2D Autonomous Mapping

Amukta Nayak Tiffany Chiang Kanupriyaa
18-500 ECE Design Experience, Team B4
Electrical and Computer Engineering, Carnegie Mellon University
{apnayak, tchiang, knosurna}@*andrew.cmu.edu*

***Abstract*—The goal of this project is to create a small ground vehicle that autonomously and simultaneously localizes and maps the 2D maze it is placed in. The maze will have smooth, straight walls and a smooth ground. This is an educational demonstration of an autonomous vehicle that may be used for search and rescue or scientific exploration.**

Keywords—Mapping, 2D, Autonomous, Visualization, Localization, SLAM, Maze,

I. Introduction

Autonomous vehicles are often used to explore difficult to reach or dangerous places for situations like search and rescue, and scientific exploration in general. This project aims to create a low cost, educational demonstration of autonomous mapping and navigation by building a vehicle that explored a simulated environment. Mechanical design challenges will be largely eliminated by reducing a real, textured environment to a static environment with smooth ground and flat walls. The ground vehicle will be capable of producing a 2D map of the maze it is placed in by traversing through it and updating live changes. It should use simultaneous localization and mapping (SLAM) to do this efficiently, and a user should be able to start and stop the vehicle through a web app.

II. Requirements

The ground vehicle shall:

- be capable of exploring an area with smooth surfaces.
- traverse the area above a minimum decided speed
- have a battery life of at least two hours
- track map coverage
- be able to simultaneously localize itself and map the area
- travel efficiently without hitting any walls

The user must be able to:

- see the map being updated as the vehicle moves
- start and stop the vehicle through a web appplication

III. Architecture

A. Summary

This system is divided into five subsystems: sensor, SLAM, path planning, motion, and map visualization. The sensor subsystem consists of the lidar, encoder, any other sensors we may add later to boost performance, as well as I/O and preprocessing software. This subsystem will consist entirely of off the shelf components. For the lidar (RPLidar) we will use the included SDK and ROS package to read data points. For the encoder we are writing a custom ROS node to parse the encoder digital signal. The SLAM subsystem will receive sensor data, and compute the estimated occupancy grid map and robot position on the map. This subsystem will handle ICP frame transformations, pose estimation, mapping, and graph optimization. The path planning subsystem will read the resulting map and position from the SLAM subsystem, and direct the robot on a path to build a complete map of the maze autonomously. This subsystem includes creating a waypoint based graph representation of the maze, building a search path to explore the maze, and computing the motor velocity commands necessary to traverse the maze. The SLAM and path planning subsystems will both be implemented entirely by our team, with the exception of message transmitting/receiving libraries for the ROS framework. The motion subsystem consists of the motor driver software, motor controller and motor. The motor velocity commands are communicated from the path planning subsystem to the motor driver, which is responsible for creating the PWM signal needed to run the motor at the desired velocity. This subsystem uses off the shelf motors, motor controller, and the included motor driver library with a custom wrapper. The robot's motion will change the robot's perception of the maze, thus changing the sensor inputs and restarting this cycle. Lastly, the map visualization subsystem is a web application that also receives and displays the most recent occupancy grid map and robot position. The overall web application will be custom designed for this application, however we will utilize the React.js library to create the map visualization.

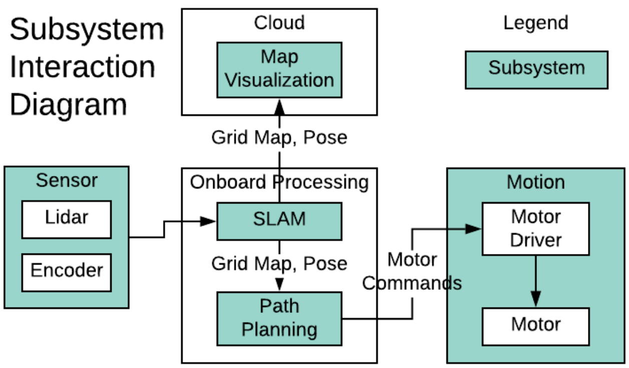


Fig. 1. Subsystem Interaction Diagram

B. SLAM Subsystem

Many robotics problems have already solved the problem of localization when the environment around the robot or ground vehicle is known. Similarly, the problem of mapping when the position of robot is known in the environment has also been solved. The issue we are trying to address in this project in Simultaneous Localization and Mapping in the robotics space where neither the position of the robot nor the map of the environment is previously known.

In our approach we have decided to use an ICP based pose-graph SLAM algorithm which is described below. In our approach we solve the problem in two stages. The first stage includes building the pose graph using the ICP algorithm to find subsequent poses (Graph Construction). The second stage consists of using pose graph optimization to find the best pose data that best satisfies the constraints between the nodes in the graph (Graph Optimization). Finally, we discuss the ICP algorithm used to determine the poses used in the graph.

