18-500 Search and Rescue Robot

Authors: Raymond Xiao, Keshav Sangam, Jai Madisetty
Affiliation: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—A system capable of autonomously assisting firefighters detect human presence in an unknown environment. This system will allow for firefighters to quickly detect human presence via a remote notification from the robot. Using mmWave technology to assist with detection in smoke-filled environments is too expensive for the scope of this project; thus, we will ignore that aspect of the problem and use CV instead.**

***Index Terms*—autonomous, iRobot, OpenCV, LIDAR, SLAM, NVIDIA Jetson Xavier NX, path planning, robotics, SAR**

1 INTRODUCTION

The Search and Rescue Robot originated from the initial idea of developing a robot that a rescue team would be able to deploy in order to pinpoint the exact location of a lost person(s) in dangerous environments like forests or mountain ranges, for example. However, to narrow the scope of our project, we decided to focus in on building fires.

Building fires are not the same as house fires. During house fires, firemen can simply access their truck and obtain what they need; however, this would not be the case on the 15th floor of building fire and as a result, are a lot more stressful and dangerous for firefighters. One major responsibility for firefighters is to accumulate any victims to one safe floor where care will then be transferred.

Firefighting is a very dangerous profession, and a lot of research and data show the contributions that the profession exposures have in chronic illnesses, such as cancer and heart disease, and in behavioral health issues that may end in suicide [1]. In addition to these long-term health issues, firefighters face serious risks on the scene of a fire as there is exposure to various combustion products. In addition to harsh chemical exposure, firefighters have to deal with oxygen depletion from the air which can result in a loss of physical performance, confusion and inability to escape [6]. It appears that reducing the amount of time spent in these dangerous environments would improve the well being of those who put their lives on the line.

SAR is a completely autonomous robot that can quickly navigate unexplored rooms and relay human presence and even vital signs to a third party in real time. Such a robot would facilitate searching empty rooms or floors, potentially resulting in a significant reduction in the amount of time firefighters would spend in these dangerous environments. Firefighters would be able to deploy a number of these robots throughout a burning building, ideally one per floor, and have the robots scout ahead and provide valuable real time information. Additionally, this has potential to speed up rescue of those caught in building fires. Firefighters would not have to waste time searching empty rooms, and would only search rooms that contain humans.

The closest competing technology to SAR is Thermite, which is a robotic firefighter. It is designed to provide fire suppression, situational awareness, and intelligence gathering to first responders. They are user-controlled, and users are provided a real-time video feed [8]. However, unlike SAR, this robot is more meant for outdoor support rather than to be able to search buildings.

The requirements for our solution were chosen with the intention of making SAR able to operate in as close to real conditions while still staying in our budget. Our primary requirements are the following:

1. Completely autonomous robot that can create a map of its environment
2. 0% false negative rate (always detect human presences)
3. Light-weight, efficient, and fast

2 USE-CASE REQUIREMENTS

The solution has various main use case requirements. The use case requirements listed below are necessary to guarantee a minimum level of system performance.

1. One of the main goals of SAR is to lighten the rigorous workload of firefighers. Thus, we do not want firefighters to be responsible for controlling the robot. The robot must be able to navigate environments 100% autonomously with no assistance from external sources. This way firefighters can go about putting fires out while the SAR robot(s) is searching for humans. This will be achieved through path planning and SLAM algorithms.
2. The robot must be able to locate humans with 100% false negative since no recognition could be the difference between life and death. However, false positives are far less of an issue, since you can never be too careful. This will be achieved with a robust CV algorithm.
3. The solution must be able to detect one human presence per room with an area of approximately 10m^2 in under 25 seconds. This number was determined through our desire for high speeds. Although 25 seconds is not a very tight bound, this project is merely a first step.

