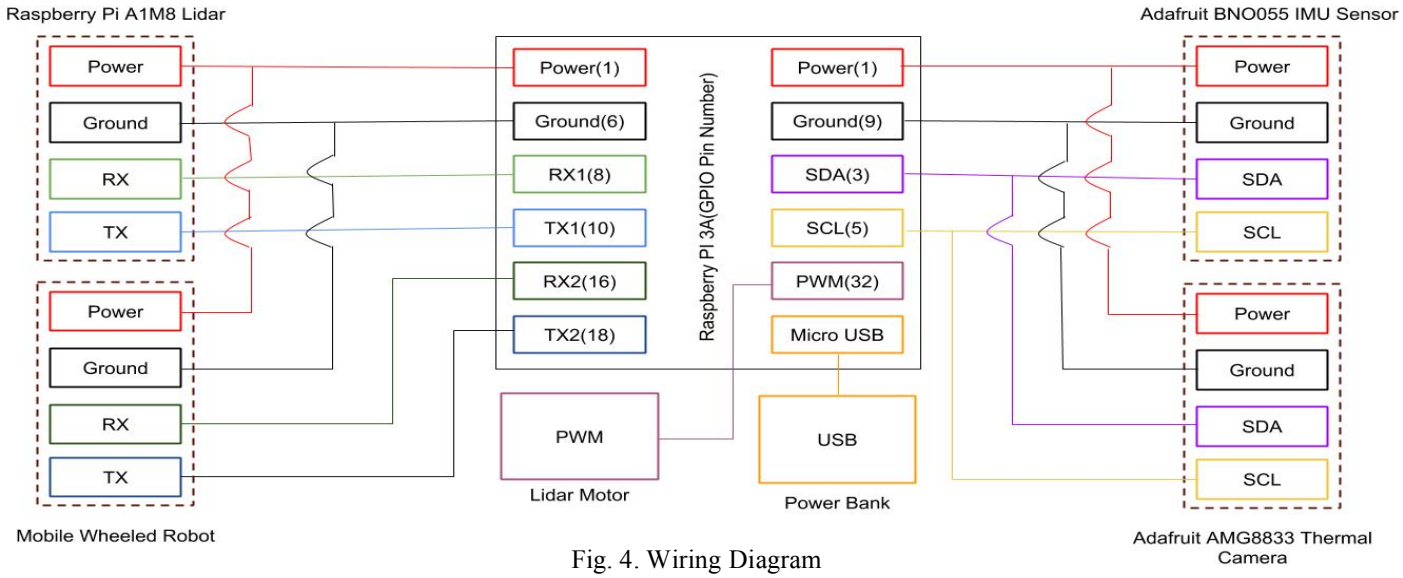



Fig. 4. Wiring Diagram

### *Lidar*

We decided to go with a Lidar over a camera to capture depth information for SLAM primarily because of the computational cost of capturing depth information from a camera. In addition to this the camera would need to either be a 360-degree camera or we would have to rotate it to capture 360-degree information.

There are a variety of extremely powerful Lidar sensors all of which would meet our requirements as we would just need a scan every 2 seconds and a 1-degree resolution. Essentially, we would just have to find the cheapest Lidar sensor that would meet all our requirements. Our exact requirements are listed below:

- Distance Range: 0.2 m - 8 m since we are exploring indoor environments only
- Frequency: 1 scan every 2 seconds
- Field of View: 360 degrees since it avoids having to turn the robot after every movement.
- Resolution: 1 degree, since we require it to be accurate enough so as to not collide with obstacles.

