EE175AB Final Report

MR.SCOT: Multi-Robot System for Collaborative Object Transport

EE 175AB Final Report

Department of Electrical Engineering, UC Riverside

Project Team Member(s)	Gustavo Correa, Vincent Tran, Kyle Semelka
Date Submitted	March 18th, 2019
Section Professor	Dr. Roman Chomko
Project Advisor	Dr. Konstantinos Karydis
Revision	1.1
URL of Project Wiki/Webpage	Project Page - https://www.gustavojcorrea.com/cooperative-transport ROS Package - https://github.com/gustavojcorrea/turtlebot_mrs Documentation - https://github.com/gustavojcorrea/ucr-ee-senior-design Youtube Video - https://youtu.be/aWxMVJgCwsQ
Permanent Emails of all team members	Kyle Semelka - kylesemelka@gmail.com Gustavo Correa - gustavojcorrea1@gmail.com Vincent Tran - vincentkimtran@gmail.com

Summary

This report presents the Technical Report for MR.SCOT, the Multi-robot System for Collaborative Object Transport.

Revisions

Version	Description of Version	Author(s)	Date Completed	Approval
0.1	Modified template of report	Gustavo Correa, Kyle Semelka, Vincent Tran	03/13/2019	
0.5	First draft	Gustavo Correa, Kyle Semelka, Vincent Tran	03/16/2019	
1.0	Submission version of report	Gustavo Correa, Kyle Semelka, Vincent Tran	03/18/2019	
1.1	Reformatted version of v1.0	Gustavo Correa, Kyle Semelka, Vincent Tran	03/18/2019	

Table of Contents

1 Executive Summary

The aim of this research project is to develop the hardware and software for **MR. SCOT**, a **M**ulti-**R**obot **S**ystem for **C**ollaborative **O**bject **T**ransport. Such transport systems enable the human operator to transport diverse objects while reducing physical and mental strain. The three-person research project, in collaboration with the Electrical Engineering Senior Design project, **will investigate the simultaneous localization and mapping (SLAM), docking procedure, and coordinated movement of multiple robots**.

MR.SCOT is scalable and has been designed to **transport diverse shapes of objects** by latching an N number of TurtleBot (Roomba shaped) robots in different configurations (e.g., linear, square). Depending on the shape of the object to be transported, the robots can change their configuration to accommodate objects such as a long box or couch. **The current prototype for MR.SCOT consists of two TurtleBot robots as seen in Figure 1.**

Figure 1: Final prototype of MR.SCOT

The hardware for MR.SCOT consists of the TurtleBot robot (v2.0, Kobuki) as the robot base, a chassis constructed with PVC tubing and custom 3D printed parts, and a carrying platform with 3D printed mounts for the latching mechanism and plexiglass to support the carried object. The TurtleBot robot is powered with 14.8V 4400 mAh Lithium-Ion batteries and the Linux PC (Intel Nuc) is either powered by a 19V portable battery pack or 19V 65W power supply. The Arduino Uno, Load sensors, RGB camera, and LIDAR sensor are powered by the 5V output of the Intel Nuc while the electromagnet circuit used for docking is powered by the 12V/1.5A output of the TurtleBot robot.

The software for MR.SCOT was programmed in C++ and Python for modules run in the Intel Nuc and Robot Operating System (ROS), or C++ for modules run in the Arduino Uno. One Intel Nuc Linux computer with ROS serves as the **master** which runs the graphical user interface (GUI) and state machine with modules for trajectory planning, waypoint navigation, docking, and multi-robot coordination. A second and third Intel Nuc with ROS serve as **slaves** and are placed on top of the robots. These computers run the ROS Navigation Stack which accepts input from the RGB Camera and LIDAR sensor to perform sensor fusion, SLAM, and waypoint navigation. All three Intel Nuc computers communicate using a wireless network router and ROS communication protocols.

2 Introduction

2.1 Design Objectives and System Overview

The objective of our group is to design a scalable autonomous multi-robot system which can dock robots together and transport objects to a specified location on map. Such transport systems enable the human operator to transport diverse objects while reducing physical and mental strain.

The following technical objectives were in mind when designing the system. First we designed the system to perform simultaneous localization and mapping (SLAM). We were able to achieve SLAM in lab rooms in the engineering building, as well as the outdoor patio. We were able to demonstrate autonomous docking of two robots using AR tags and RGB Camera. We demonstrated coordinated movement and implemented the movement with a differential controller. We also designed a scalable multi-robot system for an N number of robots.

This project overall utilized knowledge in autonomy, simultaneous localization and mapping, path planning, and multi-robot coordination. The EE144 Intro to Robotics class provided the foundation in robotics concepts for the group to use and build upon.

The following figure describes the high-level design of the multi-robot system. We achieved an error of <5 cm for way point navigation using the navigation stack, achieved an error of <2 cm in the y axis for docking, and achieving multi-robot coordination where the overall multi-robot system has an angular velocity of 0.2 m/s while turning.

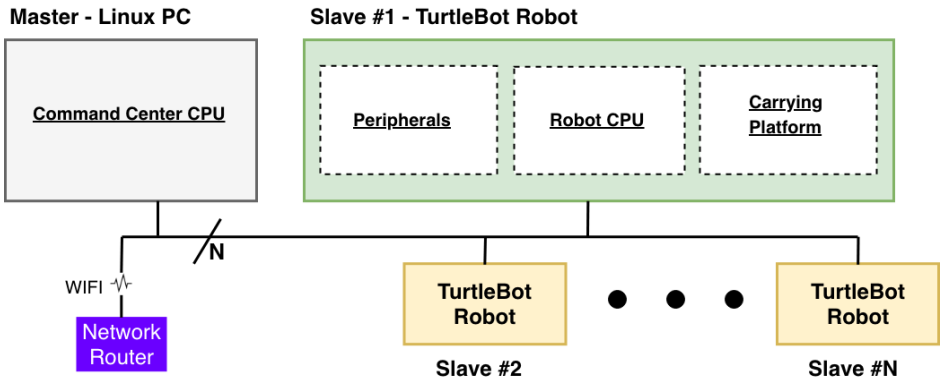


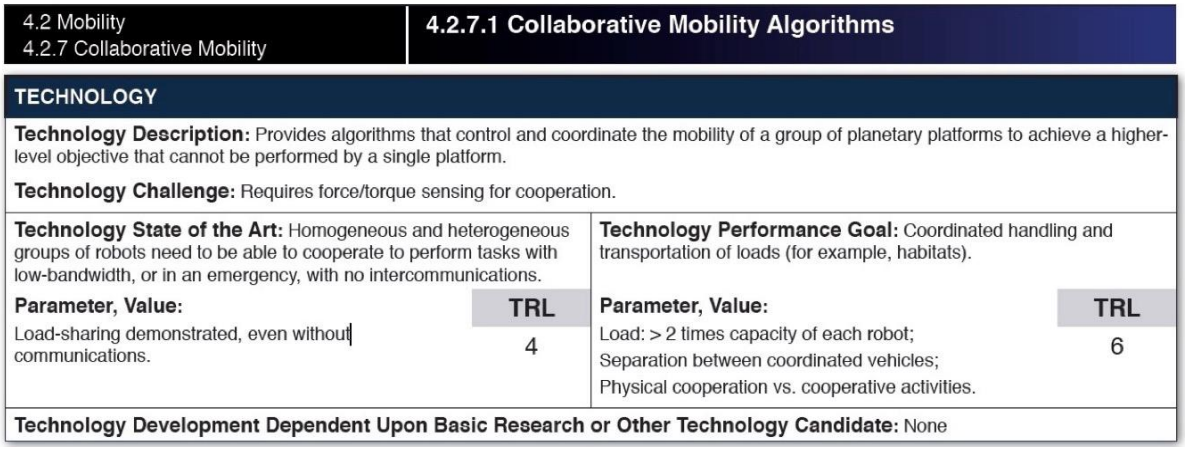
Figure 2: High-Level Block Diagram

Vincent was responsible for integrating the LIDAR sensor into the navigation stack, Gustavo was responsible for developing the state machine and the modules within for docking and multi-robot coordination, and Kyle was responsible for developing the graphical user interface and electronics for the load

sensors and electromagnets. This project is meaningful because it serves as both a research platform as well as an education platform that engineering students can use to apply concepts in robotics.

2.1.1 NASA Technology Roadmap Excerpts (2015)

The following are screenshots are goals which NASA has outlined to accomplish between 2015-2035. See reference [6] for more information.