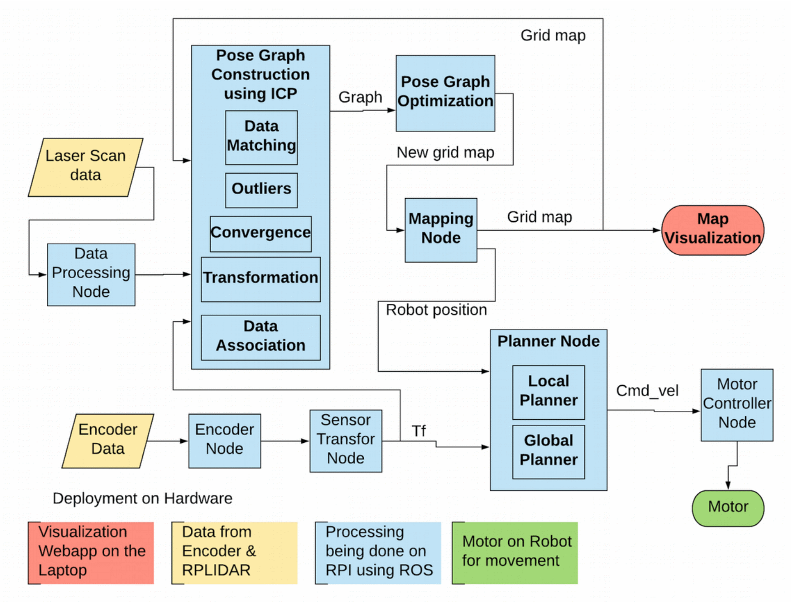


Fig. 2. Overall Software Architecture Diagram

The first stage consists of building a graph where the poses of the robot are modeled by the nodes in the graph. Each pose in the graph contains the scan taken at that point as well as the x,y and z component from the starting point which is considered ground zero. The edge between two nodes represents the spatial constraint relating the two robot poses. This means that it is modeled by a posterior distribution over the relative transformation between the two poses. These transformations are either odometry measurements between sequential robot positions or are determined by aligning the observations acquired at the two robot locations (ICP).

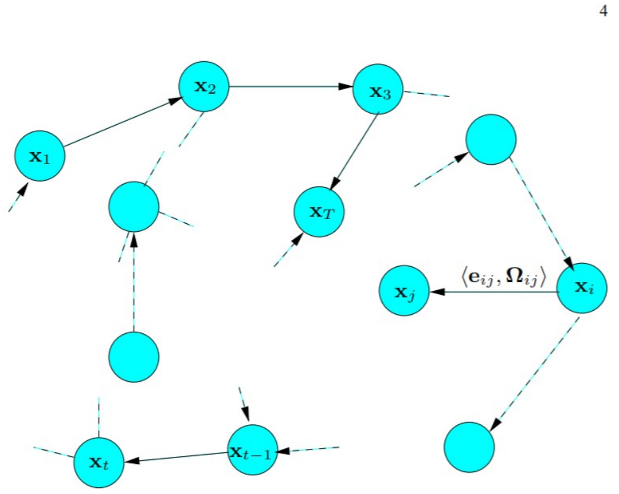


Fig. 3. SLAM Graph

To explain this is detail we will go through the steps required to construct two nodes and an edge in the graph. Let us imagine the first node we are looking at is X1. To get the second node X2 we use the two scans taken at both of these nodes and put them through an ICP algorithm (described below). This algorithm returns what it believes is the position (x2,y2,z2) of pose X2 by matching both of these scans. Now we can model the edge between the two poses as a posterior distribution and add the second pose to the map.

$$M_{t+1} = M_t\, U\, \{ < Z_{t+1} , X_{t+1} > \}$$

Since we are using the ICP algorithm we can find the pose of loop closures using the same ICP algorithm we use to build the entire map.

The ICP algorithm works by comparing two scans taken a small distance apart where a significant amount of the scan are of the same area and hence overlap.

Data Matching is being done using point-to-plane algorithm. In this algorithm a Matcher module links the points in the reference scan to the points in the reading scan. When this metric is used, we minimize the sum off the squared distance between each reading point and the tangent plane at its corresponding reference point. The reading point projects along the normal and the intersection at the reference scan is taken as the corresponding point match. This structure is slightly better for smooth structures and worse for noisy structures.

Outliers remove some of the links between points in the reading and their matched points in the reference. This is done so irregularities can be decreased while we are running the convergence iteration later. We are removing all matches that are further away than the mean distance between points plus 0.01 inches.

Data Association is for storing the matching points in a data structure. We will be using a kd-tree to store these values for optimization and faster runtime.

Convergence Test is used for iteration over all point pair matches. If the function converges under certain conditions such as 150 maximum iterations and error falls below 1 cm and 0.01 radian the test has succeeded, and we are returned the transformation. We will be using the point-to-plane error function.

Transformation module takes the transformation given by the ICP algorithm and applies it to the previous pose to give the current pose that we were searching.

Once the graph is constructed we seek to find the configuration of the robot poses that best satisfies the constraints. Thus, in graph-based SLAM the problem is decoupled in two tasks: constructing the graph from the raw measurements (graph construction), determining the most likely configuration of the poses given the edges of the graph (graph optimization).

