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.

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

III. Architecture

A. Summary

Our architecture consists of both hardware and software systems, as described in detail in the following section.

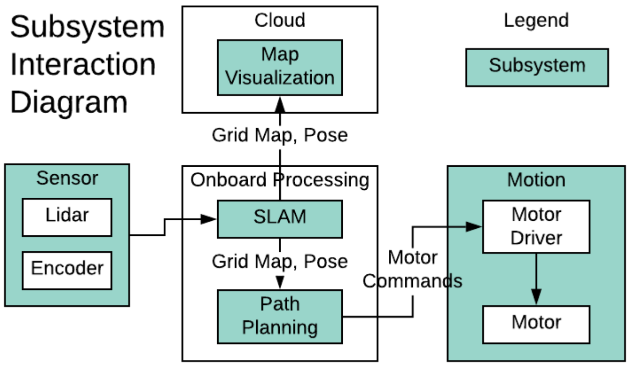


Fig. 1. Subsystem Interaction Diagram

B. Electrical and Mechanical System

The mechanical structure of the robot consists of mounts made for various components laser cut tempered wood, stacked vertically using standoffs. The base is a two wheeled differential drive balanced with two ball caster wheels, which has a simpler motor control interface and mechanical design, and high maneuverability. Heavier components such as the motors and battery pack are compacted on bottom layers to increase stability, while the Raspberry Pi and motor controller are allocated more space for heat dissipation. We decided to use a Raspberry Pi 3 on our robot running an ubuntu 16 OS with ROS. ROS allows easier integration between separate subsystems, and the wireless capabilities of the Pi 3 allow us to port code and run tests remotely.

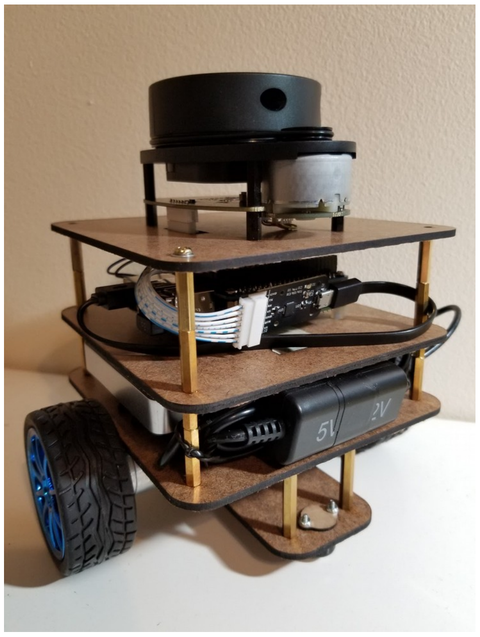


Fig. 2. Robot

C. Software Subsystems

This system is divided into six subsystems: Sensors, SLAM, Odometry, Path Planning, Motion, and Map Visualization. Each subsystem (with exception of the web application) is contained in a ROS package and communicate via ROS topics.

1. Sensor - This subsystem consists of the lidar, encoder, as well as I/O and preprocessing software. For the lidar (RPLidar) we used the manufacturer included SDK and ROS package to read data points. The RPLidar publishes an array of data points which correspond to the distance at which an object was detected at that degree. This data is subscribed to by the SLAM module which uses the scan data to calculate pose transformation for SLAM. For the encoder, a custom ROS node parses the digital signal from Phase A and B hall sensors in the encoder into vehicle position in terms of "ticks", a measurement of the wheel position relative to its starting position.

2. SLAM Software - The SLAM subsystem is the software that computes the occupancy grid map of the surrounding environment by taking in scans from the encoders and outputting the pose estimation from one scan to another.

3. Odometry - The odometry subsystem receives encoder ticks and calculates the robot's transformation frame from its starting pose to current pose.

4. Path Planning - The path planning subsystem is separate into the local path planner and global path planner. The local path planner produces motor velocity commands to move the robot from current point A to destination B in the maze, where A and B can be connected in a straight path without obstacles in between. The global planner analyzes the grid map generated by the SLAM subsystem and provides the local planner with destinations to go to. The path planning subsystem subscribes to the robot pose transformation matrix and the grid map produced by the SLAM subsystem. Within the path planning subsystem, the global planner uses this information to calculate map coordinates of the next destination, and the local planner uses the coordinates of the current pose and destination to calculate motor velocities. In the final version of our system, due to delays with implementing SLAM, we were unable to complete integration and testing for the global path planner. Therefore the global path planner is replaced by a human controlling the robot. The UI can be used remotely via ssh with X forwarding. The UI also outputs variable motor speeds, which still demonstrates the PID velocity function.