4.2 Mobility 4.2.7 Collaborative Mobility	4.2.7.1 Collaborative Mobility Algorithms

TECHNOLOGY

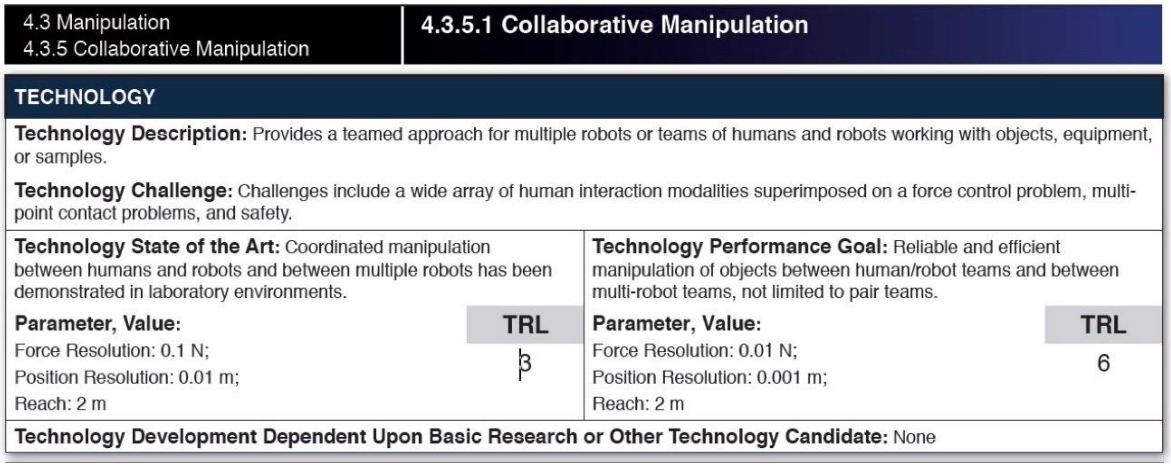
Technology Description: Provides algorithms that control and coordinate the mobility of a group of planetary platforms to achieve a higher-level objective that cannot be performed by a single platform.

Technology Challenge: Requires force/torque sensing for cooperation.

Technology State of the Art: Homogeneous and heterogeneous groups of robots need to be able to cooperate to perform tasks with low-bandwidth, or in an emergency, with no intercommunications.		**Technology Performance Goal:** Coordinated handling and transportation of loads (for example, habitats).	
Parameter, Value: Load-sharing demonstrated, even without communications.	**TRL** 4	**Parameter, Value:** Load: > 2 times capacity of each robot; Separation between coordinated vehicles; Physical cooperation vs. cooperative activities.	**TRL** 6

Technology Development Dependent Upon Basic Research or Other Technology Candidate: None

Figure 3: NASA Collaborative Mobility Algorithms Roadmap



4.3 Manipulation 4.3.5 Collaborative Manipulation	4.3.5.1 Collaborative Manipulation

TECHNOLOGY

Technology Description: Provides a teamed approach for multiple robots or teams of humans and robots working with objects, equipment, or samples.

Technology Challenge: Challenges include a wide array of human interaction modalities superimposed on a force control problem, multi-point contact problems, and safety.

Technology State of the Art: Coordinated manipulation between humans and robots and between multiple robots has been demonstrated in laboratory environments.		**Technology Performance Goal:** Reliable and efficient manipulation of objects between human/robot teams and between multi-robot teams, not limited to pair teams.	
Parameter, Value: Force Resolution: 0.1 N; Position Resolution: 0.01 m; Reach: 2 m	**TRL** 3	**Parameter, Value:** Force Resolution: 0.01 N; Position Resolution: 0.001 m; Reach: 2 m	**TRL** 6

Technology Development Dependent Upon Basic Research or Other Technology Candidate: None

Figure 4: NASA Collaborative Manipulation Roadmap

4.5 System-Level Autonomy 4.5.4 Multi-Agent Coordination	4.5.4.1 Multi-Agent Coordination	
TECHNOLOGY		
Technology Description: Provides an infrastructure for distributing autonomous functionalities across platforms. **Technology Challenge:** Challenges due to the heterogeneity of the hardware and software tools that are interfaced. Challenges include verification and validation of the complex agent interactions, and the capability and management of agent system group goal direction.		
Technology State of the Art: Multi-agent systems have been used for dynamic load balancing of networked systems.		**Technology Performance Goal:** Ability to support heterogeneous hardware and software applications.
Parameter, Value: Range of hardware and software systems that can be integrated: very limited; Range of operations that can be performed: very limited.	**TRL** 4	**Parameter, Value:** System responsiveness; System reliability.	**TRL** 6
Technology Development Dependent Upon Basic Research or Other Technology Candidate: None		

Figure 5: NASA Multi-Agent Coordination Roadmap

2.2 Backgrounds and Prior Art

Designing a single robot to achieve a simple task such as carrying a box is feasible and has been achieved. Distribution companies like Amazon currently utilize teams of robots, each individually transporting package towers across warehouses (see Figure 6). Using teams of robots leads to an increased scalability of warehouse capabilities, improves shipping efficiency, and reduces strain on the human operator. **A single robot in this type of system, however, is limited by its payload capacity, size, and shape.**

Figure 6: Amazon Robot

In a cooperative multi-robot system, single robots can be combined together in different configurations to transport objects of diverse shapes. Approaches include projects at UPenn (See Figures 7 & 8), Stanford (Figure 9), MIT (Figure 10A/B). **The approach for our multi-robot system transports objects by interlocking 2 TurtleBot (Roomba shaped) robots in different configurations.** Depending on the shape of the object to be transported, the robots can change their configuration to accommodate objects such as a long box or couch.

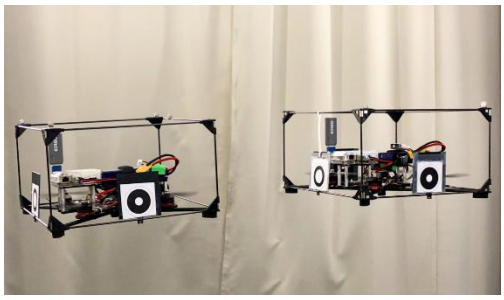


Figure 7: ModQuad-Vi

https://www.modlabupenn.org/2019/03/05/4806/

Project name: ModQuad-Vi: A Vision-Based Self-Assembling Modular Quadrotor
Organization: University of Pennsylvania
Authors: M. Yim et al., 2019

Description:
This project presents a modular robot system consisting of flying robots that localize each other using vision tags and dock together using magnets. (*see Reference [3]*)

Figure 8: ModQuad

https://www.modlabupenn.org/2018/03/18/a-flying-gripper-based-on-cuboid-modular-robots/

Project name: A Flying Gripper Based on Cuboid Modular Robots
Organization: University of Pennsylvania
Authors: M. Yim et al., 2018

Description:
This project presents a modular robot system consisting of flying robots that change configurations to manipulate objects. (*see Reference [2]*)

Figure 9: OuijaBots

https://msl.stanford.edu/multi-robot-manipulation-without-communication

Project name: OuijaBots: Omnidirectional Robots for Cooperative Object Transport with Rotation Control using No Communication
Organization: Stanford University
Authors: M Schwager et al., 2016

Description:
This project presents an omnidirectional robot platform which cooperatively manipulates an object without communication. (*see Reference [5]*)

Figure 10A: Manipulation of deformable object.

https://www.youtube.com/watch?v=mAt_Olb0O7U&feature=youtu.be

Project name: Local Motion Planning for Collaborative Multi-Robot Manipulation of Deformable Object
Organization: MIT and ETHZ
Authors: D. Rus et al., 2013

Description:
This project presents a multi-robot platform which can manipulate deformable objects. (*see Reference [4]*)

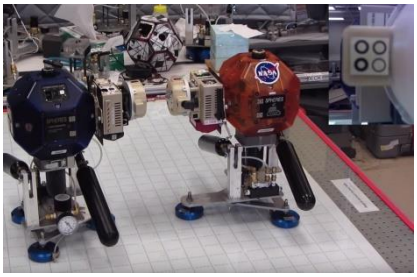


Figure 10B: SPHERES

https://www.youtube.com/watch?v=ZC37OzfIkjo&t=9s

Project name: SPHERES Docking Port
Organization: MIT
Authors: D. Miller et al.

Description:
This project presents a multi-robot platform which aims to simulate docking in a zero-g environment. (*see Reference [1]*)

2.3 Development Environment and Tools

2.3.1 Hardware Tools and Technologies

- Electronics testing and soldering
    - External variable power supply initially used to power on the electromagnet
    - Digital multimeter used to measure the current and voltage utilized for the electromagnet
    - Hakko FX-600 soldering iron
- Autodesk EAGLE: Schematic/PCB design software used to create low-level schematic
- SolidWorks: 3D CAD software used to design the robot chassis and mounts for parts
- Makerbot Replicator +: 3D printer and software used to 3D print all parts

2.3.2 Software Tools and Technologies

- Ubuntu 16.04.6 LTS on Intel NUC
- Robot Operating System (ROS Kinetic Kame) - used for 95% of software
- Web Development
    - Apache HTTP Web Server - utilized to create a remote GUI
    - HTML5/CSS3/JavaScript - utilized to design the Web GUI
    - Bootstrap (v4.3.1) - utilized as a template for Web GUI modules
- MATLAB (v2017a) - utilized to process data from experiments and generate graphs
- Arduino IDE (v1.8.9) - utilized to program the Arduino Uno microcontroller