Now to get the posterior distribution, we find the probability of getting X2 given the odometry data, the previous pose, the map up till now and the scan at pose 2.

$$P(X_2 | X_1 , U, M, Z) = P(Z \mid X_2 , M)\, P(X_2 \mid X_1, U)$$

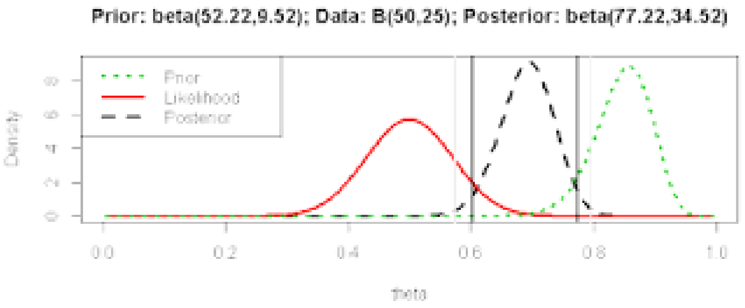


Fig. 4. SLAM Graph

Thus the probability is a function of the motion model P(X2 | X1, U) and the perceptual (observation) model P(Z | X2 , M). Throughout this paper, we will assume the probabilistic motion model to be as shown in the figure below. As can be seen, the shape of this conditional density resembles that of a banana. This distribution is obtained by the kinematic equations assuming that the robot is noisy along its rotational and translational axis.

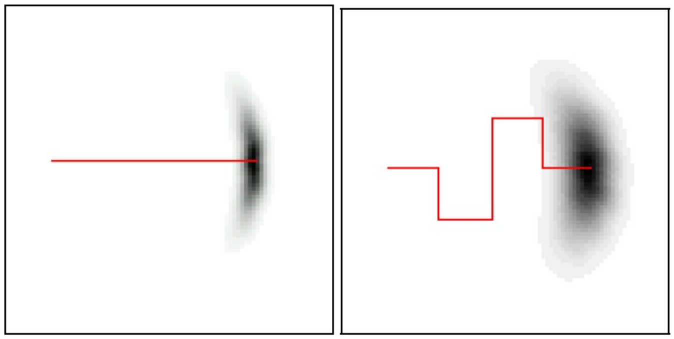
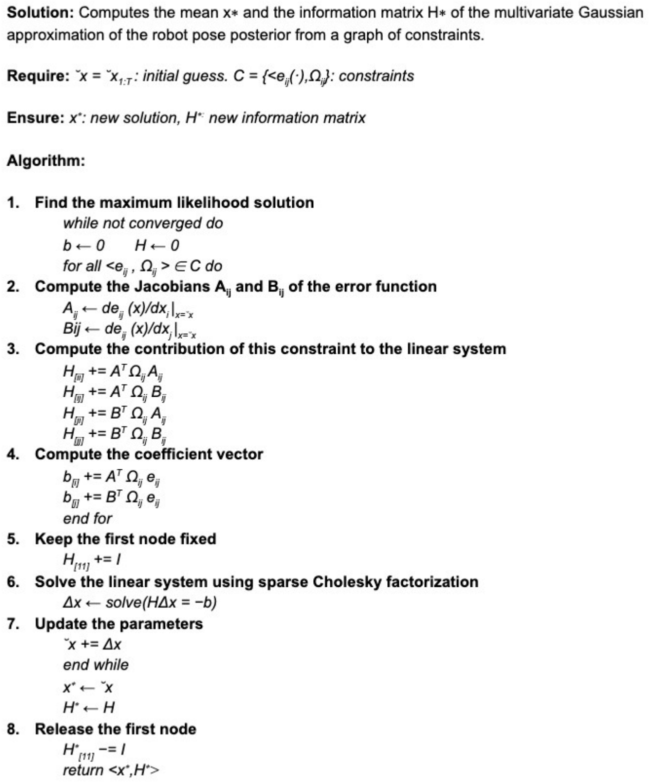


Fig. 5. SLAM Graph

Now we have to compute the Gaussian approximation of the posterior over the robot trajectory. This would include computing the mean of the Gaussian as configuration of nodes that maximizes the likelihood of the observations. We can find this by characterizing the problem as a constraint optimization problem. The algorithm for the optimization is taken from another paper.



Solution: Computes the mean x∗ and the information matrix H∗ of the multivariate Gaussian approximation of the robot pose posterior from a graph of constraints.

Require: $\breve{x} = \breve{x}_{1:T}$: *initial guess.* $C = \{<e_{ij}(\cdot),\Omega_{ij}>\}$: *constraints*