We decided to go with the RPLidar A1M8 as it fulfilled all of these requirements and has a mount which can be easily screwed on top of our mobile robot. Moreover, it only costed 99$ which would mean we could easily buy one more in case this one got damaged. The specifications of the RPLidar A1M8 are given below.

| Distance Range | Field of View | Angular Resolution | Scan Rate |
|---|---|---|---|
| 0.15 - 12 m | 360° | 1° | 5.5 Hz |

### *Thermal Camera*

To detect humans, we decided to use a Thermal Camera instead of a normal camera. This was because, it would be less computationally expensive as we would just get temperature readings and it would be much easier to filter them out. Using a normal camera would mean we would have to use OpenCV for image processing which would put more load on an already strained Raspberry Pi. We decided to use the **Adafruit AMG8833** thermal camera. The thermal camera gives us results in 8x8 which we interpolate to 32x32 to detect humans. After testing it multiple times, we figured out the temperature range for humans and only detected humans when they were in that particular threshold. This would filter out a lot of false positives that we were initially concerned about as we are testing a range.

### *3D Mount*

We created a 3D Mount for our robot that would be placed on top of the iRobot Create 2. This mount would have compartments for the Raspberry Pi as well as the thermal camera. The Lidar would be placed on top of the mount so that it is free to rotate and get information from the environment. A 3D diagram of the mount is given below.

## VII. Software Architecture and Description

An overview of all the software components and how they interact with each other is shown in figure 6. All of the nodes will be implemented using ROS, which handles communication between nodes running in parallel. Below, we provide a description of each of the nodes shown in the diagram.

### *Simultaneous Localization and Mapping (SLAM):*

The SLAM subsystem answers one very important question for the robot - Where am I? It is responsible for generating a map of the environment as well as localizing the robot within it. The choice of our SLAM algorithm was influenced by two key choices we made - map representation and sensor data.

We decided to use Occupancy Grid Maps to represent our environment. The figure below shows an instance of such a map. It consists of uniform cells with each cell containing a probability of it being occupied (0 for unoccupied and 1 for occupied). We use this representation because our end-goal was autonomous navigation and the nature of these maps lends itself very suitably to plan paths and avoid obstacles. For example, given such a map we could easily recognize if the robot is moving to a cell which is occupied and prevent it from doing so.

Secondly, we decided to use a 2D Lidar sensor for extracting depth information from our environment. We chose a 2D Lidar over a 3D Lidar because navigating difficult and uneven terrains was out of the scope of our project. Moreover, 3D lidars generate far more points than a 2D Lidar making processing expensive. Since we want to reduce the time the robot is stationary, we decided against a 3D Lidar. Similarly, the large number of features generated by RGB cameras makes map creation expensive.

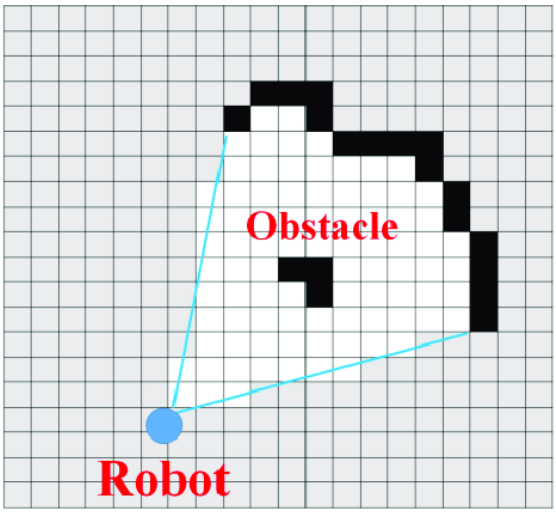



Fig. 5. Occupancy Grid Map

Given these 2 constraints of a Grid Map and 2D Lidar sensor and the fact that we are exploring an unknown environment without any pre-determined landmarks, we decided to use gmapping[2] to implement this node which uses Iterative Closest Point for Scan Matching to improve robot localization and uses a Particle Filter to estimate the probability of every cell being occupied in the grid map.