2.3.3 ROS Packages (descriptions are quoted from the package documentation)

- **SMACH:** *a task-level architecture for rapidly creating complex robot behavior*
    - Utilized to create a state-machine for the project
- **roslibjs:** *core JavaScript library for interacting with ROS from the browser*
    - Connects GUI web application to ROS
- **ros2djs:** *standard JavaScript 2D visualization manager for ROS*
    - Utilized to display the map and camera on the Web GUI
- **nav2djs**: *a tool to display and interact with a robots autonomous navigation capabilities*
    - Utilized to send navigation waypoint goals from the GUI to the robots
- **ar_track_alvar:** *an open source AR tag tracking library*
    - Utilized to track AR tags on the robots
- **navigation:** *a 2D navigation stack that takes in information from odometry, sensor streams, and a goal pose and outputs safe velocity commands that are sent to a mobile base*
    - Utilized to accept input from the LIDAR/RGB camera to create a map and output velocities to the robots
- **gmapping:** *a package that provides laser-based SLAM*
    - Utilized to create a 2D occupancy grid map from the lidar and pose data collected from the robots
- **multirobot_map_merge**: *merging multiple maps without knowledge of initial positions of robots*
    - Merge two different maps from two robots
- **rrt_exploration**: *implements a multi-robot RRT-based map exploration algorithm*
    - Rapidly-Exploring Random Tree (RRT) algorithm.
- **kobuki**: *holds the ROS wrapper of Kobuki's C++ driver plus various ROS tools and applications*
    - Utilized to send velocities to the TurtleBot Robots

2.4 Related Documents and Supporting Materials

ROS documentation - http://www.ros.org/

See the References Section

2.5 Definitions and Acronyms

MR.SCOT - Multi-robot System for Collaborative Object Transport

PCB(s) - Printed Circuit Board(s)

ROS - Robot Operating System

DWA - Dynamic Window Approach

GUI - Graphical user interface

Web - Short for website

HTML - Hypertext Markup Language

CSS - Cascading Style Sheets

JS - Javascript

3 Design Considerations

3.1 Realistic Constraints

3.1.1 Weight and Size Constraints

The weight and size of each robot in the multi-robot system is constrained to the payload capacity and size of the TurtleBot robot. According to the TurtleBot specifications, the TurtleBot robot has a 5 kg payload capacity when run on hard floors and a 4 kg payload capacity when run on carpet. We also constrained the height of the robot to reduce inertial resistance during movement.

3.1.2 Voltage/current Supply Constraints

To increase runtime of the robots we took into consideration the "Motor Overload Detection" feature of the TurtleBot which disables power on detecting high current (>3A) as noted in the TurtleBot specifications. The voltage and current draw of the electromagnet was also constrained to 12V/1.5A since the electromagnet was powered with the batteries powering the TurtleBot robot.

3.1.3 Velocity Constraints

The translational and rotational velocity of the TurtleBot has a maximum of 0.7 m/s and 180 deg/s, however we constrained these velocities to lower values to reduce problems with odometry measurements and LIDAR sensing.

3.1.4 Simultaneous Localization and Mapping (SLAM) Constraints

A constraint we had was localizing the robots without using an external positioning systems such as VICON optical motion capture cameras or GPS sensors. We opted to use the RPLIDAR A2 360 degree laser range scanning LIDAR which came with distance, height, and obstacle detection constraints. Although we were able to achieve simultaneous localization and mapping (SLAM) with the LIDAR sensor and ROS Navigation package, we were also constrained by the accuracy of the software modules in the Navigation stack.

3.1.5 Docking and Coordinated Movement Constraints

The first constraint we had for docking two robots together was the nonholonomic and velocity constraints of the chassis of both robots. This constraint prevents the mobile robots from moving laterally as seen in robots with omniwheels or mecanum wheels. (See the modern robotics book chapter 13). Another constraint for docking was designing the latching mechanism so that the two robots could dock together within a horizontal tolerance of 2 cm. Testing on the docking procedure is explained more in detail in **Section 11.**

3.1.6 Technical and Skill Constraints

This project requires an in depth understanding of the Robot Operating System architecture as well as fundamental concepts in robot kinematics, control, and planning.

3.1.7 Hardware Constraints

Hardware constraints include the battery runtime of the TurtleBot robot which is around 3 hours with the small version of the battery and around 2 hours with the portable battery packs.

3.1.8 Computing Performance Constraints

Computing requirements for the Robot Operating System led us to choose the Intel Nuc for the Linux PC.

3.2 System Environment and External Interfaces

Describe the system, hardware and software that your product must operate in and interact with, any communication protocols and APIs the system must comply to, etc.

3.3 Industry Standards

Robot Operating System

802.11n Wi-Fi (450Mbps transfer rate)

USB (480 Mbps transfer rate)

3.4 Knowledge and Skills

The development of MR.SCOT requires an understanding of robotics, embedded systems programming, physics, and physics.

Courses and Experience

Gustavo Correa

- EE/CS 120A/B, CS 122A - Beginner, Intermediate Embedded Systems
- EE 255 - Real-time Embedded Systems (Graduate Course)
- EE 1A/B - Engineering Circuit Analysis
- CS 10/13 - Introduction to Computer Science (C++)
- CS 14, CS 100 - Data Structures & Algorithms, Software Construction
- EE 144 - Introduction to Robotics
- ME 002, ME 009 - Introduction to Mechanical Engineering, 3D Solid Modeling with SolidWorks
- Previous Experience:
    - UCR ARCS Lab Research (2018-2019): Developed PCBs and ROS software
    - Summer Research at UC Berkeley (2018): Learned ROS and AR Tag Tracking

Kyle Semelka

- EE/CS 120A/B, CS 122A - Beginner, Intermediate Embedded Systems
- EE 255 - Real-time Embedded Systems (Graduate Course)
- EE 1A/B - Engineering Circuit Analysis
- CS 14, CS 100 - Data Structures & Algorithms, Software Construction
- Previous Experience:
    - Software Engineering Job at ASPIN Lab (2018 - Present)

Vincent Tran

- EE/CS 120A/B, CS 122A - Beginner, Intermediate Embedded Systems

- EE 255 - Real-time Embedded Systems (Graduate Course)
- EE 1A/B - Engineering Circuit Analysis
- EE 144 - Introduction to Robotics
- Previous Experience:
    - Internship at Aerospace Corp (2018)

New Knowledge and Skills

- SMACH ROS package for state machine programming
- Website graphical user interface development
- Robot docking implementation
- Differential drive controller for a multi-robot system
- Latch mechanism using electromagnets

3.5 Budget and Cost Analysis

3.5.1 Items Borrowed

Table 1: Borrowed items.

Item	Price	Qty.	Subtotal
TurtleBot 2.0 (Kobuki)	$320.00	2	$640.00
Intel Nuc	$400.00	3	$1,200.00
RPLIDAR A2	$320.00	2	$640.00
19V Portable Battery Pack	$50.00	2	$100.00
Arduino Uno	$25.00	1	$25.00
Orbbec Astra Pro 3D Camera	$160.00	1	$160.00
		Total	**$2,765.00**

3.5.2 Items Purchased

Table 2: Purchased items.

Item	Price	Qty.	Subtotal
12V Electromagnet	$24.00	1	$24.00
5VRelay	$10.00	1	$10.00
Load Cell - 50kg	$11.00	4	$44.00
Load Cell Combinator	$2.00	1	$2.00
HX711 Load Cell Amplifier	$10.00	1	$10.00
Molex Connector for Kobuki	$0.45	1	$0.45

Crimp for Molex Connector for Kobuki	$0.15	1	$0.15
		Total	**$90.60**

3.5.3 Cost Analysis

In a scenario where this system would be used to transport objects in a distribution warehouse, the current costs for two robots seems unfeasible for scalability. A solution to this problem would be to optimize the parts list and mechanical design to reduce costs on materials. Multi-robot systems haven already been implemented in distribution warehouses so we can analyze those test cases further to improve our system.

3.6 Safety

Our safety objectives are the following:

1. Must be able to operate autonomously without injuring someone.
2. Must be able to be handled without getting shocked by exposed wire.

One safety consideration had to do with the electromagnet. Since our electromagnet was 12V, so this created a safety issue with powering it. We needed to ensure that any 12V power lines going from the Turtlebot to the Electromagnet were properly insulated and were designed in a way to prevent accidentally touching it when handling the robot. To do this, we utilized proper connectors to the robot using Molex connectors, heatshrink, and electrical tape to cover any exposed pieces of wire.

Another safety consideration was preventing the robots from crashing into people who may walk in the path of them. This problem was easily solved by us choosing the Kobuki TurtleBot. The software and hardware include features that automatically cease operation of the robot if they detect a crash into anything. The hardware for this feature included bumper pads on the front and sides of the robots. Whenever these bumpers are depressed, control is switched in the software to a safety loop which stops all movement.

Building the platform of our robot proved to be a safety consideration for us. We used a dremel to cut plexiglass used for supporting the load cells and also for the platform of both robots. To do this safely, we used work gloves, safety goggles, and standard safety procedures for cutting using a dremel.

3.7 Performance, Security, Quality, Reliability, Aesthetics etc.

Performance: To ensure consistent performance, we work together to overcome bugs in the code that could result in poor performance. We design the software using multithreading, resulting in utilizing the CPU more.

Security: The peripherals of our robot are tucked away underneath our main platform. Users would need to intentionally move components around in order to get to the internals of our robot.