Ensure: x^*: *new solution,* H^* *new information matrix*

Algorithm:

```
1. Find the maximum likelihood solution
       while not converged do
       $b \leftarrow 0$      $H \leftarrow 0$
       for all $<e_{ij}, \Omega_{ij}> \in C$ do
2. Compute the Jacobians $A_{ij}$ and $B_{ij}$ of the error function
       $A_{ij} \leftarrow de_{ij}(x)/dx_i|_{x=\breve{x}}$
       $Bij \leftarrow de_{ij}(x)/dx_j|_{x=\breve{x}}$
3. Compute the contribution of this constraint to the linear system
       $H_{[ii]} += A^T \Omega_{ij} A_{ij}$
       $H_{[ij]} += A^T \Omega_{ij} B_{ij}$
       $H_{[ji]} += B^T \Omega_{ij} A_{ij}$
       $H_{[jj]} += B^T \Omega_{ij} B_{ij}$
4. Compute the coefficient vector
       $b_{[i]} += A^T \Omega_{ij} e_{ij}$
       $b_{[j]} += B^T \Omega_{ij} e_{ij}$
       end for
5. Keep the first node fixed
       $H_{[11]} += I$
6. Solve the linear system using sparse Cholesky factorization
       $\Delta x \leftarrow solve(H\Delta x = -b)$
7. Update the parameters
       $\breve{x} += \Delta x$
       end while
       $x^* \leftarrow \breve{x}$
       $H^* \leftarrow H$
8. Release the first node
       $H^*_{[11]} -= I$
       return $<x^*,H^*>$
```

Fig. 6. SLAM Algorithnm

C. Path Planning Subsystem

After the SLAM subsystem calculates an estimated occupancy grid map and robot pose, the path planning subsystem is responsible for using this information to instruct the robot how to traverse through the maze without hitting obstacles, and how to autonomously build a complete map. The path planning subsystem consists of three parts: the local planner, global planner, and graph builder.

Given the robot's current pose, and the global goal pose, the local planner will compute the motor velocity trajectory needed for the robot to travel to the goal pose. First, a value function of the surrounding space is generated. This is represented as a grid map containing the Manhattan distances from each pixel to the global path leading to the goal position. If the global goal position is not in the surrounding space, the closest pixel to the goal will be treated as the local goal. Then, potential trajectories (motor velocities) are simulated, and a weighted score is computed using factors such as whether or not the trajectory hits an obstacle, duration, distance of the resulting position to the local goal, and standard deviation of normal distance from the simulated path to the desired global path. The trajectory with the lowest score (fastest, closes to the desired global path, doesn't hit any obstacles) will be sent to the motion subsystem as motor velocity commands. We will be taking the base_local_planner ROS package and customizing the cost function to exclude pixels the robot cannot traverse to, and to use the dynamic window approach (DWA) instead of the default trajectory rollout algorithm.

The trajectory rollout algorithm samples trajectories from a much larger sample space the the the DWA algorithm, because it simulates velocity samples for each simulation step from the current time until the robot reaches the goal position, whereas the DWA algorithm tests outcomes of the velocity samples for the next immediate simulation step. The trajectory rollout algorithm may produce trajectories that are closer to the desired global path, but DWA is a much more efficient algorithm (fewer steps to simulate). Since our application has a simplified environment, with all waypoints on the global path only 1 maze segment and in direct view of the previous waypoint, the DWA will be sufficient.

The global planner is responsible for searching for unexplored frontiers through the graph representation of the maze generated by the graph builder. This will be implemented with the depth first search algorithm. Due to the design of the maze, all linked nodes are guaranteed to have a straight line path connecting the two in the physical maze, so the global path will simply be set to the straight line path between each node.

The graph builder will receive the most updated grid map, and build a graph representation of the maze. First, it will generate waypoints at intervals of 1ft (or the dimension of each maze segment) to create a 36 node graph for a 6x6 maze, or a 16 node graph for a 4x4 maze. The grid map coordinates of each waypoint will be stored as well. Then, for each node it will search in the direction its 8 neighboring pixels in the grid, and continue scanning away from the starting waypoint until it either reaches a wall, a neighboring node, or unexplored pixels of the grid map. If an explored neighboring node is found on the path, the two known nodes will be linked and the link will be marked as explored. If an unexplored pixel is found, the unexplored node closest to the unexplored pixel, will be linked with the starting node, and the link will be placed on a queue to be explored. This unknown link can be referred to as a frontier for autonomous mapping. After all the paths have been explored, we will have a graph representation of the known portion of the map.

D. Map Visualization Subsystem

The map representation of the maze will be an occupancy grid map, which is a 2D array where a filled in cell represents a wall and a blank cell represents a traversable hallway. The robot will asynchronously broadcast newly discovered wall coordinates to a Node.js web server through POST requests. Every 50 packets, the entire map's data will be broadcasted, to make up for dropped/lost packets or drift. The web application's UI will be built on the React.js library, with the React D3 graphing library to graph the map representation. Upon the server receiving new coordinate information, the graph will be updated and served to clients. There will be a button on the web app to start (when the robot isn't mapping and traversing) or stop (when it is currently mapping and traversing). This will send a request to the robot's Raspberry Pi, so that the user can start/stop the robot easily.

E. Motion Subsystem

We are using a 12V, 350 rpm DC motor kit, which has an encoder, wheel and motor mount. The max speed of the wheel is 229 ft / min, which is more than enough for our requirement. We will be using the Adafruit Raspberry Pi Motor Hat, which can power up to 12V motors and 1 A of current, which is enough for the stall current for the motor. This motor controller also includes a python library that we will use to control the motor duty cycle.