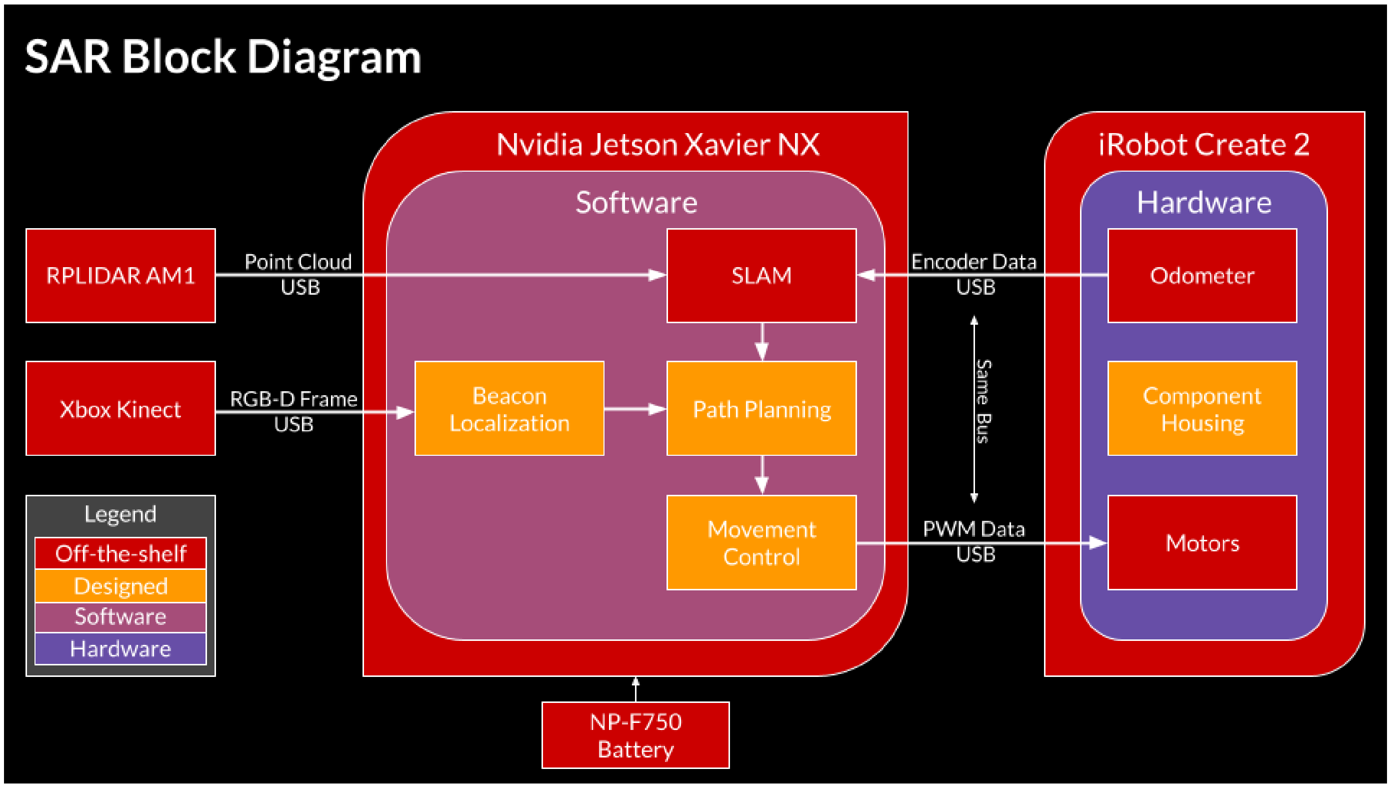

Figure 1: Block Diagram of Entire System

4. The entire system's weight must not exceed 4.5 kg since weight directly correlates to higher energy consumption and lower speed. This limit is mainly derived from the combined weights of the iRobot, NVIDIA Jetson Xavier, battery pack, and depth camera.

5. The battery life of the system must exceed 15 minutes. This is so the robot can navigate through multiple rooms on a single charge.

3 ARCHITECTURE

The architecture of the SAR consists of image processing, robot automation, and robot control. All the software we develop will be hosted on the Nvidia Jetson Xavier NX.

There are three primary sensors that we will be utilizing. All of the sensors are used as inputs to at least one block of software we are developing. Discussion of the software will occur later in the report. The first of which is a lidar, specifically Slamtec's RPLIDAR AM1. The lidar's primary function is to provide real time sensor information for our Simultaneous Localization and Mapping (SLAM) algorithm. Next, we are employing an Xbox Kinect, which is essentially a camera that provides the classic RGB color information of an image frame, alongside a depth map of the frame. This will be referred to as an RGB-Depth or RGB-D frame. The purpose of the RGB-D frames is to inform our visual beacon detection algorithm. Finally, we have the odometer, which comes pre-packaged with the iRobot Create 2. The odometry data is given by the iRobot's two wheel encoders, encoders being sensors which can detect and enumerate the revolutions of a wheel. This will again be used to provide information for the SLAM algorithm.

On the hardware side, the iRobot Create 2 will house all the equipment necessary for the robot to function. These components include all the off-the-shelf parts seen in the diagram; specifically, the RPLidar AM1, the Xbox Kinect V1, the NP-F750 battery that will be used to power the Nvidia Jetson Xavier, and of course the Nvidia Jetson Xavier itself.

The software of SAR consists of two major parts. There is the robot control subsystem, consisting of SLAM (informed by lidar and odometry), path planning (informed by the kinect), and movement control. The robot control subsystem will be built using the open source Robot Operating System (ROS) infrastructure. There is also the beacon detection subsystem, also known as the human analogue detection subsystem, consisting of computer vision and image processing.

The beacon detection subsystem utilizes RGB-D frames and a known beacon format to detect and determine the distance of beacons in the environment. The detection will be performed using a combination of OpenCV functions and our own algorithms. Currently, we are exploring a variety of beacon types that each have a different detection method. AprilTags or neon colored blocks are examples of two such beacons. Detecting the beacons will involve filtering out erroneous detections. Finally, once the beacon is detected, the RGB-D frames from the Kinect will be enough to determine the distance of the beacon from the robot. This information will be fed into the path planning algoritm.