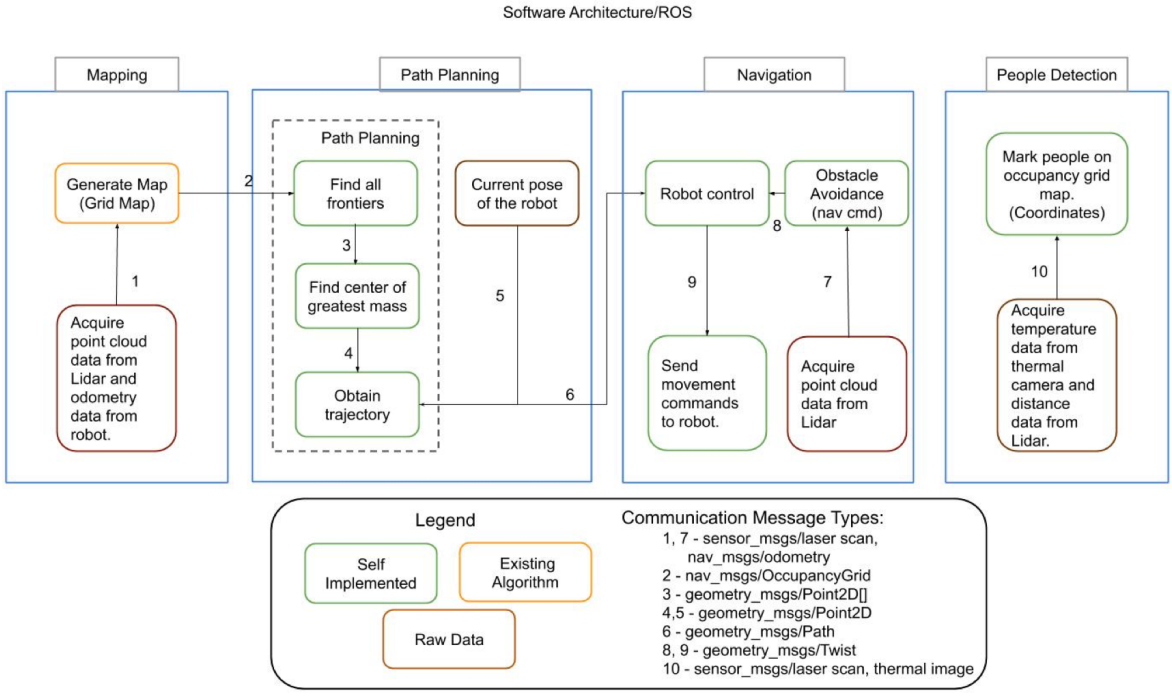



Fig. 6. Software Architecture

## *Navigation*

### ○ Frontier Exploration

Since we are exploring a completely unknown environment, we cannot use algorithms such as wall following to traverse the area since the environment may not have parallel walls or walls at right angles. Moreover, using such a heuristic cannot determine whether we have completed the search or not. As a result, we used the notion of frontiers to explore the environment. By definition, frontiers are cells at the boundary of known and unknown regions. At every iteration, we find clusters of frontiers on the map. We sort these clusters by size and use the heuristic to move to the center of the cluster with the greatest mass (number of frontier cells). Following from the definition of frontiers, once we find no more frontier cells, we have explored all unknown regions of the environment.

There are cases in which frontiers exist but are unreachable. This might be because of noise in LIDAR readings, the LIDAR seeing through obstacles, or very narrow areas in the environment. In order to accommodate for these cases, we check whether the cluster of frontiers is reachable by a path not going through obstacles.

In figure 7, frontiers are marked by orange cells. Notice how these cells exist at the boundary between known (white) and unknown regions (grey) regions.

### ○ Path Planning

Once we have established frontiers on the map, which one should the robot proceed towards? Here, we decided to use the heuristic to move to center of the frontier cluster with the greatest mass.

Once we have this goal and the current position of the robot, we use a traditional A* Search to find the lowest cost path to the goal cell. Each cell has a cost associated with it proportional to whether it is empty or occupied. This search returns a list of robot poses from current position to the goal position. This list is passed on to the robot control which executes the trajectory of the robot until the robot is within the set threshold of the goal cell.

Upon initial testing, we noticed that due to the nature of A* search to find the shortest path, the robot would move very close to obstacles, resulting it in colliding against walls when making sharp turns. As a result, we made two key changes to our implementation. First, we added padding to obstacles on the map so that cells surrounding obstacles are also considered as obstacles themselves. This resulted in the path shifting away from obstacles. Second, we added an extra cost to our heuristic. We add a cost to each cell proportional to its distance to an obstacle up to a certain distance.