Quality Control: To ensure exceptional quality, we test each component individually before including it in our robot. We also make sure each part is designed to work with the parts it is connected to.

Reliability: We test each feature many times and in varying test cases to ensure it will perform as expected, no matter the situation. This results in a reliable robot that can be trusted to do what it was design to do.

Aesthetics: To improve the aesthetics of our robot, we design the wiring and parts such that they are presentable. For the platform, we choose plexiglass as a material because it looks clean and is transparent, revealing the interesting engineering underneath. We design the 3D printed mounts ourselves, creating an appealing design. We use zip ties to bundle up wires, preventing any loose wires.

3.8 Documentation

To document the senior design project, the group utilized Google Drive as the main online platform to save all files, designs, meeting notes, code, and 3D models. The structure of the senior design Google Drive Folder is outlined below. The group also utilized GitHub as the web-based hosting services for version control using Git. The outline of the GitHub repository is outlined below. For every meeting and every weekly progress report the following details were mentioned: accomplishments from the prior week, challenges and solutions to challenges, action items due by the next meeting, and important deadlines. Gustavo also utilized the Notability IOS application to document sketches and later uploaded them to the Google Drive folder.

- Google Drive Folder: **UCR EE Senior Design 18-19**
    - Applications
        - UCR Mini-Grant Application
    - Assignments
        - Weekly Progress Reports
        - Fall and Winter Slideshow Presentations

        - Essay Reports
        - Block Diagram
        - Senior Design Report
    - Engineering
        - MATLAB Code
        - Mechanical Design Files
        - Software Backups
        - Photos/Videos
        - Project Screenshots
        - Tests Conducted
    - Meetings
        - Group Meetings
        - Meetings with Advisor
    - Purchasing
        - Receipts
    - Resources
        - Datasheets
        - EE144 Intro to Robotics Lab Guides
        - Research Papers
        - Robotics Guides
- GitHub Repositories:
    - **turtlebot_mrs** (Link: https://github.com/gustavojcorrea/turtlebot_mrs)
        - turtlebot_mrs
        - turtlebot_mrs_msgs
        - README
    - **ucr-ee-senior-design** (Link: https://github.com/gustavojcorrea/ucr-ee-senior-design)
        - Documents
        - ROS
        - MATLAB
        - Arduino
        - SolidWorks

3.8.1 Fall Quarter Project Timeline

Date	Milestones Timeline (BLUE: Urgent Deadlines)
Week 1 10/1/2018	• **Everyone:** Drafted 10+ project ideas
Week 2 10/8/2018	• **Everyone:** Research into current multi-robot systems in industry and research universities
Week 3 10/15/2018	• **Everyone:** • Evaluate different robot platforms and finalize the proposal

Week 4
10/22/2018
	• **Everyone:**
	  • Create system and specific block diagrams
	  • Divide and assign tasks
	  • Research into available ROS packages

Week 5 10/29/2018	• **Vincent:** Interface sensors into ROS • **Gustavo:** Simulate Turtlebot in ROS • **Kyle:** Learn ROS and learn about Kalman filter

Week 6
11/5/2018
	• **Vincent:** Program teleoperation node
	  • Integrate Lidar into TurtleBot's URDF
	• **Gustavo:** Position controller
	  • 11/1,11PM @ 381M SKye Hall:Attend Mini-Grant workshop
	• **Kyle:** Start Kalman filter

Week 7 11/12/2018	• **Vincent:** Map a room with Lidar and single TurtleBot • SUBMITTED 11/12: Apply for $1000 UCR Mini-Grant • **Gustav & Kyle:** Program waypoint navigation
Week 8 11/19/2018	• **Vincent:** Combine Lidar with navigation stack • **Gustavo:** Simulate navigation stack with RRT algorithm • **Kyle:** Design platform for robot
Week 9 11/26/2018	• **Vincent:** Complete integration of lidar into navigation stack • **Kyle:** Design platform • **Gustavo:** Develop waypoint program for the showcase
Week 10 12/3/2018	• **Everyone:** Showcase demo
Finals Week	• **Everyone:** Study for finals

3.8.2 Winter Quarter Project Timeline

Date	Milestones Timeline
Week 1 1/7/2019	• **Vincent:** N/A • **Gustavo:** AR Tag tracking with Orbec Astra camera • SMACH state machine tutorials • **Kyle:** Bought parts, researched electromagnet and load cells

Week 2
1/14/2019
	• **Vincent:** N/A
	• **Gustavo:** Build another robot with Kyle
	  • Design entire state machine
	  • Design improved platform
	• **Kyle:** Purchased remaining parts, build another robot

Week 3 1/21/2019	• **Vincent:** Lidar integration with RRT package and ROS nav stack • **Gustavo:** Designed State machine with a menu to send waypoints • CAD Design magnet mount • **Kyle:** Prototype magnet circuit for the latch mechanism
Week 4 1/28/2019	• **Vincent:** Fine tune navigation stack • **Gustavo:** Dock robots together • **Kyle:** Integrate magnet with ROS, setup ROS web GUI
Week 5 2/4/2019	• **Vincent:** Fine tune navigation stack • **Gustavo:** Docking and coordinated movement • **Kyle:** Connect GUI to the state machine
Week 6 2/11/2019	• **Vincent:** Continue from week 5 • **Gustavo:** Continue from week 5 and redesign magnet mount • **Kyle:** Continue from week 5

Week 7
2/18/2019
	• **Vincent:** Perform initial tests on AR tag range and waypoint accuracy
	• **Gustavo:** Built the final prototype of the carrying platform with Kyle
	  • Tested and improved docking and coordinated movement
	• **Kyle:** Improve the functionality of the Web Gui
	  • Attach load sensors to the carrying platform

Week 8 2/25/2019	• **Everyone:** Final project demo and perform tests
Week 9 3/4/2019	• **Everyone:** Slideshow Presentation and perform more tests
Week 10 3/11/2019	• **Everyone:** Senior Design Showcase • Submit the Final Design Report

3.8.3 Fall Quarter 2018 Project Ideas

During Fall Quarter 2018, the team came together to discuss implementation ideas for the multi-robot system. Below are sketches of ideas.

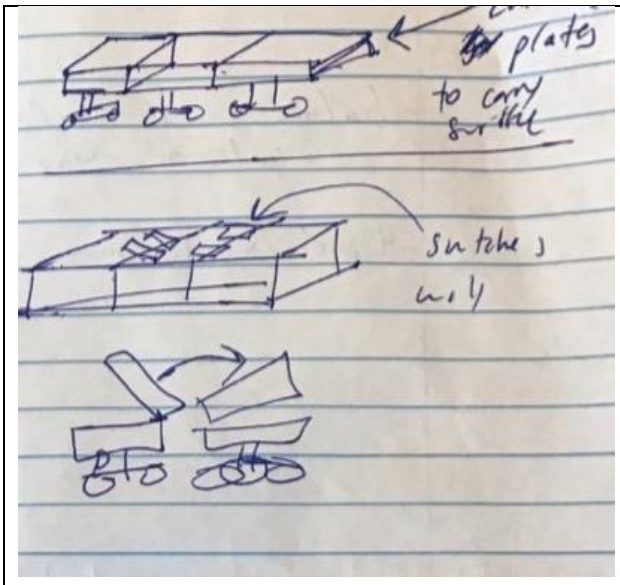


Figure 11: OuijaBots Robots distributing payload amongst themselves.

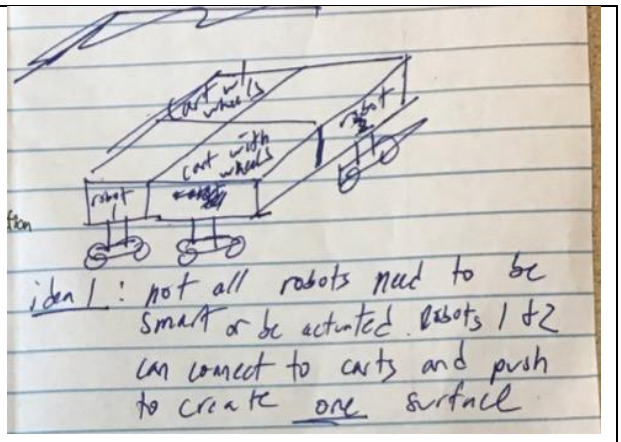


Figure 12: Robots in a 2x2 configuration.

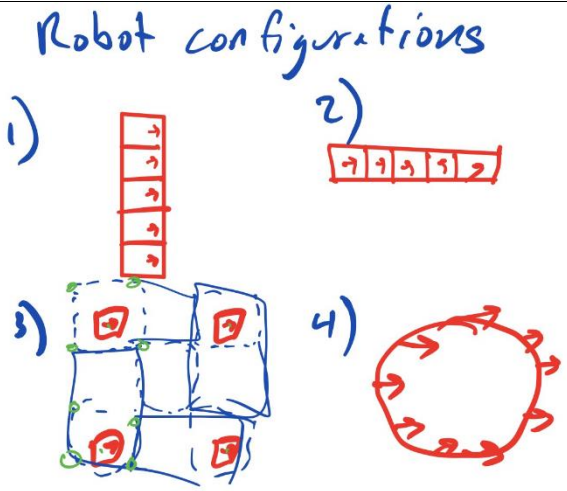


Figure 13: Possible robot configurations.