As aforementioned, the robot control subsystem has three major blocks. First, we will discuss SLAM. The two inputs for SLAM are the lidar point cloud data and the odometer data. In order for the robot to autonomously move through its environment, it first needs to understand how the environment is structured. Only then can it effectively plan a route through the environment. The lidar provides data in the form of a point cloud. Each point in the point cloud is a 3D coordinate describing the location of a point in the environment relative to the lidar sensor. By pre-processing and interpolating the data, the SLAM algorithm can create a map of the environment. The SLAM algorithm also solves a correspondence problem between lidar point cloud packets (i.e., it determines which points from one packet are the same as which points from another packet). By doing this, the SLAM algorithm can get a general sense of where the robot is located in the environment and how the robot is moving in the environment. This localization is refined with odometry data. SLAM algorithms we are considering include the ROS packages for

Next, there is the path planning algorithm. Now that we have a map of the environment and a way to detect beacons, we need a way to autonomously plan a route through the environment. This involves two different algorithms, for exploration and for waypoint navigation. Exploration is necessary in order to fully search the provided environment. While exploring, if the beacon detection system finds a new beacon, the robot must switch to the waypoint navigation algorithm to approach the beacon. The reason we are approaching the beacon rather than just detecting and reporting the location of it is to simulate an automated vital sign detection program, as motivated by our use case. Both algorithms will likely be the same graph traversal algorithm (i.e., Djikstra or A* or similar), but their end node will motivated by the type of navigation the robot is pursuing. In particular, the end node of the exploration algorithm will be the unmapped region of the environment, whereas the end node of the waypoint navigation algorithm will be the beacon.

Finally, once the robot knows what route to take, it actually needs to move along the path. This will be done by the movement control block. This block involves providing each iRobot motor a linear velocity, in the form of a pulse-width modulated signal. By changing the ratio between the velocities of each motor, we can make the robot move in any direction and turn to face any direction.

Finally, the Jetson itself will be powered by an external battery pack on-board the iRobot Create. The Jetson will provide power to the lidar via the USB bus. The iRobot Create 2 will be powered via its internal battery pack.

4 DESIGN REQUIREMENTS

To develop a robot that will be able to operate in as close to real conditions as possible, we will need to steer as close to our design requirements as possible.

The weight of the robot is 3.583 kg. Accounting for the weight of the Jetson Xavier (.024 kg) and the sensors (.051 kg), the approximate total weight is 3.658 kg. The average speed of the robot base with all accessories attached is around 0.6 m/s. Our tentative current environment size is 10 m^2.

1. *Complete autonomy with 100% false negative rate*: In order to guarantee complete autonomy while also guaranteeing all rooms with humans will be flagged, we need to ensure the robot will not get stuck trying to map out a certain portion of the room, get lost while looking for a certain beacon, or enter an infinite loop while trying to map an unknown environment. Thus, anytime SAR is looking for a beacon for more than 15 seconds, we will switch the search algorithm from waypoint to explore. 15 seconds comes from the constraint that the robot can detect beacons from up to 8 meters away and given the average speed of the robot is 0.6 m/s, we can compute the max time it will take for the robot to approach the beacon. Thus,

   $$8m/(0.6m/s) = 13.3 \text{ s} \tag{1}$$

   time should be more than enough time for the robot to approach the beacon before we switch to exploration to prevent anymore wasted time.

2. *Detection (or lack thereof) within 25 seconds*: In order to detect human presence, in the worst case, we would need to search every square meter of the room. The average speed of the robot is 0.6 m/s, and LIDAR can generate point cloud data from 8 meters away. Thus, having the robot rotate in place in the center of an empty room during exploration should generate a full point cloud map, since the maximum distance from the wall to the lidar is the diagonal of the room:

   $$\text{distance} = 5\sqrt{2}\text{meters} \tag{2}$$

   Assuming that the angular velocity of the robot is the same as the linear velocity, we can estimate that a full rotation will take

   $$\frac{2\pi \text{ radians}}{0.6 \text{ radians/second}} = 10.5 \;\; \text{seconds} \tag{3}$$

   We decided to allot 0.25*10 s per corner for potentially navigating the corners behind obstacles and an extra 5 s for slack.

3. *Weight must not exceed 4.5 kg*: We intend to maintain a speed of approximately 0.6 m/s, so we can search compromised rooms quickly. The iRobot and all other components weigh 3.6 kg. We found 4.5 kg total weight to be the point at which the iRobot slows

down a sizable amount, thus, we aim to not exceed 4.5 - 3.6 kg = 0.9 kg for any structure keeping the components in place.

4. *Battery life must exceed 15 minutes*: If we were to launch one SAR robot per floor, where we would expect 10 rooms a floor, we would need to search within 25 s and then find another room within a short period of time. If finding and searching a room takes 40 s, we would expect SAR to finish searching a floor within approximately 7 minutes. This expectancy is doubled for slack.

Rest of page intentionally left blank.