The result of path planning with and without these changes can be seen in the figures below.

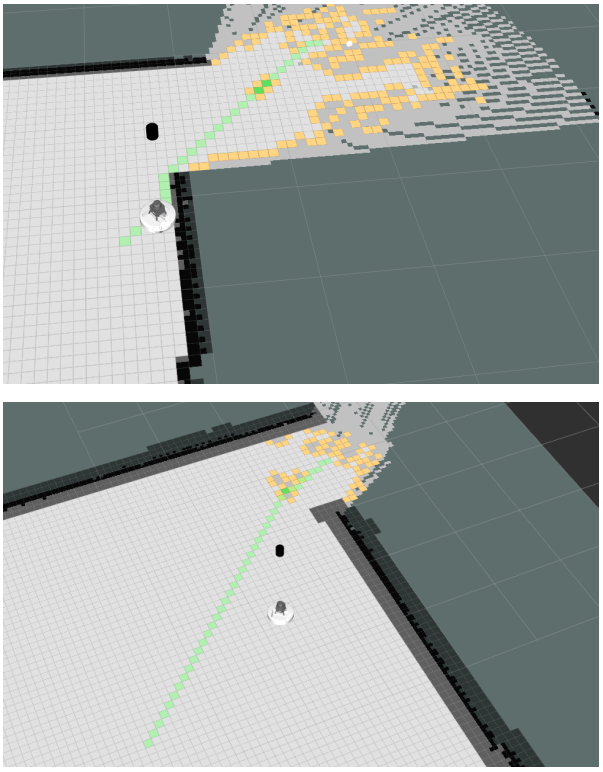


Fig. 7. Planned Path (green) with and without expanded obstacles and heuristic

### ○ Reactive Obstacle Avoidance

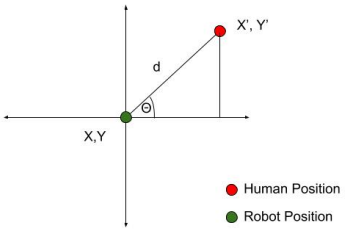
Whenever the robot executes a trajectory, we have a thread that continuously looks through the lidar scans to determine if the robot is going to collide against an obstacle if it executes the most recent control command. If the robot is bound to collide, this thread stops the robot in place and fetches a new trajectory from that point to the goal cell.

## *Human Detection*

For human detection, we decided to use thermal images taken by a camera mounted on our robot. We made this decision because it poses a smaller computational cost as compared to RGB cameras. Since the thermal camera is stationary, we require the robot to make a 360-degree sweeps periodically, allowing us to scan for humans in the vicinity of the robot.

One disadvantage of the thermal camera is that we cannot extract depth information from images since it only outputs an 8x8 image. We overcome this by aligning the center of the Lidar and the Thermal Camera such that the vertical line running through the middle of our thermal images corresponds to the 0th degree range measurement of the Lidar. As a result, when a human is detected, we correlate the angle of the thermal camera to an offset in the Lidar which gives us the distance of the

person from the robot. Given this distance (d), the angle ($\theta$) and the current position (x, y) of the robot, we find the position (x', y') solving the following equations –




$$d^2 = (x - x')^2 + (y - y')^2$$

$$\theta = \tan^{-1}(\frac{y' - y}{x' - x})$$

Another drawback to using a thermal camera is that it cannot discern between humans and other heat emitting objects like radiators or fireplaces. As a result, it is possible to detect multiple false positives. We plan on minimizing this risk by experimenting with temperature thresholds which are most often exhibited by humans. We found that for our thermal camera, temperature ranges in the range 24-29 $^{O}$C.

## VIII. TESTING AND RESULTS

*Test Phase I: SIMULATED TESTING*

**Test Environment:**

→ A total of two test environment are used in this phase to ensure the pathing algorithm works as expected.

→ The 2 test environments are shown in the figures below, with an approximate area of 1300 square feet and 2020 square feet respectively.