F. Sensor Subsystem

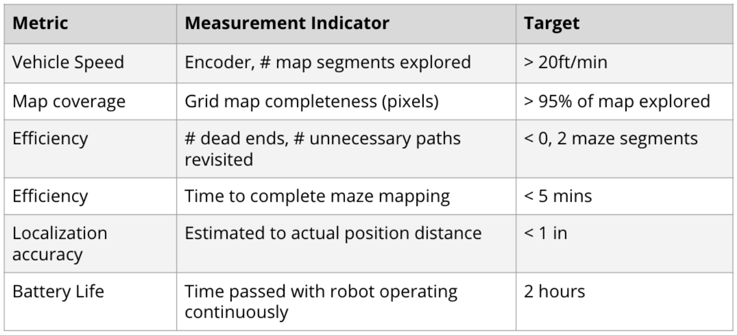
We will be using the A1M8 RPLidar due to its affordable cost, and convenient SDK. This module has also been used by previous capstone teams. It reads 360 points per rotation at 1 degree resolution, and rotates 5.5 times per second, resulting in 1980 points per second.

IV. Testing

A. Performance Metrics

As mentioned in the design requirements, we will be using the following performance metrics:

.



Metric	Measurement Indicator	Target
Vehicle Speed	Encoder, # map segments explored	> 20ft/min
Map coverage	Grid map completeness (pixels)	> 95% of map explored
Efficiency	# dead ends, # unnecessary paths revisited	< 0, 2 maze segments
Efficiency	Time to complete maze mapping	< 5 mins
Localization accuracy	Estimated to actual position distance	< 1 in
Battery Life	Time passed with robot operating continuously	2 hours

Fig. 7. Performance Mentrics

The map coverage and localization accuracy metrics are based on the resolution of the RPLidar. The vehicle speed, efficiency (time to complete maze), and battery metrics are based on the following calculation. Assuming our maze is 6ft x 6ft, with hallways around 1x1ft wide, we can have a max of 36 (x2 for horizontal and vertical direction hallways) ft or 24 (x2 for horizontal and vertical direction hallways) ft of hallways. Depending on how hallways are connected, the robot may need to travel through the same hallway a few times (let's say 2 times for the whole maze) to explore the whole map, in which the worse case scenario would be 144 (72x2) ft. Therefore 20 ft / min will allow us to cover 100 ft in 5 mins, which should comfortably allow the robot some mistakes.

B. Validation Plan

We will be building a modular maze to test the robot against the aforementioned performance metrics. The maze will be modular, easily configurable, and multiple types of junctions in order to ensure that the robot dynamically localizes itself and maps the area it is in. The maze will have 1 ft wide hallways and the following modular combinable 1x1x1 ft section types.

Validation will have two phases. In Phase 1, we will build a small maze with 4-6 sections to test basic navigation without autonomous mapping. In Phase 2, we will expand the maze to a 6x6 grid, which will be the size of the final demo grid. We will be testing the map coverage on various configurations, and concurrently test the map algorithm with software simulations.

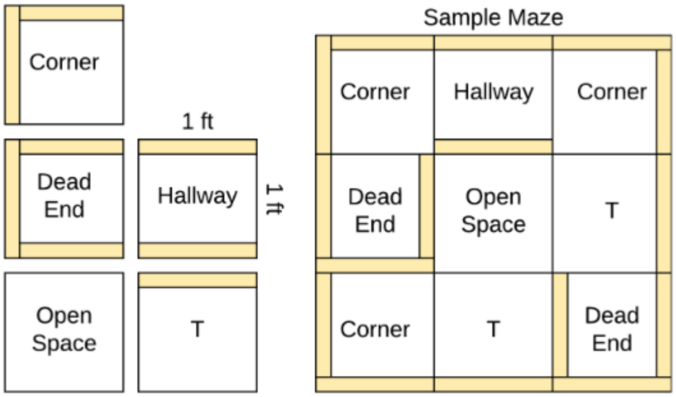


Fig. 8. Maze components and a sample 3x3 maze

The validation timeline will be discussed further in the Project Management section.