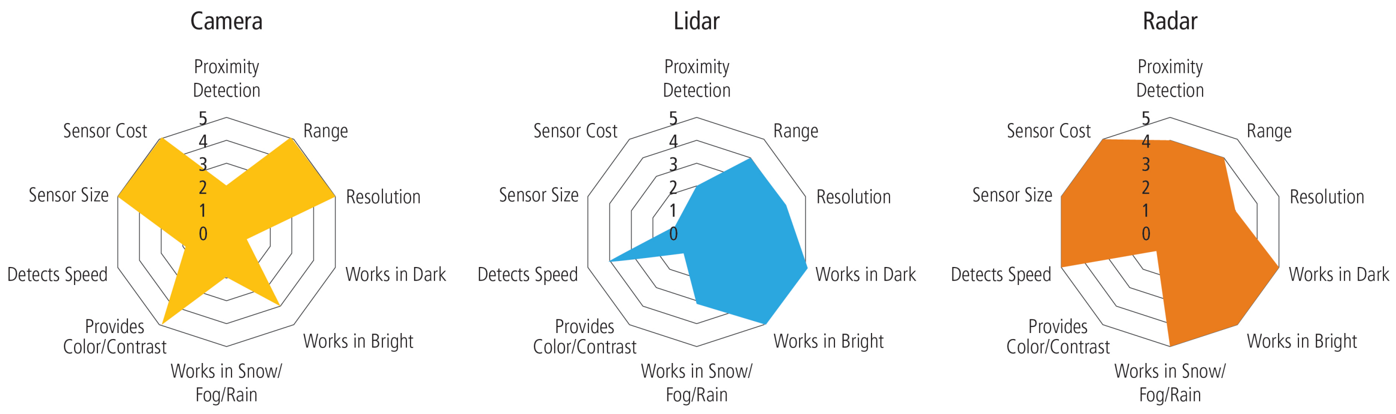

Figure 2: Trade-offs between lidar, radar, and cameras

5 DESIGN TRADE STUDIES

We have and are exploring multiple approaches for the various blocks we are designing. These different approaches came with their pros and cons that we must balance. Included below is an analysis of these different trade-offs

5.1 Robot Base

In our final design, we have elected to use the iRobot Create 2 base. Our original plan was to use the iRobot Create 1 as our base, due to the iRobot Create 1 having decent ROS support, but also because there was already one in the ECE inventory. After realizing that the iRobot Create 1 was not powering on, we decided to re-look at our options. One possiblity was designing our own custom robot base. The benefits of this included having control over the mechanical parameters of the robot, and the ability to better fit our use-case by creating a frame that was appropriately heat and smoke resistant. However, none of our group have the mechanical engineering experience necessary to design such a robot, so we opted to buy a prebuilt base. In exploring other options for pre-built bases, we settled on the iRobot Create 2 for a few reasons. The first is due to its extensive ROS support (better supported than even the Create 1). Though it is well supported, the iRobot Create 2 is still not the most supported base; for example, the Turtlebot is the de facto base for ROS simulations and in-the-wild implementations. Another critical reason is that by using the iRobot Create 2, we maximize the backwards compatibility between the software we already wrote for the Create 1 and the software we need to write for the Create 2.

5.2 SLAM Primary Sensor Input

The primary input for our SLAM algorithm is critical to ensure good robot autonomy. There are three options to consider: lidar, radar, and depth cameras. The trade-offs can be summarized in the Figure 2. Although radar would technically work the best for our scenario due to its ability to be accurate in conditions like fog (which is akin to smoke) and its insensitivity to lighting, there were a few reasons we decided to settle on lidar. The first is that radar-informed SLAM is currently a research area, and so there's little documentation on how to use it for SLAM, and even less information on using radar with ROS. The second is that the radar costs $300, which makes investing in one far too risky. We could have also used depth cameras, but in order to maximally satisfy our use case requirements, we would want our sensor for SLAM to be as invariant to weather and lighting conditions as possible. This leaves lidar, which provides a good balance of everything.

5.3 Beacon Localization Sensors

The type of beacon and how to localize said beacon had many design trade-offs. The goal of the beacons is to strike a good balance between faithfully satisfying the use-case while also ensuring consistency and realism. There were three options we considered throughout our design process: Bluetooth Low Energy (BLE) beacons, Ultra-Wide Band (UWB) beacons, and visual becaons. The benefit of BLE beacons is that they are prebuilt and proven technology for localization. Furthermore, BLE beacons are realistic in that smartphones can act as Bluetooth beacons, which makes the human detection use-mase more grounded. However, this option comes with several drawbacks. The first is that BLE beacons can only localize to within a radius of around 8 meters. This is simply not accurate enough for our purpose. We then looked to UWB beacons, another pre-built technology that offers an impressive accuracy of within 30 centimeters. However, this was not realistic enough for our liking; there's no reason for a human to have an UWB beacon on them. Finally, we settled on visual beacons, and detecting them using our own computer vision process. Visually identifying a beacon can realistically be scaled to visually identify a human. Furthermore, by employing a depth camera, we can localize the beacon extremely accurately. Although cameras are very sensitive to lighting conditions, we figured that it was a worthy trade-off for our purposes. The lidar for SLAM is meant to be as invariant to such conditions as reasonably possible given the constraints we are working under, so using the depth-camera for this purpose seemed reasonable to us.

5.4 Computer

The Jetson Xavier was chosen since this compute platform contains a 6-core CPU and 384-core GPU which allows for real time computer vision and path planning. The computer vision algorithms would benefit massively from the GPU. Using an Arduino or Raspberry Pi would not provide enough compute power since there is no dedicated GPU in these systems, which is necessary to accelerate our computer vision and our SLAM algorithms. The maximum rated power consumption of this system is 30 Watts.

5.5 Algorithm Selections

This final section is dedicated to the various pre-made implementations of SLAM, in addition to our choice of path planning algorithms. All of these trade-offs are yet to be explored, so we are discussing how we will eventually compare these approaches The ROS infrastructure provides a rapid way of testing out different SLAM and path planning techniques. Because of this, we can easily test out various implementations. Regarding path planning, we are likely using either Djikstra or A* for both types of path planning (exploration and waypoint navigation). Since Djikstra can be reduced to A* without a heuristic, we can easily test the efficacy of each implementation in ROS simulations and in the real world. We have decided to not create our own SLAM algorithm due to the complexity of the software and the timeline of our project. Thus, we will be testing the real-world efficacy of the most popular implementations; in particular, we are looking at the Hector-SLAM, Gmapping, and Cartographer ROS SLAM packages [3][4].