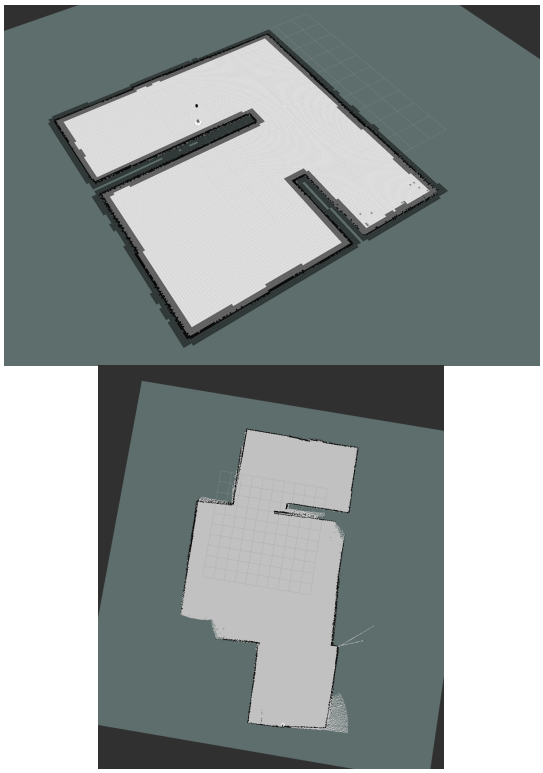


Fig. 8. Simulated Test Environment 1 and 2

**Test Metrics:**

→ The time taken to navigate the simulated area should be below 5 minutes for an area of 1300 square feet and below 7 minutes for an area of 2020 square feet.

→ The proportion of time spent on computation should be below 50% of the entire time taken.

→ The robot should not collide with walls or obstacles at all time.

**Results:**

→ The test metrics are evaluated for ten iterations in each test environment as shown in Table below.

→ From the table below, we see that the average number of iterations taken for test environment 1 of 1300 square feet is 9 iterations and the average number of iterations taken for test environment 2 of 2020 square feet is 16 iterations.

→ The average time taken in test environment 1 is 211.76 s, which satisfy our metrics of below 5 minutes and the average time taken in test environment 2 is 389.7s, which also satisfy our metrics of below 7 minutes.

→ The proportion of time spent on computation for test environment 1 and 2 are 32.5% and 41.0%, respectively, and they satisfy our metrics to keep the computation time below 50% of the entire travel time.

→ The simulated robot did not run into any collision in the ten runs in test environment 1 but did collide once in test environment 2. In order to prevent collisions in real testing, we have expanded obstacles and walls to reduce chance of collision.

| | AVERAGE NUMBER OF ITERATIONS | AVERAGE TIME TAKEN | PROPORTION OF TIME SPENT ON COMPUTATION | RUNS WITHOUT COLLISON |
|---|---|---|---|---|
| TEST ENVIRONMENT 1 | 9 | 211.67 s | 32.5% | 100 % |
| TEST ENVIRONMENT 2 | 16 | 389.7 s | 41.0% | 90 % |

*Test Phase II: REAL WORLD TESTING WITHOUT HUMAN DETECTION*

**Test Environment:**

→ A total of four test environment are used in this phase to ensure the robustness of the path planning algorithm integrated with every component.

→ The first test environment we used is the kitchen and hallway in an apartment with an approximate area of 150 square feet as determined by The RViz map and shown in figure below.

→ The second test environment we used is the hallway in

an apartment with an approximate area of 650 square feet as determined by the RViz map and shown in figure below.

→ The third test environment we used is the right wing on the first floor of Hamerschlag Hall. It has an area of approximately 1000 square feet as determined by the RViz map.

→ The final test environment resembles the area we constructed for public demo. It is 16 feet by 16 feet area, with a large room, a passage, and two small rooms and shown in figure below.