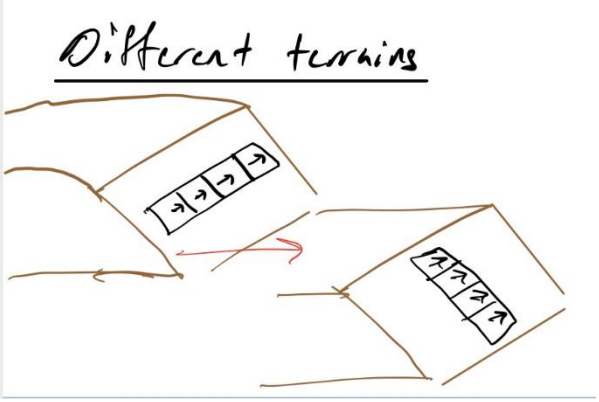


Figure 14: Robots changing configuration depending on the slope incline/decline.

3.9 Risks and Volatile Areas

Not applicable to this report.

4 Experiment Design, Experiment Results, Data Analysis and Feasibility

The following section describes features in our design which required experimentation to determine the feasibility of the proposed solution.

4.1.1 Magnet Mount

Kyle and Gustavo are responsible for this task.

After implementing and fine tuning the docking procedure, an improved latching mechanism had to be designed so that the robots could still dock together given the nonholonomic constraint of the robots (see Sections 3.1.5). The electromagnet was repositioned so that the robot could dock within a certain region.

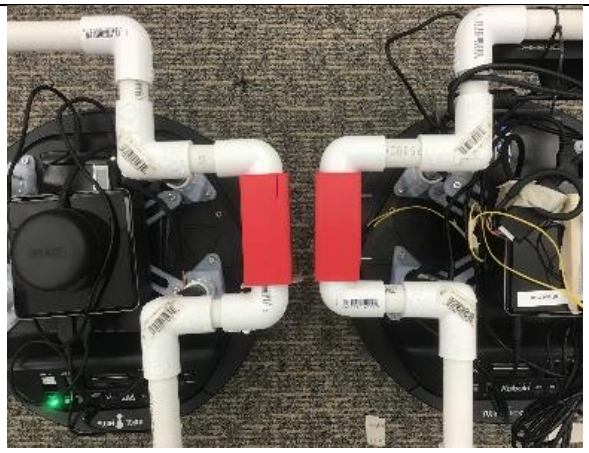


Figure 15: Latch Mechanism v1

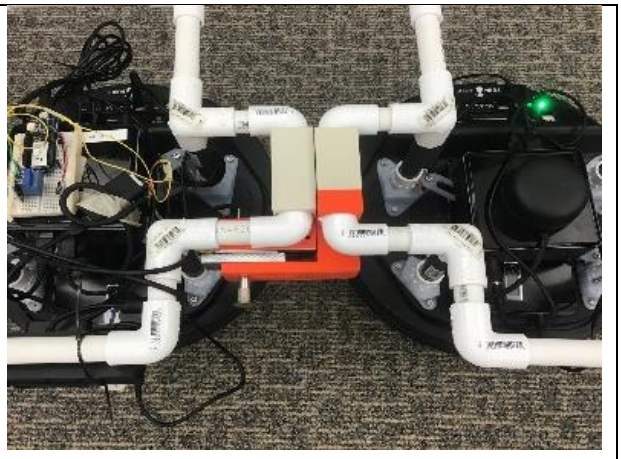


Figure 16: Latch Mechanism v2

4.1.2 Height of Robot

Kyle is responsible for this task.

Originally, the robot used 8” PVC risers to support the platform. During our experiments, we noticed that the robot was not stable when weight was placed towards the edges of the platform. As a result, we chose 4” PVC risers to increase stability and reduce inertial forces when moving.

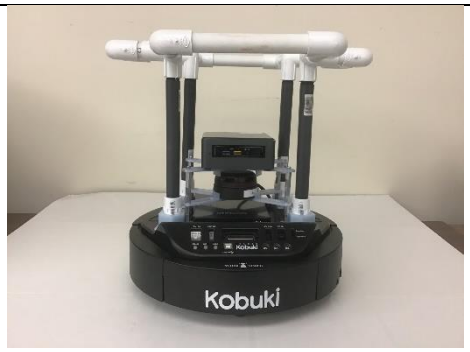


Figure 17: Robot with 8" PVC Risers

Figure 18: Robot with 4" PVC Risers

4.1.3 Number of Robots

Vincent and Gustavo are responsible for this task.

The original idea was to use four robots for the project. This has changed to two robots so that we can implement different control algorithms on two robots.

4.1.4 Localization of Second Robot

Vincent and Gustavo are responsible for this task.

We tried different methods to localize the second robot such as using AR Tags, LIDAR and the ROS mapmerge package, and inverse transformation math. We opted to go with AR tags due to the reliability of this method.

4.1.5 Path Planning

Vincent is responsible for this task.

An important goal for this project was to be able to dock two robots together, which would require a precise path to get the electromagnets touching each other. We will look at a package called rrt_exploration that creates a random path to its goal. We expect it to easily reach its goal.

4.1.6 LIDAR Test

Vincent is responsible for this task.

We needed a 2D laser scanner, such as lidar, to perform SLAM (Simultaneous Localization and Mapping). We were provided with the RPLIDAR which guarantees up to 8 meter range. We will launch the lidar in a room and view if data comes out. We expect to see an outline of what the lidar captures.

4.1.7 Created Map with LIDAR

Vincent is responsible for this task.

To perform SLAM, we will use a ros package called gmapping. We will have a robot roam around the room with the lidar generating the map. We will view what the map looks like and configure the parameters to make it better.

4.1.8 Setup Navigation Stack

Vincent is responsible for this task

We will use the Navigation Stack to send waypoints for the robot to arrive to. We will test a few points such as (1,0), (1,1). We expect the robots to arrive to those points with slight offsets.

4.1.9 Load Sensors

Kyle is responsible for this task

Our original idea was to use a single load cell. However, after some research, we learned that this was going to lead to an unstable upper platform since the weight would only be distributed on a single point in the center. As a result of our research, we chose four load cells in order to distribute the weight evenly and have enough points of weight distribution so that the platform won't wobble.

4.1.10 Electromagnetic Latch

Gustavo and Vincent are responsible for this task

Our first idea for combining the robots into one robot was by using a mechanical latching mechanism. However, this created some difficult problems to overcome, such as how to engage and disengage the latch on command and how to make the latch strong enough to hold both robots together. We decided to use an electromagnet in place of a mechanical latch. This way, we are able to control when the robot engages and disengages the latch very easily and in a stable manner.

4.1.11 Path Planning

Vincent is responsible for this task

The first package we looked at was rrt_exploration, which we ended up not using. This package can provide a random path to get to its destination, meaning that its orientation can cause the docking

platform to collide if the robot isn't parallel when coming in contact. We ended up using the navigation stack because it was precise enough to move our robots to general areas and we can input a desired orientation. Because the navigation stack still had slight precision problems, we used a PID controller as the second controller to reorient and reposition for the final docking procedure.

4.1.12 LIDAR Test

Vincent is responsible for this task

As we could see below, the lidar is able to pick up data from its surrounding environment. The data is a big array of integers, which would be difficult to visualize, but we went with what came out it. It turned out to work really well, as we used other packages such as gmapping to subscribe to the lidar topic. It worked out fine for the navigation stack as well. There wasn't any tuning needed, rather we tuned the other tools that use the lidar.