6 SYSTEM IMPLEMENTATION

As shown in Figure 1, our system is comprised of a few main subcomponents: the NVIDIA Jetson Xavier NX, the software on the Jetson, the RPLIDAR AM1, the Xbox Kinect, the iRobot Create 2 system, and the NP-F750 battery. The software will be run on the Jetson. Main software responsibilities include extracting data from the LIDAR and Xbox Kinect as well as interfacing with and controlling the iRobot system. The USB interface will be utilized for all of these peripherals. This software also includes the beacon detection subsystem which will process sensor data using OpenCV and use that data to locate human presences in a given room. The following is a detailed implementation plan of each subsystem.

6.1 Beacon Localization

The beacon localization algorithm reads RGB image frames from the Xbox Kinect v1 and extracts the location of the beacon from this sensor. To accomplish this task, the read images will go through a processing pipeline. The exact details of this processing pipeline are yet to be determined, due to us settling on using a visual beacon within the past 5 days. However, we can generalize the steps as follows:

6.1.1 Filtering

Before we can accurately detect our beacon, we will likely need to go through a filtering stage. The exact type of filtering will depend on the type of beacon we decide to use. One type of beacon used could be something akin to an AprilTag. By using an AprilTag we don't have to worry about color filtering. However, a beacon that is simply a neon color would have to go through a color filtering phase. In order to accurately detect the beacon, we transform our RGB images into HSV (Hue Saturation Value) images. By transforming our images into the HSV color space, we can take advantage of hue being a single channel to easily find a range of colors that our beacon lies within, without worrying about the saturation or the brightness of that color.

6.1.2 Denoising

If we use a color beacon, we will also have to go through an denoising process. Erroneous pixels that happen to have the correct hue value will get through our filtering and thus be incorrectly detected as a beacon. To prevent this, a simple low-pass/blurring filter operation can be done, which will remove most noise from the image. This comes with the drawback of effectively lowering the resolution of the image we get, but we don't need the improved resolution in order to best detect the beacons.

6.1.3 Detection

The detection is relatively simple. With a color beacon, we simply pre-filter the image and search for the color of our beacon. With an AprilTag, we can apply a built-in OpenCV function to determine where in the image frame our beacon is. Based on these results, we can use the depth information from the Kinect to accurately determine where the beacon is.

6.1.4 Persistence

One concern with the algorithm is persistence of beacon localization. For example, let's say the robot enters a room, and can initially find the beacon. If the robot then crosses behind a barrier that obstructs the direct line of vision to the beacon, it must remember where the beacon was. To do this, when first detecting a beacon, we will save its location in our SLAM map. While the robot has not approached the beacon, the path planning algorithm will use the saved SLAM map location to navigate to the beacon without direct line-of-sight being necessary. Finally, when the robot is suitably close to the beacon, we will inform the SLAM map to mark the beacon as visited. This way, we can not only remember where a beacon is within a room, but also remember if we have already found the beacon previously.

6.2 SLAM

In this section, we will discuss the exact implementation differences between two popular ROS Slam packages: Hector-SLAM and Gmapping. Those two algorithms differ from the information sources used for localization and mapping. GMapping uses odometry and lidar point cloud data, whereas Hector SLAM uses the lidar only. Theoretically, GMapping should perform better then Hector SLAM, especially on environments that cause lidar-based localization to be ambiguous, such as a long hallway without features. The reason such environments would cause problems with Hector-SLAM is that lidar based localization solves a correspondence problem between points in one lidar packet vs. points in the next. By determining which points correspond to which, the algorithm can get an estimate of how far the robot has moved. In a hallway environment, there is a huge number of points from the second frame that could theoretically map to a single point in the first frame. Because of this trade-off, Hector SLAM theoretically has poor performance when it comes to linear movement in large featureless areas, though it will still be able to compute angular motion with a high degree of accuracy. In this scenario, GMapping can rely on odometry to get a bearing on the linear motion of the robot. However, real-world tests show that even with the extra information that GMapping is receiving, it often still performs worse than Hector SLAM [2]. Since these are pre-designed packaegs, our implementation plan for this subsystem revolves around testing the efficacy for our robot in particular.

6.3 Path Planning

In this section, we will discuss the implementation differences between two shortest path algorithms: Djikstra's algorithm and A*. These two algorithm's differ as A* uses heuristics to optimize search using Djikstra's algorithm. Since edge weights from one node to another will not be consistent due to obstacles, we will use A* as our path planning algorithm with an approximate straight-line distance (from node to beacon) as a heuristic function. We will add a number of potential nodes to search from to a priority queue with the combined edge weight and heuristic value. As a result, we will prioritize nodes that are most likely to produce the optimal path.