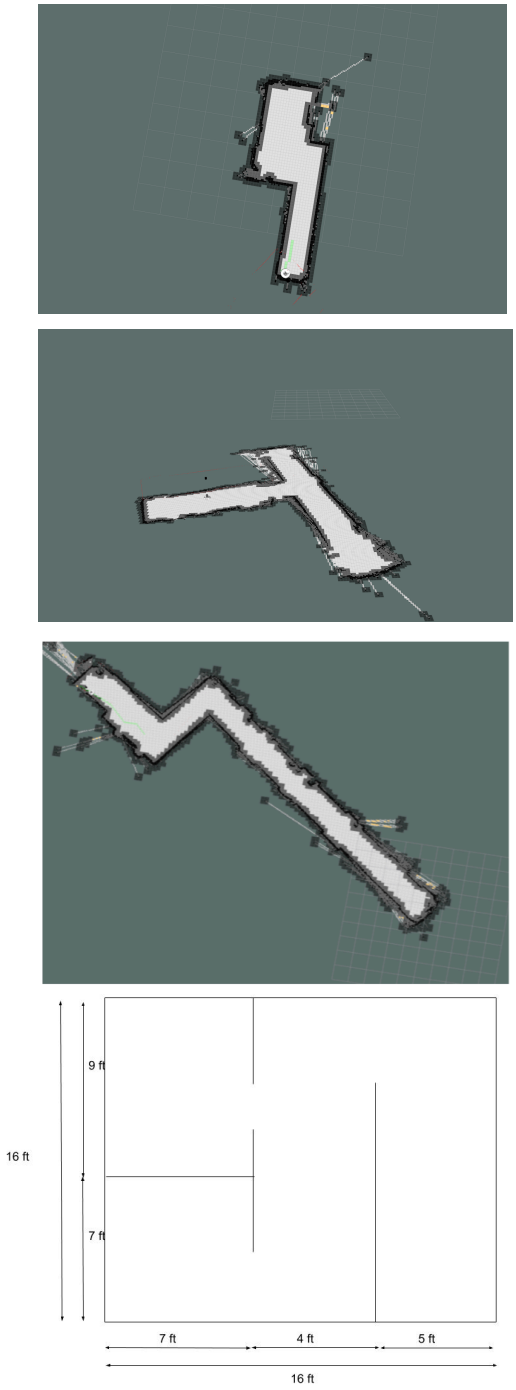



Fig. 9 Test Environments 1, 2, 3, 4

**Test Metrics:**

→ The time taken to navigate the simulated area should be below 2 minutes for an area of 150 square feet, below 3 minutes for an area of 256 square feet, below 5 minutes for an area of 650 square feet, and below 8 minutes for an area of 1000 square feet.

→ The proportion of time spent on computation should be below 50% of the entire time taken.

→ The robot should not collide with walls that causes failure at least 80 percent of the time.

**Results:**

→ The test metrics are evaluated for ten iterations in each test environment as shown in Table below.

→ From the table below, we see that the average number of iterations taken for test environment 1 of 150 square feet is 3 iterations and the average number of iterations taken for test environment 2 of 650 square feet is 6 iterations.

→ The average time taken in test environment 1 is 93.8s, which satisfy our metrics of below 2 minutes and the average time taken in test environment 2 is 247.7s, which also satisfy our metrics of below 5 minutes.

→ The proportion of time spent on computation for test environment 1 and 2 are 65.4% and 57.0%, respectively, and they do no satisfy our metrics to keep the computation time below 50% of the entire travel time.

→ The rumba ran into collisions twice in the ten runs in test environment 1 and collided three times in test environment 2. Although the algorithm expanded the obstacle and walls, adding cost to path going near obstacles, it still appears that the localization of the robot can deviate from the actual pose of the robot that lead to collisions. However, the robot generally is able to recover from collision and re-plan its path forward.

| | AVERAGE NUMBER OF ITERATIONS | AVERAGE TIME TAKEN | PROPORTION OF TIME SPENT ON COMPUTATION | RUNS WITHOUT COLLISION |
|---|---|---|---|---|
| TEST 1 | 3 | 93.8 s | 65.4% | 80 % |
| TEST 2 | 6 | 247.7 s | 57.0% | 70 % |
| TEST 3 | 8 | 290.0s | ~60% | 70% |
| TEST 4 | 5 | ~190s | ~70% | 90% |

*Test Phase III: REAL WORLD TESTING WITH THERMAL DETECTION INTEGRATED.*

**Test Environment:**

→ A total of two test environment are used in this phase to ensure that the updated pathing algorithm as well as thermal camera detection works as expected.