5. Motion - This subsystem consists of PID velocity node, and motor controller. The path planning subsystem sends velocity commands PID velocity node, which uses encoder data to calculate the motor commands needed to achieve the target velocities. This software subsystem interacts with the Adafruit Raspberry Pi motor hat, which outputs the correct PWM signal to the motor. The robot's motion will change the robot's perception of the maze, thus changing the sensor inputs and creating a sensor feedback loop.

6. User Web App - This subsystem is a web application

that receives and displays the most recent occupancy grid map and robot position. After every round of ICP, the grid map will change slightly. This updated grid map is sent to the web server via a POST request. The server sends all clients the map it stores every 5 seconds. The client visualizes the data it receives. The overall web application will be custom designed for this application, however we will utilize the React.js library to create the map visualization. Our software components will all be deployed on the Pi to achieve autonomous movement with the exception of the Web App which the user can use on any computer.

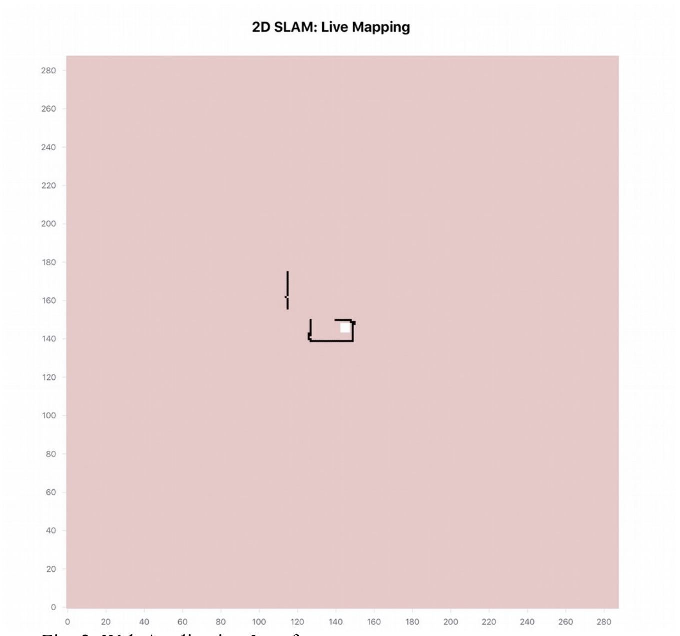


Fig. 3. Web Application Interface

IV. Design Trade Studies

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. 4. Performance Metrics

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. Theoretical Design

We used the resolution of the RPlidar to make the grid accuracy of the map. Since the resolution can be adjusted we used 288 by 288 so extra lag doesn't slow down the Pi.

Our total traversal time depended on how large we wanted the maze to be. Using an average of 20 ft/min we could traverse all mazes relatively fast.

C. Validation Plan

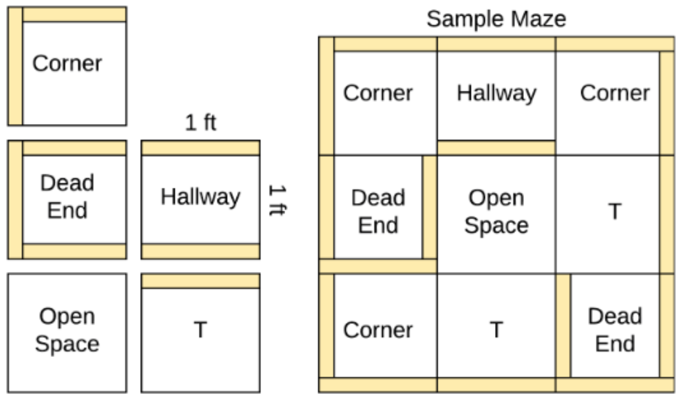


Fig. 5. Maze components and a sample 3x3 maze

We built a modular maze to test the robot against the aforementioned performance metrics. The maze was modular, easily configurable, and had multiple types of junctions in order to ensure that the robot dynamically localized itself and mapped the area it was in. The maze had 1 ft wide hallways and the following modular combinable 1x1x1 ft section types.

Validation will had two phases. In Phase 1, we built a small maze with 4-6 sections to test basic navigation without autonomous mapping. In Phase 2, we expanded the maze to a 6x6 grid, which was the size of the final demo grid. We tested the map coverage on various configurations, and concurrently tested the map algorithm with software simulations. This is a diagram of the configurable pieces and a sample 3x3 maze.