6.4 iRobot Create 2 Control System

The iRobot Create 2 is a prebuilt robot that contains all the components necessary for customization. We decided to go with this robot since it provides flexibility at a relatively low cost. Designing and building our own robot would be better since we would be to add specific sensors and modify it to our needs, but due to time and budget constraints, we are purchasing a prebuilt system. In addition, the iRobot has an Create 2 Open Interface that allows us to easily communicate with it using UART. As an example, to drive backwards at a speed of 200 mm/s with a turn radius of 500 mm, we could send 137 (the opcode for the drive command), 255 (the upper byte of the velocity), 56 (the lower byte of the velocity), 1 (the upper byte of the turn radius, and 244 (the lower byte of the turn radius).

6.5 Power

To power our Jetson, we are using a NP-F750 Rechargeable Battery. This battery is made of lithium ion and has an operating voltage of 7.4V. It's maximum capacity is 5200mAh. Based on the Jetson Xavier's 30W power rating, this battery will provide approximately 76 minutes of usage before recharging is required. The iRobot Create 2 base will use its built-in power supply. According to the specs, this internal battery can last up to 2 hours given 0 load. This should definitely be enough to run the robot with a light load (caused by the Jetson and the sensors) for at least 30 minutes.

7 TEST & VALIDATION

Since our project has multiple components, thorough testing and validation is required to ensure that our system works. We will be testing both the hardware and software components separately as well as together.††

7.1 Tests for Complete Autonomy

To test whether the SLAM and path planning algorithms works effectively, we will activate the robot in a series of different environments. To measure the individual accuracy and performance of each part, we will do the following:

- *Mapping*: Subjectively compare the error between the true environments and the mapped environments. If the algorithm performs well enough, the differences between the top-down view and the scan should be minimal.

- *Localization*: In each environment, set the robot to follow a path with known starting and ending points. After letting the robot traverse between the points, measure the distance between the robot's calculated endpoint within the map and the true endpoint. We will consider the localization to be a success if the robot is within a 1 meter radius of the true endpoint.

- *Exploration*: First we will disable the waypoint navigation portion of the path planning. We then let the robot run exclusively on explore mode through the various environments. If the SLAM map at the end covers more than 90% of the true area, we will consider it a success.

- *Waypoint Navigation*: Disable the exploration portion of path planning. Place the robot in a series of different individual rooms, with various obstacles that may or may not block direct sight-lines to the beacon. If the robot can successfully navigate towards the beacon 100% percent of the time, we will consider it a success.

7.2 Tests for Beacon Localization

This test is similar to that of waypoint navigation. Place the robot in a series of different individual rooms, with various obstacles that may or may not block direct sight-lines to the beacon. If the robot can successfully detect (rather than navigate towards as in the waypoint navigation test) the beacon 100% of the time, we will consider it a success.

7.3 Tests for Battery

- *Weight*: We will simply measure the end weight of the robot to ensure that it is under 15 pounds. This should definitely be achievable given that the iRobot base is around 10 pounds and the sensors and Jetson are not very heavy.

- *Battery Life*: Run the robot under full load until the battery depletes. If it lasts greater than 15 minutes, we will consider it a success. If the battery life is shorter than this, a battery with a higher energy capacity will be used.

7.4 Tests for Xbox Kinect

To ensure that the Xbox Kinect is working as intended, we will connect it to the Jetson Xavier via USB. We will interface with the Kinect using OpenCV and read the RGB frames and depth map that it is sending. This data can be verified both visually and by writing a suite of tests.

8 PROJECT MANAGEMENT

8.1 Schedule

Our project has been broken down into 4 components: proposal and planning, implementation and design, verification and optimization, and finalization. The first three weeks of this semester were spent finalizing the idea and gathering all the necessary components. During this time period, we also explored various design trade offs such as using a Jetson versus an Arduino. During the second phase, the subsystem components will be implemented and integrated into one cohesive unit. The third stage will then begin where we test our system under a variety of differing environments to make sure it is robust. This is also the place where we will fix any issues that arise. Lastly, we will finalize our design and get ready to present. The schedule is shown in Fig. 4.

8.2 Team Member Responsibilities

While each team member is assigned to a different portion of this project, there are a lot of interconnected components and so various responsibilities will be shared. Each team member is also responsible for making sure that the others are on schedule and that no one is falling behind. The table below lists the primary and secondary responsibilities of each team member.

Team Member	Primary Tasks	Secondary Tasks
Keshav Sangam	LIDAR and Xbox Kinect setup and integration	SLAM and beacon localization for 'human' detection
Jai Madisetty	Designing path planning algorithm	iRobot movement control and SLAM
Raymond Xiao	Program iRobot to react to sensor data	Integration of all components to form cohesive system

8.3 Bill of Materials and Budget

The bill of materials is shown in Table 1. The main hardware components used in this project are the NVIDIA Jetson Xavier NX, the iRobot Create 2 system, the RPLIDAR AM1 and Xbox Kinect, and the NP-F750 Battery to power the entire system. For software components, we will be using libraries such as OpenCV and pycreate2.

8.4 Risk Mitigation Plans

This project has a lot of inherent risks associated with it. The entire system needs to be integrated seamlessly, otherwise, the whole system will not work. The main critical risk factors are no previous knowledge of ROS, interfacing with various sensors, and ensuring that the robot is completely autonomous. We will mitigate the first risk by ensuring that everyone of the team participates in basic ROS tutorials and has a working knowledge of it. The Xbox Kinect has various tutorials online [7] on how to interface with it using OpenCV that will be helpful. Beacon detection can be simplified by using AprilTags. AprilTag detection is already in the OpenCV library, which should provide us an easy alternative to a neon beacon. In addition, the RPLidar comes with a startup kit to allow for easy ROS interfacing. To mitigate the last risk, we will thoroughly test the system under a wide variety of environments to ensure that it can reliably operate in all types of situations. Furthermore, problems regarding development of the various robot control algorithms can be mitigated with online tutorials; the worst case could be using a prebuilt package to provide robot control.