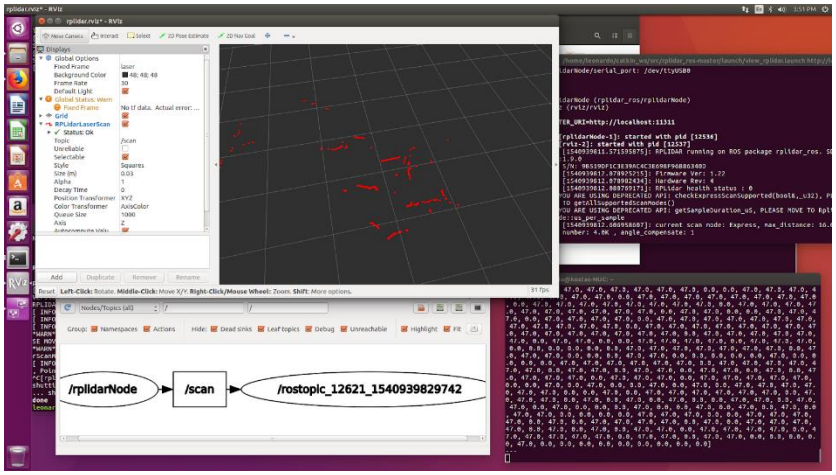
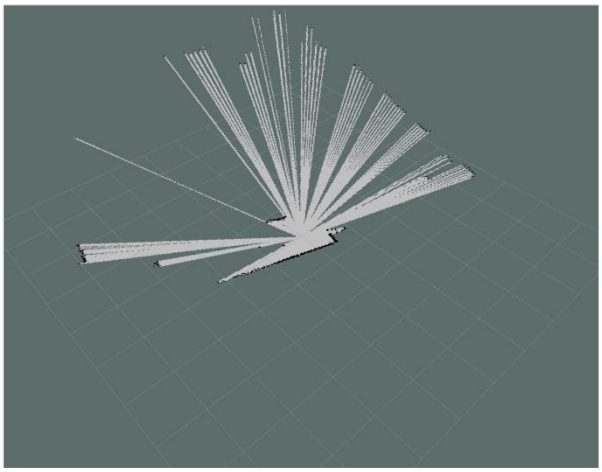
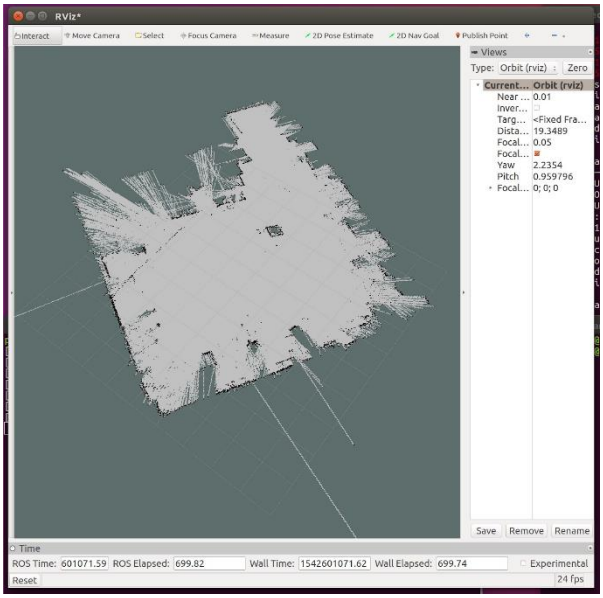


Figure 19: Lidar displayed in RViz.

4.1.13 Created Map with LIDAR

Vincent is responsible for this task

Default Gmapping Settings	Configured Settings
 **Figure 20: Generated map before configuration.**	 **Figure 21: Generated map after configuration.**
As we could see, the generated map is nowhere near legible before we configured some parameters.	After configuring some parameters, the map turned out to be better. With this map, it will better localize the robots, ultimately allowing us to perform much more precise navigating.

4.1.14 Setup Navigation Stack

Vincent is responsible for this task

Overall the navigation turned out to work very well. It allowed us to move to way points precisely after we configured the navigation stack. We also used the navigation stack to be able to return home after it performs the full demo. Without the navigation stack, we wouldn't be as free to move with the robots. More detailed analysis of the navigation stack parameters and tests could be seen in the Tests at Section 11.

5 Architecture and High-Level Design

For the High-Level Design and System Architecture refer to Section 5.1 For the Hardware and Software Architecture refer to Section 5.2 and 5.3, respectively.

5.1 System Architecture and Design

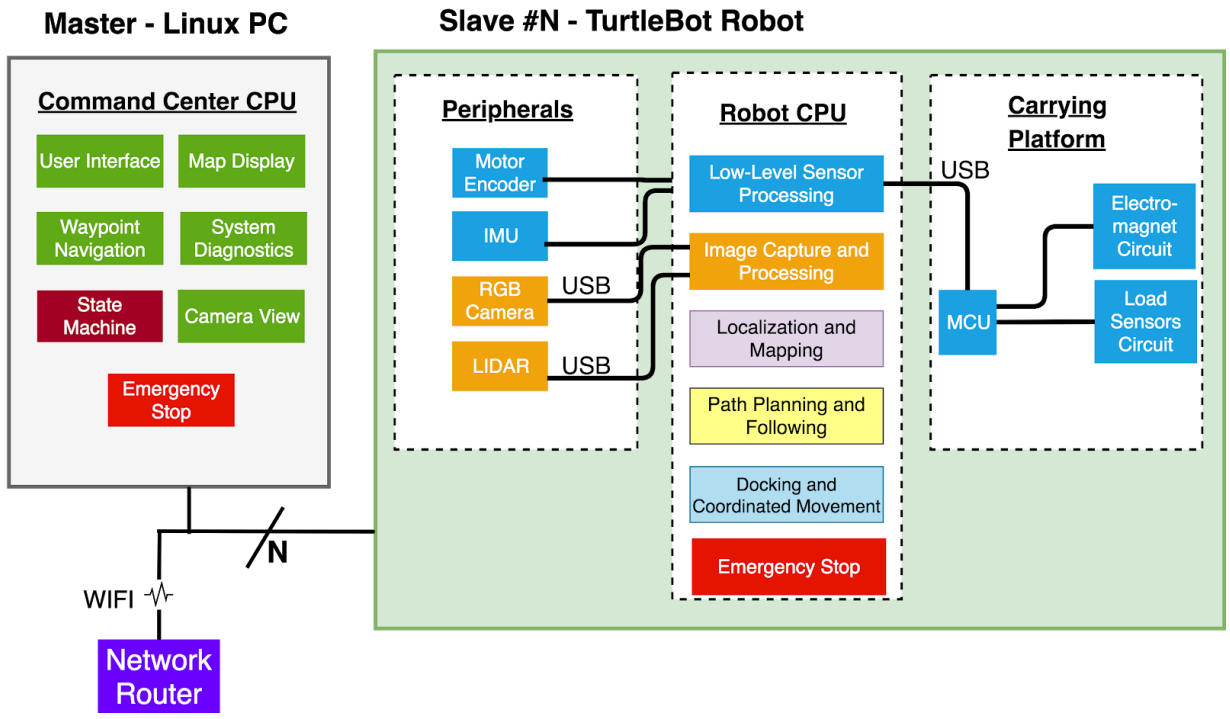


Figure 22: High Level Design System Block Diagram

5.1.1 Command Center CPU

Kyle is responsible for designing and programming the Web-based graphical user interface consisting of the map display, waypoint navigation, system diagnostics, camera view, and state machine viewer.

Gustavo and Vincent are responsible for programming the state machine which integrates the path planning and following, docking, and coordinated movement software modules.

5.1.2 Low-Level Sensor Processing

Vincent is responsible for integrating the motor encoder and Inertial Measurement Unit (IMU) into the Navigation stack which is used to track the odometry of the robots.

Kyle is responsible for programming the microcontroller on the TurtleBot robots which accepts input from the state machine, controls the electromagnets, and publishes values received from the load sensors.

5.1.3 Image Capture and Processing

Vincent is responsible for configuring the LIDAR sensor and using the laser scan values to create a global map using the Navigation stack.

Gustavo is responsible for configuring the RGB Camera to detect and track the AR Tag markers.

5.1.4 Simultaneous Localization and Mapping (SLAM)

Vincent is responsible for configuring the Navigation Stack which simultaneously localizes the robots in the global frame and creates a map with the LIDAR sensor.

5.1.5 Path Planning and Following

Gustavo and Vincent are responsible for configuring the Navigation Stack and PID controller to create paths for the robots to follow.

5.1.6 Docking and Coordinated Movement

Gustavo is responsible for localizing robot_2 relative to robot_1 and integrating the PID controller so that robot_1 can dock to robot_2. Gustavo is also responsible for designing the differential drive controller which models both robots as one unit.

Vincent is responsible for designing the figure-8 motion which both robots demonstrate during the demo.

5.1.7 Carrying Platform and Mechanical Design

Kyle and Gustavo are responsible for designing the chassis of both robots using PVC pipes and custom 3D printed parts.

Gustavo is responsible for designing all 3D printed parts including the mounts for PVC tubing, Intel NUC, LIDAR sensor, and electromagnet.

Kyle is responsible for designing the Arduino C++ software and electronics controlling for the electromagnets and load sensors.

5.2 Hardware Architecture

Figure 23 illustrates the Hardware Architecture Block Diagram.