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

V. System Description

A. 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.

We used the hall sensor quadrature encoders included in our DC motor kit. The encoder position can be determined by catching the rising and falling edges of the A and B phases of the hall sensors, and counting the number of ticks the encoder has turned. A significant challenge in this module was that the hall sensor signals must be collected with no interruptions to avoid missing readings. However, when we added a ROS wrapper around the encoder reading python script, the ROS publisher was originally executed serially with the encoder reader, resulting in incorrect encoder readings. This in turn breaks the PID velocity controller, since the encoder-detected velocity is now capped at the ROS publishing rate, which is much lower than the target velocity. Therefore the PID velocity controller will continue to increase the motor speed in an attempt to reach the target velocity. The solution to this problem was to reimplement the encoder and encoder ROS wrapper in C++ with multithreading, so that encoder ticks are not missed while the ROS publisher sleeps in between publishing new encoder data.

B. Odometry Subsystem

Encoder ticks are converted into metric distance or radians by measuring the number of ticks per revolution. The encoder we used has 682.4 ticks per rotation according to the datasheet (closer to 700 when tested) and the wheel has a 0.065m diameter, which translates to approximately 3342 ticks per meter. This value can then be used to convert the ticks detected into distance (in meters) traveled by the left and right wheels. Wheel velocity is calculated by measuring travelled distance over small constant time units (delta time).

$$ticks\ per\ meter\ =\ \frac{ticks\ per\ rotation}{wheel\ circumference}$$

Fig. 6. Ticks per meter calculation

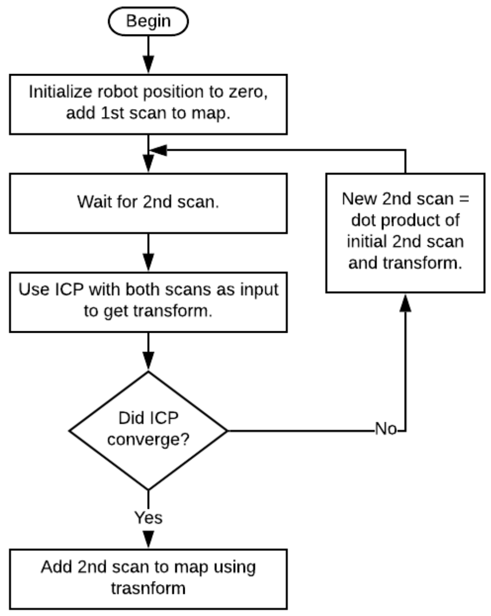

Begin
Initialize robot position to zero, add 1st scan to map.
Wait for 2nd scan.
Use ICP with both scans as input to get transform.
Did ICP converge?
No
New 2nd scan = dot product of initial 2nd scan and transform.
Yes
Add 2nd scan to map using trasnform

Fig. 7. SLAM Subsystem Diagram

C. SLAM Subsystem

For this subsystem we have used the Iterative Closest Point algorithm. This algorithm takes in two scans and determine how much the distance the robot has travelled before taking the second scan. This allows us to determine incremental pose transformations each time the robot takes an additional scan and place the scan points on the map accordingly. We also use the incremental pose transforms to determine where the robot is with respect to its starting point which we take to be (0,0,0).

ICP (Iterative Closest Point) works by taking first matching the points in the scans to each other and then determining what the transform between the two scans would be with this matching. If the mean error from the matching is below a certain threshold we take this as the final transform. Otherwise the ICP runs again using the dot product of the Scan A and the calculated transform until mean error is below the threshold or we cross the maximum number of iterations allowed.

1. Data Matching- This is the process by which each point in Scan A is mapped to a point in Scan B. Hence every scan points need to be matched with that scan point that is closest to it if the two points we projected in space. I have used the Python SkLearn library to do this computation.

2. Outliers- Determining outliers, or points that do not occur in both scans helps make the ICP better by giving better accuracy for the transform. For our system we have not used any outliers since we only have 360 points so each reduction would cost us in terms of accuracy.

3. Convergence- Convergence is achieved when the difference between the previous mean error between the scan points and the current mean error is less than a certain threshold. In our case we used 0.001 inch as the threshold since we are taking scans very close together. We have a maximum iteration of 50 which is a good amount and gives us 100% convergence each time.

4. Data Association Structure- These structures are used to store the corresponding data points in an optimized way. For our implementation we used the matrices from the Python Numpy library to store these.