Table 1: Bill of materials

Description	Manufacturer	Quantity	Unit Cost	Total
NVIDIA Jetson Xavier NX	NVIDIA	1	$1,797.92	$1,797.92
iRobot Create 2 System	iRobot	1	$200.00	$200.00
RPLIDAR AM1	Slamtec	1	$99.99	$99.99
Xbox Kinect	Microsoft	1	$40.00	$40.00
NP-F750 Battery	Powerextra	1	$40.00	$40.00

9 RELATED WORK

There are few projects that truly encapsulate our use-case. Some papers we looked at include "See Through Smoke: Robust Indoor Mapping with Low-cost mmWave Radar" [5]. This paper was determined the efficacy of using radar for SLAM, specifically for situations like fires. However, we are not using the neural network created in the paper since we have decided to move away from radar. Furthermore, as mentioned earlier, there is a commercially available firefighting robot known as Thermite [8]. However, this system is mainly for fire control (as in literally fighting the fire via a hose), and less for search and rescue.

10 SUMMARY

Our design is an autonomous robot which will help firefighters quickly scour a building for survivors and people in need of help. This robot is designed to efficiently navigate through a building floor in dangerous conditions, scouting for humans and providing valuable information back to the firefighters. This robot can save both time and lives as it will mark rooms that need to be searched whilst skipping empty rooms. Challenges that we anticipate in the implementation phase include integration of all components, ensuring that the robot is fully autonomous, and achieving 100% beacon localization accuracy.

Glossary of Acronyms

Include an alphabetized list of acronyms if you have lots of these included in your document. Otherwise define the acronyms inline.

- SAR - search and rescue
- LIDAR - light detection and ranging
- SLAM - simultaneous localization and mapping
- CV - computer vision
- USB - universal serial bus
- ROS - Robot Operating System
- UWB - Ultra-Wide Band
- BLE - Bluetooth Low Energy

References

[1] Rita Fahy. "Firefighter fatalities in the United States". In: *National Fire Protection Agency* (Oct. 2021).

[2] Alessandro Francescon. *Navigation stack test: GMapping vs Hector Slam*. 2015. URL: `http://www.geduino.org/site/archives/36`.

[3] *Hector SLAM*.

[4] *Hector SLAM*. 2019. URL: `http://wiki.ros.org/gmapping`.

[5] Chris Xiaoxuan Lu et al. "See Through Smoke: Robust Indoor Mapping with Low-cost mmWave Radar". In: (2020). arXiv: `1911.00398 [eess.SP]`.

[6] "Occupational Health and Safety". In: *Canadian Centre for Occupational Health and Safety* (Feb. 2022).

[7] Lee Stott. "Image Based Motion Analysis with Kinect V2 and OpenCV". In: *Microsoft Education* (Mar. 2019).

[8] "Thermite". In: *Howe Howe Technologies* (Feb. 2022).