→ The first test environment we used is the kitchen and hallway in an apartment with an approximate area of 150 square feet as determined by The RViz map and shown in figure below.

→ The second test environment resembles the maze we constructed for public demo. It is 16 feet by 16 feet maze, with a large room, a passage, and two small rooms and shown in figure below.

**Test Metrics:**

→ Same as Test Phase 2.

→ The robot should detect at least 90 percent of people in the unknown environment.

→ The difference between the marked and actual position of the humans detected should be within 1 ft.

**Results:**

→ The test metrics are evaluated for ten iterations in each test environment as shown in Table below.

→ From the table below, we see that the average number of iterations taken for test environment 1 of 150 square feet is 3 iterations and the average number of iterations taken for test environment 2 of 256 square feet is 5 iterations.

→ The average time taken in test environment 1 is 127.2s, which satisfies our metrics of below 3 minutes and the average time taken in test environment 2 is 242.0 s, which also satisfies our metrics of below 5 minutes.

→ The proportion of time spent on computation for test environment 1 and 2 are 48.2% and 43.3%, respectively, and they both satisfy our metrics to keep the computation time below 50% of the entire travel time since the extra rotation times added in our algorithm reduces the percentage of total time spent on computation.

→ The rumba run into collisions twice in the ten runs in test environment 1 and collided twice in test environment 2. Although the algorithm expanded the obstacle and walls, adding cost to path going near obstacles, it still appears that the localization of the robot can deviate from the actual pose of the robot that lead to collisions. However, the robot generally is able to recover from collision and re-plan its path forward.

→ The overall percentage of people detected in the first test environment is approximately 83%. The thermal camera can only credibly identify humans within around a 1m range. it was not able to meet the metrics of identifying at least 90% of humans. The overall percentage of people detected in the second test environment is approximately 70%. The robot moves into a room and the lidar is able to map the entire room with high likelihood due to the relatively small size of the room, so it sometimes misses humans standing at the side of the wall due to limited range of thermal camera.

→ The error between marked and actual position of humans detected is around 0.8 ft and 1 ft for test environment 1 and test environment 2, respectively. It satisfies the test metrics that the different between the marked and actual position of humans detected should be within 1 foot.

| | AVERAGE NUMBER OF ITERATIONS | AVERAGE TIME TAKEN | PROPORTION OF TIME SPENT ON COMPUTATION | RUNS WITHOUT COLLISON | PERCANTAGE OF PEOPLE DETECTED | ERROR BETWEEN MARKED AND ACTUAL POSITION |
|---|---|---|---|---|---|---|
| TEST ENVIRONMENT 1 | 3 | 127.2 s | 48.2% | 80 % | 83% | 0.8 ft |
| TEST ENVIRONMENT 2 | 5 | 242.0 s | 43.3% | 80 % | 70% | 1 ft |

Another issue we found is that in large test environments, the time taken by the robot to finish exploration had anomalies in which some runs would take 60-90s more to finish. This occurs because residual frontiers are left behind in areas the robot has already explored causing it to re-explore those areas. We think this might be happening due to noise in LIDAR data or the LIDAR not being able to get the rays back from some surfaces.