5. Transformation- This transformation is calculated by using the matrices of the two scan points. We calculate the dot product of A.T and B and then use Single Value Decomposition function results from the Numpy library to calculate the rotation and value between the two scans.

6. Loop Closure- Since we are using ICP for the entire SLAM the accuracy is high enough that we do not need loop closure for determining the map. In this case loop closure will only be needed for determining which path to explore next.

D. Path Planning Subsystem

1. Local Planner

The local planner calculates the distance and angle between current pose of the robot, published by SLAM subsystem, and the destination coordinate, published by the global planner. Based on the distance and angle, the local planner will accelerate and decelerate the linear and angular velocities at a constant acceleration rate, always plateauing at a set maximum velocity. The distance and angle are continuously updated as the robot moves, forming a feedback loop.

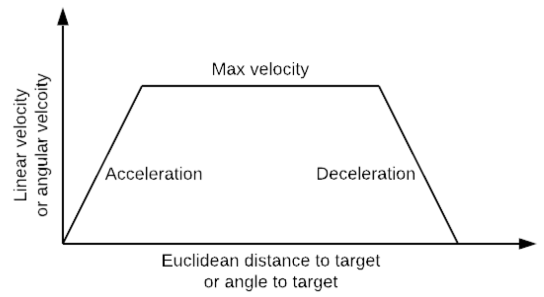


Fig. 8. Local Planner Map

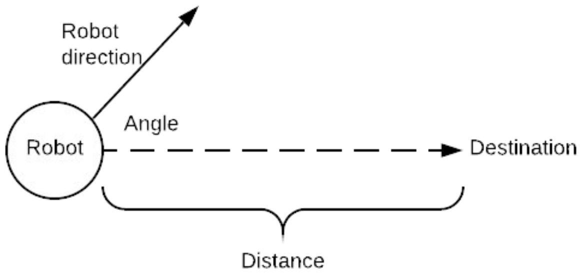


Fig. 9. Local Planner Loop

2. Global Planner

Due to implementation delays and time constraints, the global planner was replaced by a remote control interface to allow teleoperation of the robot. The remote control interface was necessary to test SLAM, because if we move the robot around manually to collect lidar data, our body will interfere with the data and cause errors in SLAM. The remote control interface creates a small window showing a coordinate frame, where the x and y axes correspond to the angular and linear velocities of the robot respectively. The linear speed is bound by 0.5 m/sec for both forwards and backwards movement, and the angular speed is bound by 5.0 rad/sec for both left and right turns. This interface allows variable speed control similar to a physical joystick controller, and is intuitive to use.

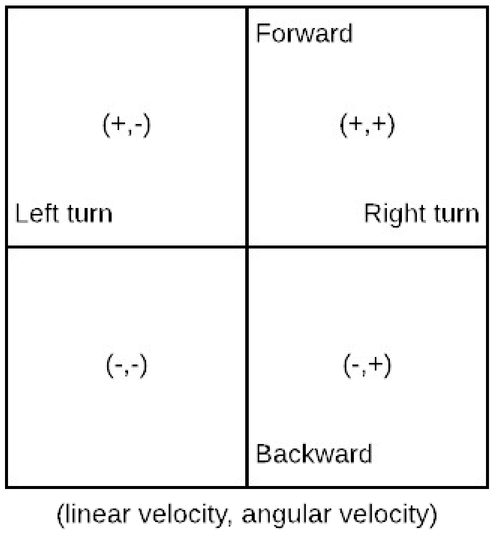


Fig. 10. Teleop Virtual Joystick Interface

E. 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 the updated map to a Node.js web server through POST requests. 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.

F. Motion Subsystem

1. Motor- 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.

2. Motor Controller- We used 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 used to control the motor duty cycle.

VI. Project Management

A. Schedule

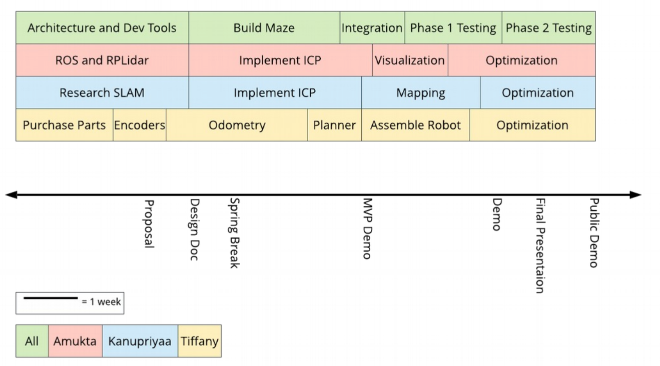


Fig. 11. Project Timeline

Refer to Appendix C for Gantt Chart

B. Member Responsibilities

Amukta was responsible for web application design and implementation for map visualization, Raspberry Pi/ ROS/VM setup, lidar data input and preprocessing, ICP setup and implementation.