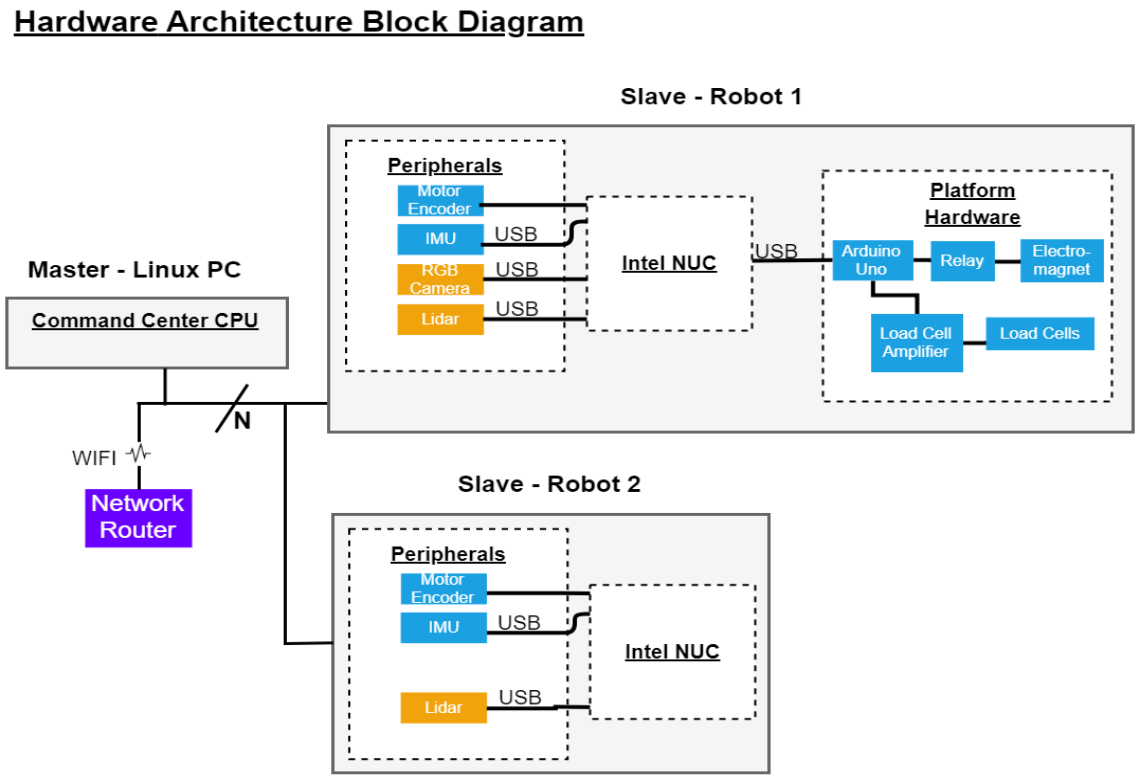


Figure 23: Hardware Architecture Block Diagram

5.2.1 4.1.1 Intel NUC Everyone is responsible for implementing software modules on the Intel Nuc.	**5.2.2 4.2.4 Platform Hardware** See Section 5.1.7.
5.2.3 4.2.2 Motor Encoder and IMU See Section 5.1.2.	**5.2.4 4.2.5 Network Router** Everyone utilizes the network router.
5.2.5 4.2.3 LIDAR Sensor and RGB Camera See Section 5.1.3	

5.3 Software Architecture (only required if your design includes software)

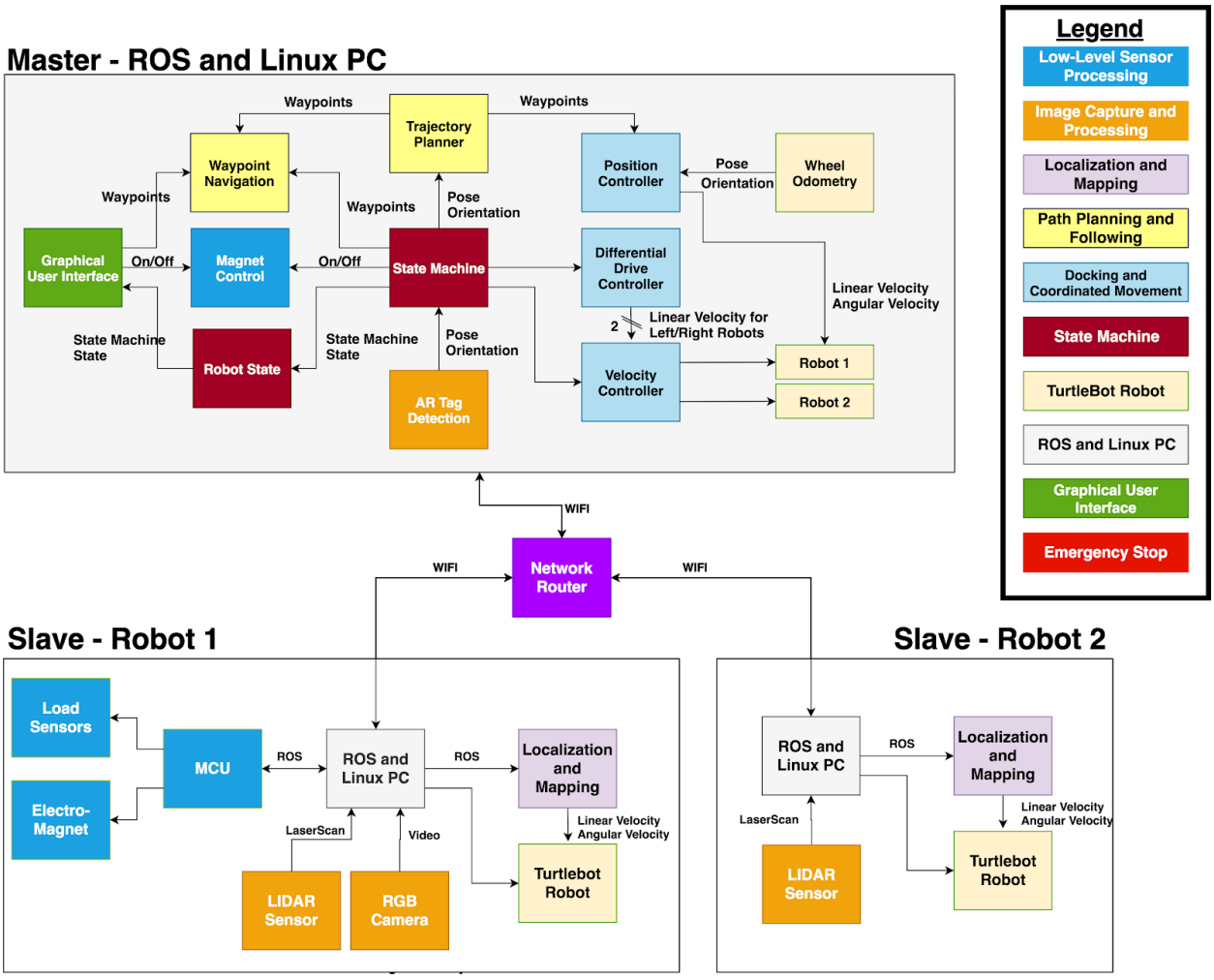


Figure 24: Software Architecture Block Diagram

5.3.1 Master - ROS and Linux PC

5.3.1.1 ROS Packages

The following ROS packages are used for the Master: SMACH, roslibjs, ROS2DJS, NAV2DJS

5.3.1.2 State Machine

Gustavo implemented the state machine using the SMACH ROS Package.

5.3.1.3 Path Planning and Following

Gustavo and Vincent are responsible for implementing the trajectory planner and waypoint navigation modules.

5.3.1.4 Docking and Coordinated Movement

Gustavo is responsible for implementing the position, differential drive, and velocity controllers that assist in docking and moving the two robots together.

5.3.1.5 Graphical User Interface

Kyle is responsible for designing and implementing the graphical user interface.

5.3.2 Slave - Robot 1

ROS Packages

The following ROS packages are used for the Master: SMACH, roslibjs, ROS2DJS, NAV2DJS, AR Track Alvar, Navigation Stack, Gmapping, Mapmerge, RRT Exploration

Navigation Stack

See the Navigation Stack Section 5.3.4 for Vincent's implementation.

MCU - Load Sensing

The load sensing module is implemented on the Arduino Uno. This module publishes its information to the GUI and state machine through a ROS Publisher.

MCU - Magnet Control

The electromagnet is controlled by the Arduino Uno. It can be controlled by the GUI and state machine through a ROS Subscriber.

Navigation Stack (Vincent)

Takes in information from wheel odometry and lidar data, a goal pose, and publishes velocity commands to the robot's mobile base. The navigation stack requires a planar laser mounted on the robot, and this laser is used for localization and creating a map.

LIDAR Camera Interface (Vincent)

The lidar reads raw scan results using RPLIDAR's SDK and converts to ROS LaserScan message. Lidar is used to perform SLAM (Simultaneous Localization and Mapping).

Turtlebot API (Vincent)

The turtlebot package provides all basic drivers to use a turtlebot/kobuki bot inside a ROS node.

5.3.3 Slave - Robot 2

Refer to Slave - Robot 1 implementation.