V. Project Management

A. Schedule and Risk Management

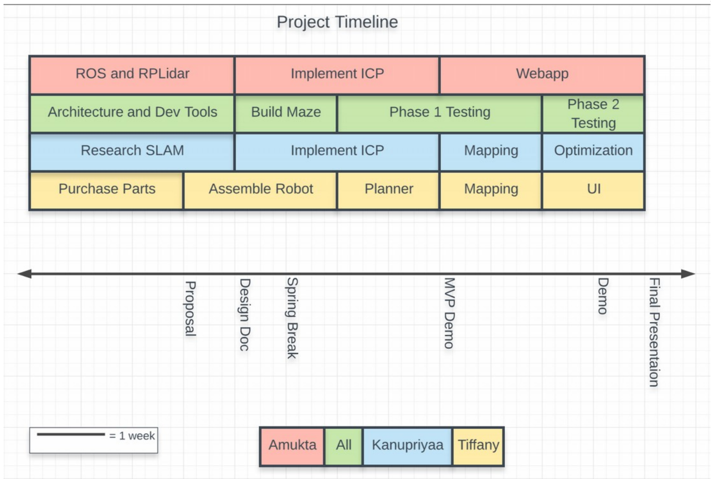
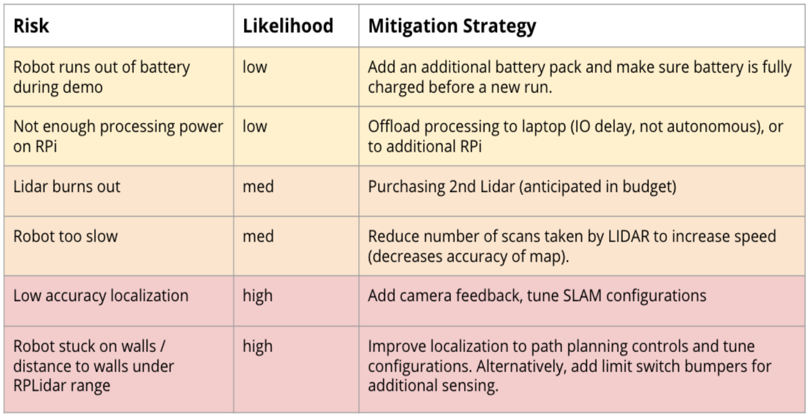


Fig. 9. Simplified Gantt Chart



Risk	Likelihood	Mitigation Strategy
Robot runs out of battery during demo	low	Add an additional battery pack and make sure battery is fully charged before a new run.
Not enough processing power on RPi	low	Offload processing to laptop (IO delay, not autonomous), or to additional RPi
Lidar burns out	med	Purchasing 2nd Lidar (anticipated in budget)
Robot too slow	med	Reduce number of scans taken by LIDAR to increase speed (decreases accuracy of map).
Low accuracy localization	high	Add camera feedback, tune SLAM configurations
Robot stuck on walls / distance to walls under RPLidar range	high	Improve localization to path planning controls and tune configurations. Alternatively, add limit switch bumpers for additional sensing.

Fig. 10. Risk Management Chart

B. Member Responsibilities

Amukta will be focusing on webapp functionality/ visualization, Raspberry Pi/ ROS/ VM setup, lidar data input/ processing, ICP setup and implementation of the first half of ICP.

Kanupriyaa will focus on SLAM calculations/ overall algorithm, ICP setup, implementation of the second half of ICP, mapping, graph optimization and sending the map representation to web UI.

Tiffany will be researching and ordering parts, finalizing the ROS navigation stack, drawing electrical schematics, hardware/chassis design, encoder and motor controllers, and planning (both local and global).

Everyone will be working on software architecture design, writing reports, building the maze, and doing Phase 1 and 2 testing.

C. Budget

The three most expensive components of the robot are the RPLidar, the Raspberry Pi, and the two motors. We assumed that low-cost scavengable parts like wires and acrylic pieces for the chassis were nearly free and therefore did need to be added to the budget. Assuming the RPLidar is sufficient and none of the foreseeable risks occur (see Risk Management), the total cost of the robot is about $250 dollars. If all the foreseeable risks occur, we would need to buy an additional RPLidar and an ultrasonic sensor, resulting in a total cost of about $370 dollars. Either way, we are well under the $600 limit. Refer to the Bill of Materials (Appendix 1).

VI. Summary

Autonomous vehicles have a variety of important applications, including search and rescue, and scientific exploration. We aim to create an educational demo of a ground vehicle that performs simultaneous localization and mapping in a simplified environment. We intend to create a demo that is configurable and user-friendly, through the use of a modular 2D maze and a web application for controlling the vehicle and visualization.

A. Future Work

This project's scope has been limited to eliminate mechanical and environmental challenges. There are many ways to extend the functionality to real-world scenarios like: low light/ dark areas, bumpy/rocky/grassy grounds, curved/ bumpy walls of various heights, sudden cliffs/ steep inclines.

Modifying the robot to be compatible with these situations would make it more useful for search and rescue, and scientific exploration applications.

REFERENCES

[1] Ellon Mendes, Pierrick Koch, Simon Lacroix. ICP-based pose-graph SLAM. International Symposium on Safety, Security and Rescue Robotics (SSRR), Oct 2016, Lausanne, Switzerland. International Symposium on Safety, Security and Rescue Robotics, pp.195 - 200, 2016, <10.1109/SSRR.2016.7784298>. <hal-01522248>

[2] Borrmann, Dorit, et al. “Globally Consistent 3D Mapping with Scan Matching.” Robotics and Autonomous Systems, vol. 56, no. 2, 2008, pp. 130–142., doi:10.1016/j.robot.2007.07.002.

[3] Censi, Andrea. “An Accurate Closed-Form Estimate of ICPs Covariance.” Proceedings 2007 IEEE International Conference on Robotics and Automation, 2007, doi:10.1109/robot.2007.363961.

[4] Grisetti, G, et al. “A Tutorial on Graph-Based SLAM.” IEEE Intelligent Transportation Systems Magazine, vol. 2, no. 4, 2010, pp. 31–43., doi:10.1109/mits.2010.939925.