Kanupriyaa was responsible for SLAM research, finalizing calculations and algorithm for our implementation of SLAM, ICP setup and implementation, mapping and sending the map representation to web UI.

Tiffany was responsible for researching and ordering components, ROS integration, electrical system design, mechanical system design, encoder interface, odometry, motion, and path planning (both local and global) software subsystems.

Everyone as a team worked on software architecture design, writing reports, building the maze, and doing Phase 1 and 2 testing.

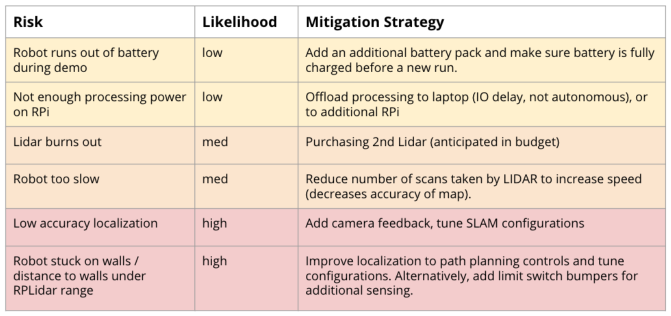
C. Budget

The three most expensive components of the robot were 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. The RPLidar was sufficient and none of the foreseeable risks occurred (see Risk Management), which brought the total cost of the robot to about $250 dollars.

If all the foreseeable risks had occurred, we would have needed to buy an additional RPLidar and an ultrasonic sensor, resulting in a total cost of about $370 dollars. Either way, we were well under the $600 limit. Refer to the Bill of Materials in the Appendix for more details. Refer to Appendix D for our full budget.

D. Risk Management

We have outlined our major risks and anticipated their costs in the budget.



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. 12. Risk Assessment

VII. Related Work

Paper [1] talks about a new approach to ICP based SLAM which uses both pose graph optimization and ICP to build a map with better accuracy.

1. Construction- 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 (Iterative Closest Point).

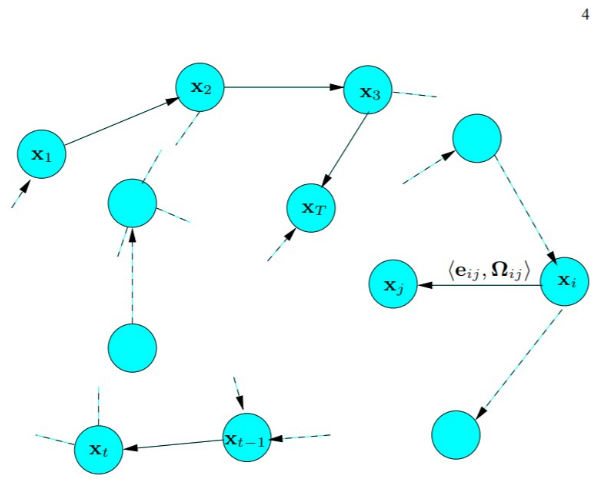


Fig. 13. SLAM Graph Construction through ICP

2. Graph Optimization- Once the graph is constructed the researchers use the configuration of the robot poses that best satisfies the constraints. Thus, in this paper using 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) using Probability Distribution:

$$P(X2 \mid X1 , U, M, Z) = P(Z \mid X2 , M) P(X2 \mid X1, U)$$

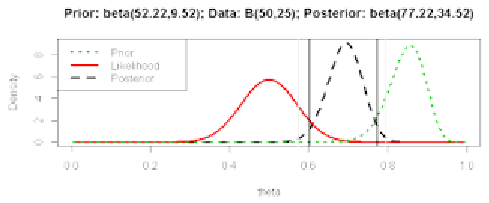


Fig. 14. Prior Distribution of Probability and Probability Equation

Other work is being on using different combination of ICP modules such as data matching and covariance functions to determine what results in best accuracy. The DARPA Project at CMU is also aiming to build robotic vehicles that can map unknown terrains by themselves.

VIII. Summary

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

What we learnt throughout this entire semester first and foremost was the importance of team work and open communication. Our team did an exemplary job in each member doing their own parts and holding others accountable for theirs.