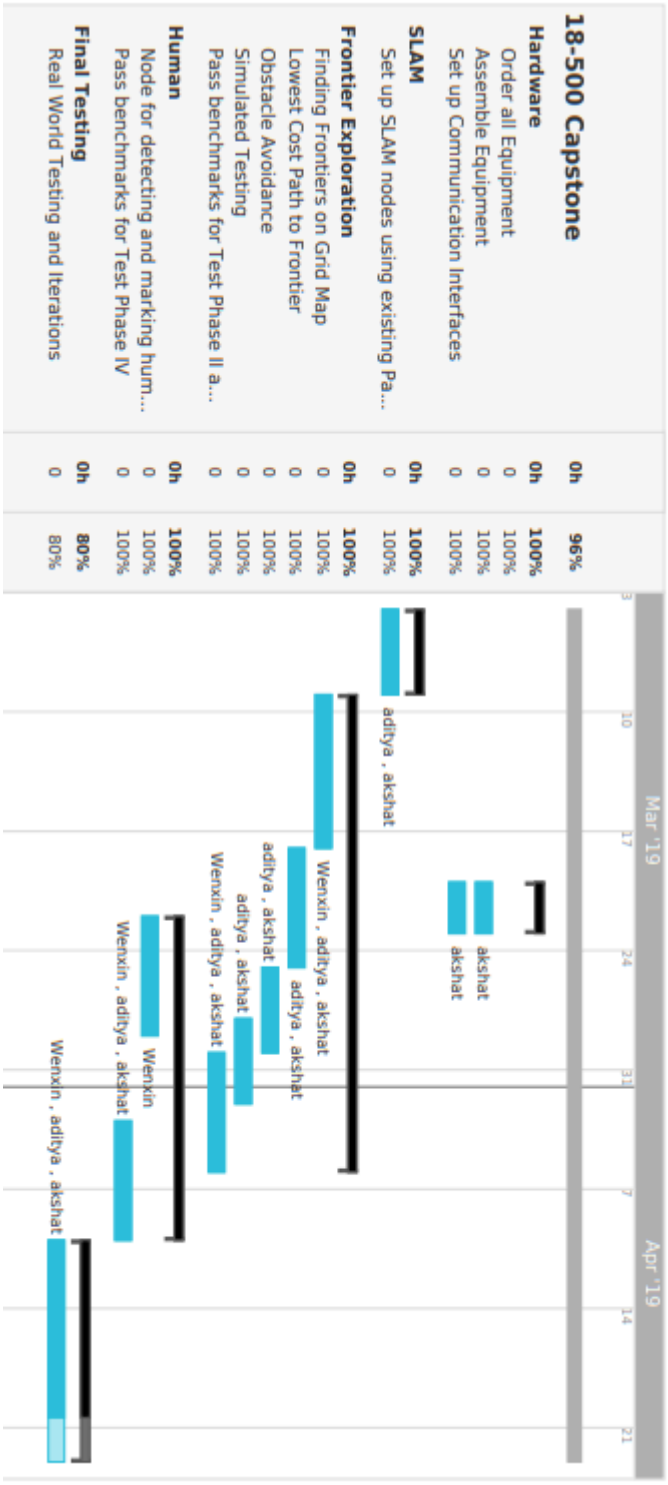
Moreover, the time taken by the robot is bounded not only by the size of the environment but also by the structure since that determines how many computation iterations are needed.

## IX. Project Management

### *Budget*

| ***Component*** | ***Price*** |
|---|---|
| iRobot Create 2 | $162.00 |
| Adafruit AMG8833 IR Thermal Camera | $39.95 |
| Power Bank | $34.95 |
| RPLidar A1M8 | $99.00 |

The Gannt chart below shows our schedule for the project.




## X. Summary

Overall, our results and demo show that the capability of robot to fully explore unknown environments is very robust to changes in size and structure. On most occasions, the robot is able to start at any location and completely map the environment. On the other hand, due to the limited range of the thermal camera, there exists a considerable probability that the robot will not detect some humans in the area.

Going forward, we think we could improve the human detection subsystem by using a Thermal Camera with a longer range or by switching to an RGB Camera instead. Moreover, another improvement that could be made is reducing computation times. This could be done by making optimizations in how we search for frontiers. Instead of repeating the search every iteration, we could store frontiers and mark them as seen using an additional check.

## XI. References

[1] Brian, Yamauchi, *A Frontier-Based Approach for Autonomous Exploration,* https://pdfs.semanticscholar.org/9afb/8b6ee449e1ddf1268ace8efb4b69578b94f6.pdf

[2] https://openslam-org.github.io/gmapping.html