[5] Kohlbrecher, Stefan, et al. “Hector Open Source Modules for Autonomous Mapping and Navigation with Rescue Robots.” Applied Cryptography and Network Security Lecture Notes in Computer Science, 2014, pp. 624–631., doi:10.1007/978-3-662-44468-9_58.

[6] Lu, Feng, and Milios. “Robot Pose Estimation in Unknown Environments by Matching 2D Range Scans.” Proceedings of IEEE Conference on Computer Vision and Pattern Recognition CVPR-94, 1994, doi:10.1109/cvpr.1994.323928.

[7] López, Elena, et al. “A Multi-Sensorial Simultaneous Localization and Mapping (SLAM) System for Low-Cost Micro Aerial Vehicles in GPS-Denied Environments.” Sensors, vol. 17, no. 4, 2017, p. 802., doi:10.3390/s17040802.

[8] Olson, E.b. “Real-Time Correlative Scan Matching.” 2009 IEEE International Conference on Robotics and Automation, 2009, doi:10.1109/robot.2009.5152375.

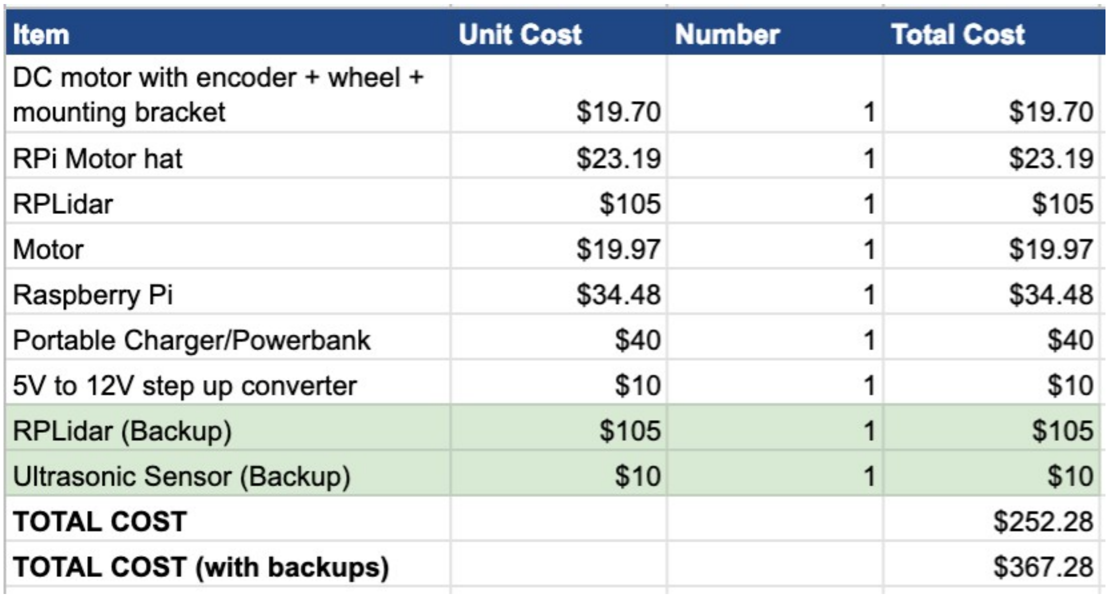
[9] Pomerleau, François, et al. “Relative Motion Threshold for Rejection in ICP Registration.” Springer Tracts in Advanced Robotics Field and Service Robotics, 2010, pp. 229–238., doi:10.1007/978-3-642-13408-1_21.

[10] Pomerleau, François, et al. “Comparing ICP Variants on Real-World Data Sets.” Autonomous Robots, vol. 34, no. 3, 2013, pp. 133–148., doi:10.1007/s10514-013-9327-2.

[11] Thrun, S., et al. “A Real-Time Algorithm for Mobile Robot Mapping with Applications to Multi-Robot and 3D Mapping.” Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065), doi:10.1109/robot.2000.844077.