In terms of the development of the project, we learnt that learning new technology is harder and takes longer than actually developing the module. The importance of early integration also came into play starkly when our mid-term demo was not up to standards and we had to work very hard to get our project together after that. All in all, this was a team effort and the credit goes to all members of the team equally.

IX. 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:

1. Low light/ dark areas
2. Bumpy/rocky/grassy grounds
3. Curved/ bumpy walls of various heights
4. 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.

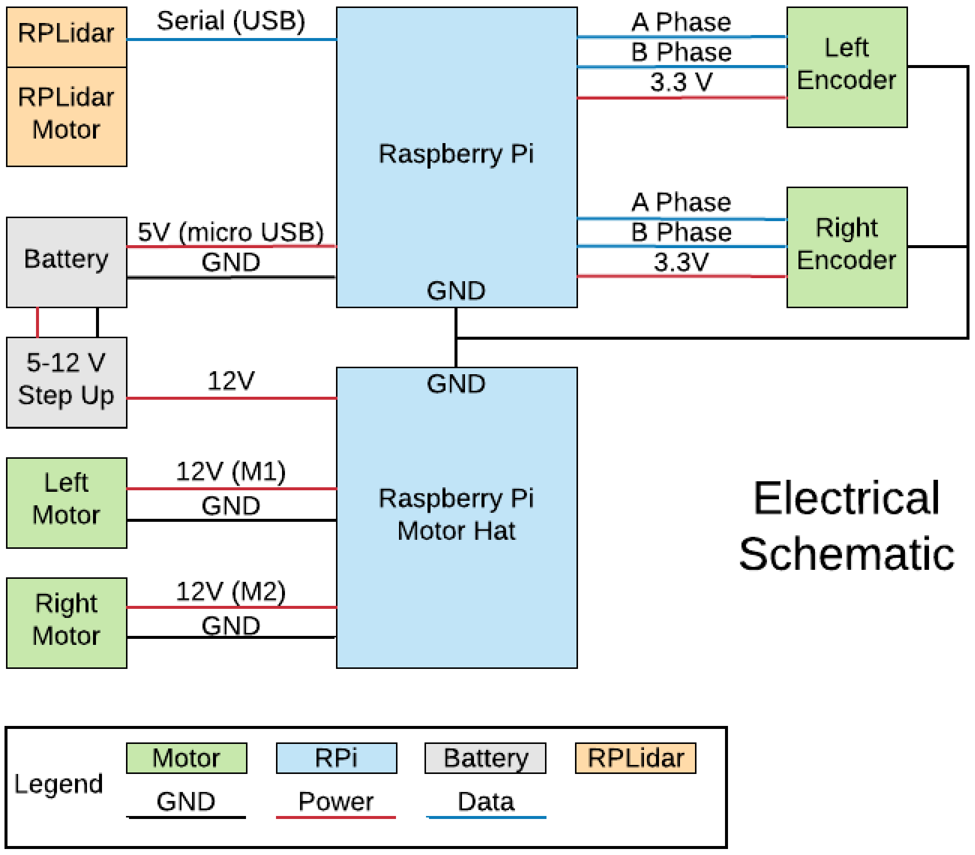
[9] Pomerleau, François, et al. “Relative Motion Threshold for Rejection in ICP Registration.” Springer Tracts in Advanced Robotics Field and Service Robotics, 2010, pp. 229–238., doi:10.1007/978-3-642-13408-1_21.

[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.