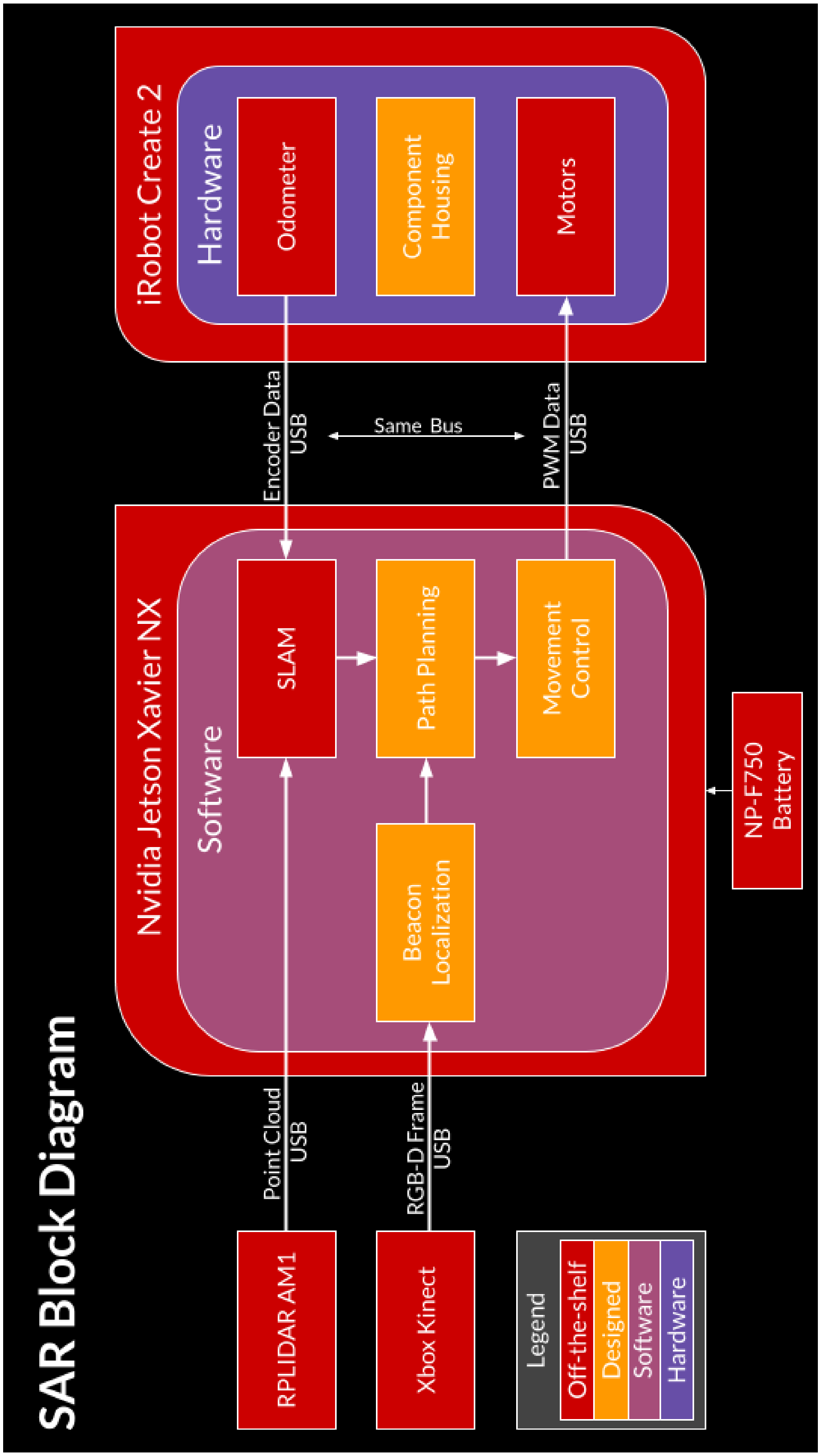

Figure 3: A full-page version of the same system block diagram as depicted earlier.

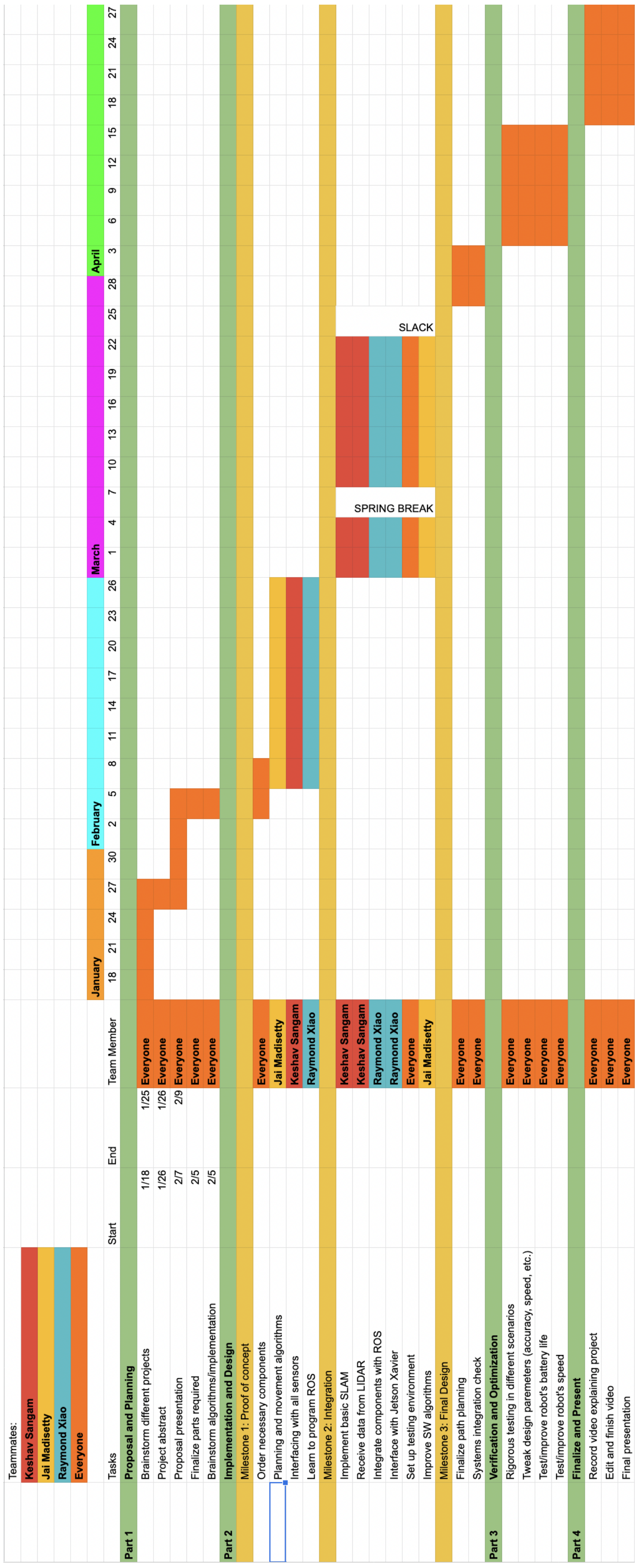

Figure 4: Gantt Chart