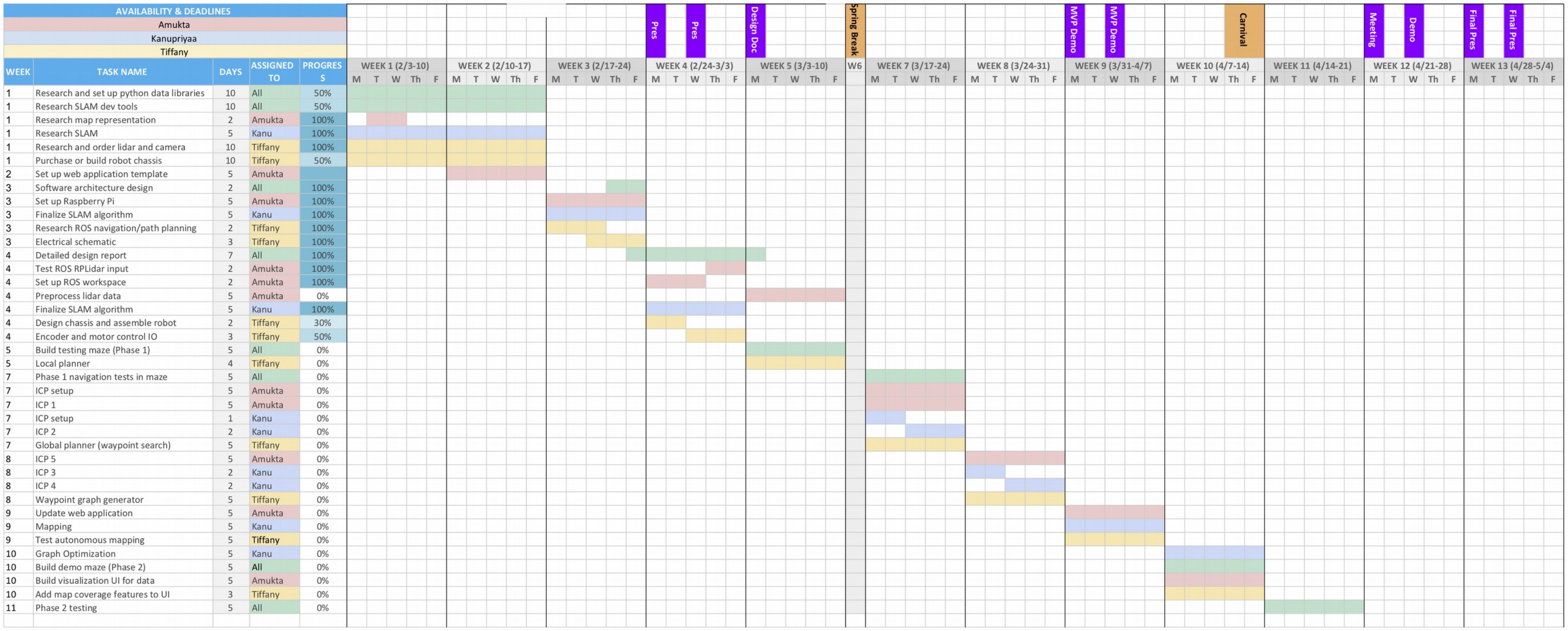
[12] Choset, Howie., et al. “Robotic Motion Planning: Sample-Based Motion Planning” Robotic Motion Planning, Ch. 7

Appendix A: Bill of Materials



Item	Unit Cost	Number	Total Cost
DC motor with encoder + wheel + mounting bracket	$19.70	1	$19.70
RPi Motor hat	$23.19	1	$23.19
RPLidar	$105	1	$105
Motor	$19.97	1	$19.97
Raspberry Pi	$34.48	1	$34.48
Portable Charger/Powerbank	$40	1	$40
5V to 12V step up converter	$10	1	$10
RPLidar (Backup)	$105	1	$105
Ultrasonic Sensor (Backup)	$10	1	$10
TOTAL COST			$252.28
TOTAL COST (with backups)			$367.28

Appendix B: Gantt Chart



WEEK	TASK NAME	DAYS	ASSIGNED TO	PROGRESS
1	Research and set up python data libraries	10	All	50%
1	Research SLAM dev tools	10	All	50%
1	Research map representation	2	Amukta	100%
1	Research SLAM	5	Kanu	100%
1	Research and order lidar and camera	10	Tiffany	100%
1	Purchase or build robot chassis	10	Tiffany	50%
2	Set up web application template	5	Amukta	
3	Software architecture design	2	All	100%
3	Set up Raspberry Pi	5	Amukta	100%
3	Finalize SLAM algorithm	5	Kanu	100%
3	Research ROS navigation/path planning	2	Tiffany	100%
3	Electrical schematic	3	Tiffany	100%
4	Detailed design report	7	All	100%
4	Test ROS RPLidar input	2	Amukta	100%
4	Set up ROS workspace	2	Amukta	100%
4	Preprocess lidar data	5	Amukta	0%
4	Finalize SLAM algorithm	5	Kanu	100%
4	Design chassis and assemble robot	2	Tiffany	30%
4	Encoder and motor control IO	3	Tiffany	50%
5	Build testing maze (Phase 1)	5	All	0%
5	Local planner	4	Tiffany	0%
7	Phase 1 navigation tests in maze	5	All	0%
7	ICP setup	5	Amukta	0%
7	ICP 1	5	Amukta	0%
7	ICP setup	1	Kanu	0%
7	ICP 2	2	Kanu	0%
7	Global planner (waypoint search)	5	Tiffany	0%
8	ICP 5	5	Amukta	0%
8	ICP 3	2	Kanu	0%
8	ICP 4	2	Kanu	0%
8	Waypoint graph generator	5	Tiffany	0%
9	Update web application	5	Amukta	0%
9	Mapping	5	Kanu	0%
9	Test autonomous mapping	5	Tiffany	0%
10	Graph Optimization	5	Kanu	0%
10	Build demo maze (Phase 2)	5	All	0%
10	Build visualization UI for data	5	Amukta	0%
10	Add map coverage features to UI	3	Tiffany	0%
11	Phase 2 testing	5	All	0%