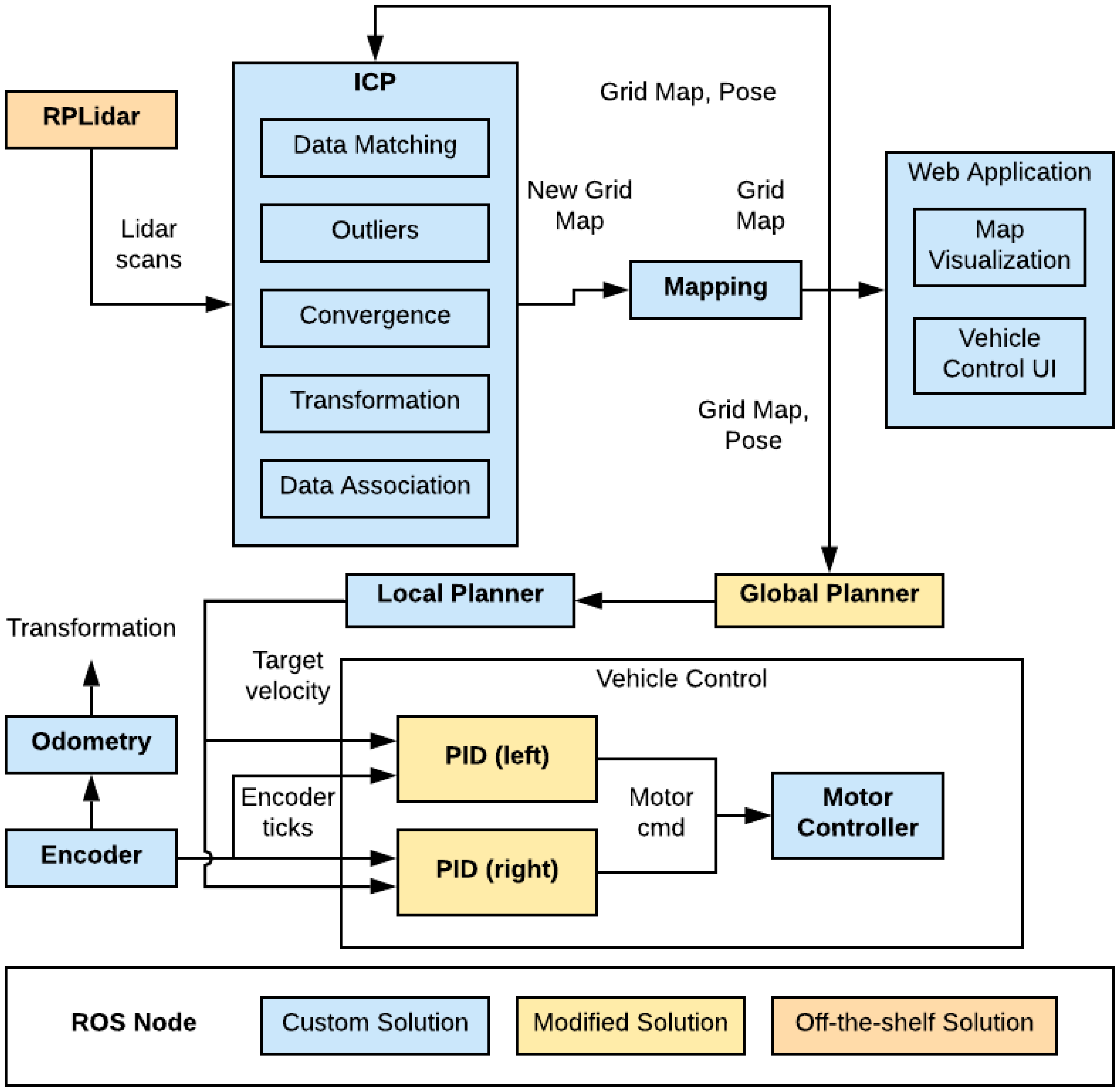
[11] Thrun, S., et al. “A Real-Time Algorithm for Mobile Robot Mapping with Applications to Multi-Robot and 3D Mapping.” Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065), doi:10.1109/robot.2000.844077.

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

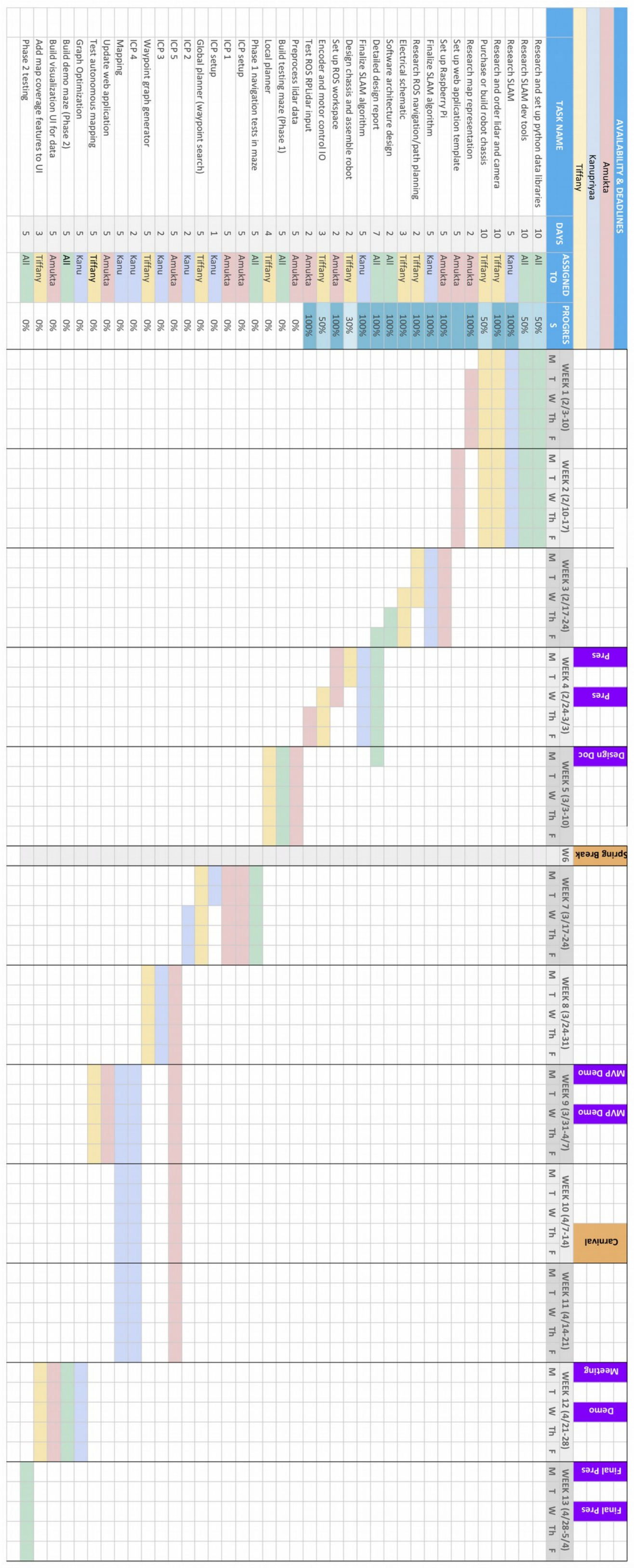
Appendix A: Electrical Schematic



Appendix B: System Block Diagram

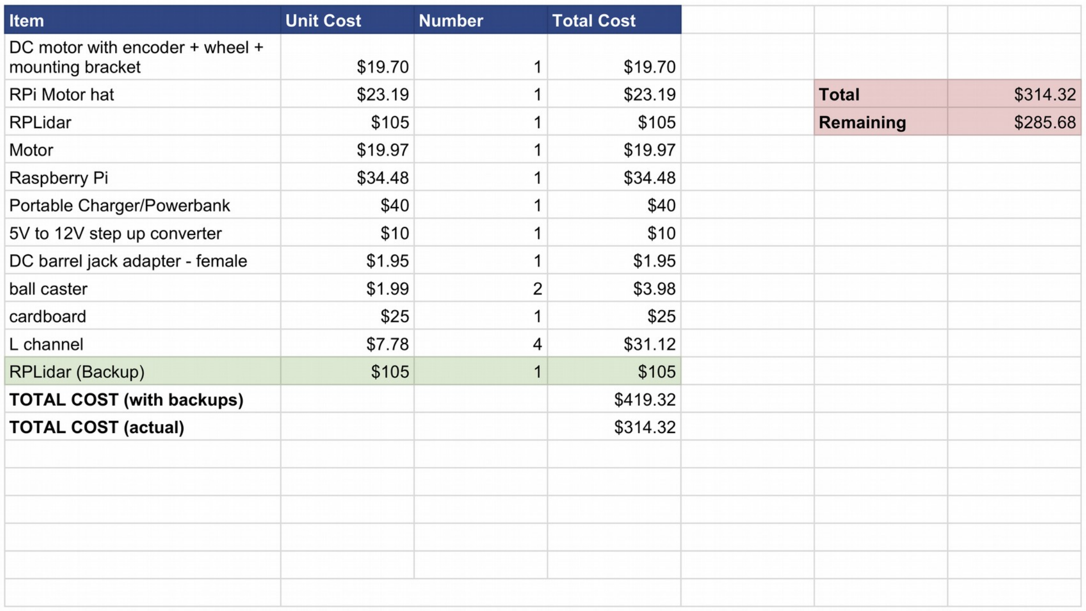


Appendix C: Gantt Chart



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

Appendix D: 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		**Total**	$314.32
RPLidar	$105	1	$105		**Remaining**	$285.68
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			
DC barrel jack adapter - female	$1.95	1	$1.95			
ball caster	$1.99	2	$3.98			
cardboard	$25	1	$25			
L channel	$7.78	4	$31.12			
RPLidar (Backup)	$105	1	$105			
TOTAL COST (with backups)			$419.32			
TOTAL COST (actual)			$314.32			