5.3.4 Navigation Stack (Vincent)

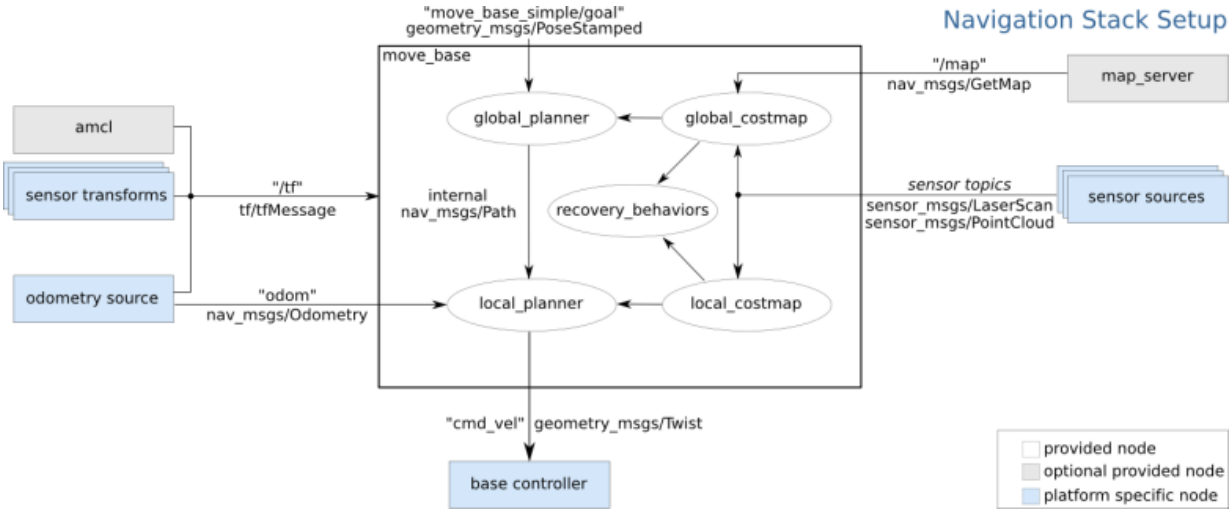


Figure 25: Navigation Stack Setup Diagram

http://wiki.ros.org/navigation/Tutorials/RobotSetup

5.3.4.1 Transform Configuration

The navigation stack requires the robot to publish information about the relationship between co-ordinate frames. In order to have proper communication in time the robot system must contain two things. The robot system must contain a Unified Robot Description Format (URDF), which is an XML format for representing a robot model. Secondly, the robot system must contain something to keep track of the changes of multiple coordinate frames over time, which in ROS can be done by using the tf package. The tf tree allows the user to keep track of multiple coordinate frames of joints and links over time. It allows us to view the relationship between those joints and links that were specified in the URDF, and also of different topics such as the map frame.

5.3.4.2 Odometry Information

As mentioned in Transform Configuration, the navigation stack requires odometry information to be published to tf with nav/msgs/Odometry message type. I used the robot's wheel odometry from its motor encoders to keep track of the robot's coordinate frame over time.

5.3.4.3 Base Controller

The navigation stack sends velocity commands using geometry_msgs/Twist message type to the base coordinate frame of the robot under the "cmd_vel" topic. We have a node that subscribe to this topic and takes velocities and converts them into commands to send to the mobile base.

5.3.4.4 Mapping

Each robot has its own generated map in order to store information such as its base coordinate frame relative to its local map. Its position and pose are tracked over time thanks to the lidar and wheel odometry, and is used to help generate the map.

5.3.4.5 Costmap Configuration

The navigation stack uses a local costmap for the robot's local planning. It uses these cost maps to store information about obstacles in its world. Here I mapped the lidar sensor topics to the costmap in order for the navigation stack to listen for updates. From here on out I configured multiple parameters. The obstacle range was set to 2.5 meters which determines the max distance the robot will put an obstacle into the costmap. Raytrace range was set to 3.0, which means the robot is allowed to clear obstacles up to 3 meters away. I explicitly defined the lidar to be used for its sensor as well.

5.3.4.6 Base Local Planner Configuration

The base local planner computes velocity commands to send to the mobile base of the robot. Here I set parameters based on the specs I thought seemed best for it to navigate. Its max velocity is 3 m/s with an acceleration limit of 2 m/s^2, minimum velocity is .1 m/s, its rotational velocity is 1 m/s with an acceleration limit of 3 m/s^2. I found these values to be the smoothest movement possible for the robots.

5.3.4.7 Launch File

Each of these configurations mentioned above contains its own separate launch file. Thanks to our system of scalable robots, I had a single launch file that contained all of the configuration launch files mentioned above along with separate namespaces and argument tags. This allows us to launch an N number of navigation stacks for an N number of robots, rather than having to make configuration files and tf trees

for each and every unique robot. I named this launch file “all.launch” which launches everything mentioned in the navigation stack.

5.3.4.8 Running Navigation Stack

With the simplified launch file, all it takes is one command and an argument of what number robot I want to start.

Example - Linux Terminal: roslaunch all.launch scot_name:=robot_1

5.3.4.9 Navigation Stack Tuning

Utilizing the navigation stack requires fine tuning to your specific robots. There are goal tolerances for the x and y position and also its orientation. The speed at which it travels is very important as well. For more information please look into Section 11.

5.4 Rationale and Alternatives

This architecture was chosen to follow the design patterns of other robotics projects in academia. We wanted to utilize the Robot Operating System since it provides a flexible framework with a large community.

6 Data Structures

The following data structures are utilized in the project.

6.1 Internal software data structure

6.1.1 ROS message to GUI

For passing information from the robot to display on the GUI, we created a custom ROS message. This message specifies the ***structure of the data which is sent from ROS to the GUI. The information which are stored in the message are the following: x and y positions for robots one and two, position of the AR tag if seen by the camera, battery level of the robot, load measured by robot 1's load sensors, and current state of the state machine.***

6.1.1.1 ROS Topics

The following ROS topics are utilized to communication between different state in the state machine.

Table 3: List of ROS topics for GUI.

Topic	Message Type	Purpose
/sm_state	std_msgs/String	Receive current state of robot
/robot_1/move_base	MoveBaseAction	Action Lib client for moving TurtleBot
/robot_1/mobile_base/ sensors/core	kobuki_msgs/SensorState	Receive battery info
/robot_2/ar_tag_pos	std_msgs/Float32	Receive x and y coordinates of AR tag
/robot_1/odom	nav_msgs/Odometry	Receive x and y coordinates of robot 1
/robot_2/odom	nav_msgs/Odometry	Receive x and y coordinates of robot 2
/robot_load	std_msgs/Float32	Receive load of robot 1
/robot_1/map	nav_msgs/OccupancyGrid	Receive occupancy grid map of robot 1
/robot_1/camera/ image_raw/compressed	sensor_msgs/CompressedImage	Receive feed of camera of robot 1
/emergency_stop	std_msgs/Bool	Send stop signal to TurtleBot
/toggle_magnet	std_msgs/Bool	Send on or off signal to electro-magnet

6.2 Global data structure

6.2.1 Transform (Tf) Tree

The Tf tree keeps track of all of the joint locations utilized in the project.

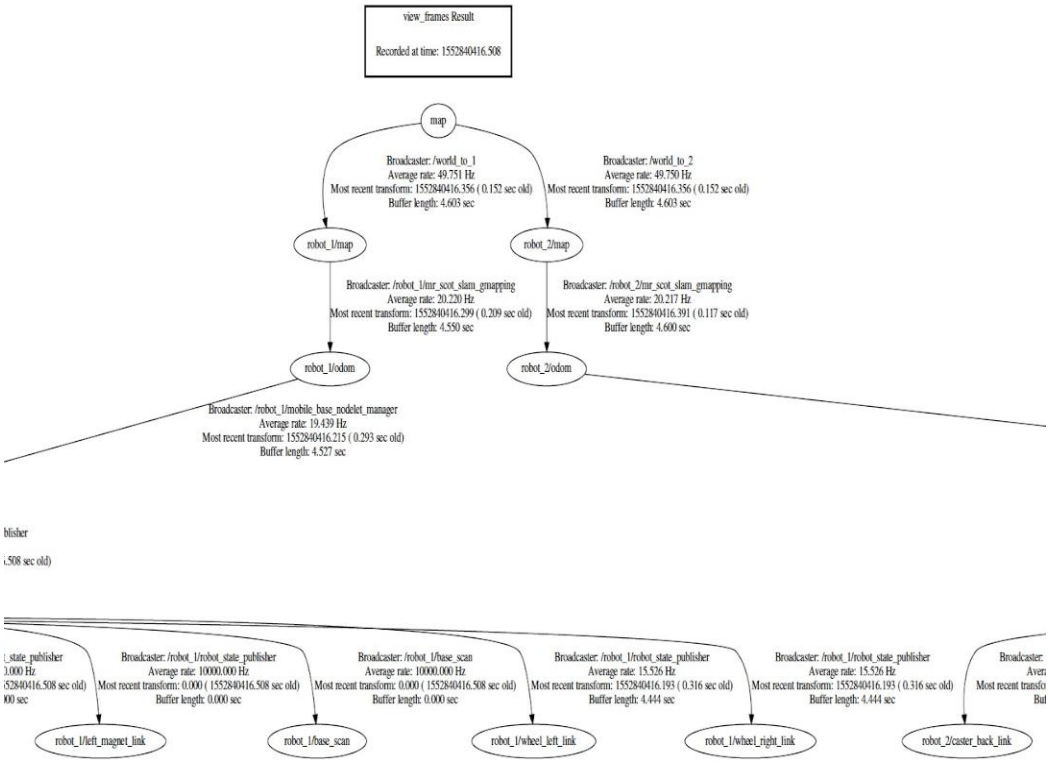


Figure 26: tf tree

6.3 Temporary data structure

No temporary data structures were used in this application.

6.4 Database descriptions

No databases were used in this application.

7 Low Level Design

MR.SCOT's low level design implementation is illustrated in the following schematic. The Intel NUC serves as the Linux PC which connects and communicates to the robot, Arduino MCU, latching mechanism, and sensors.

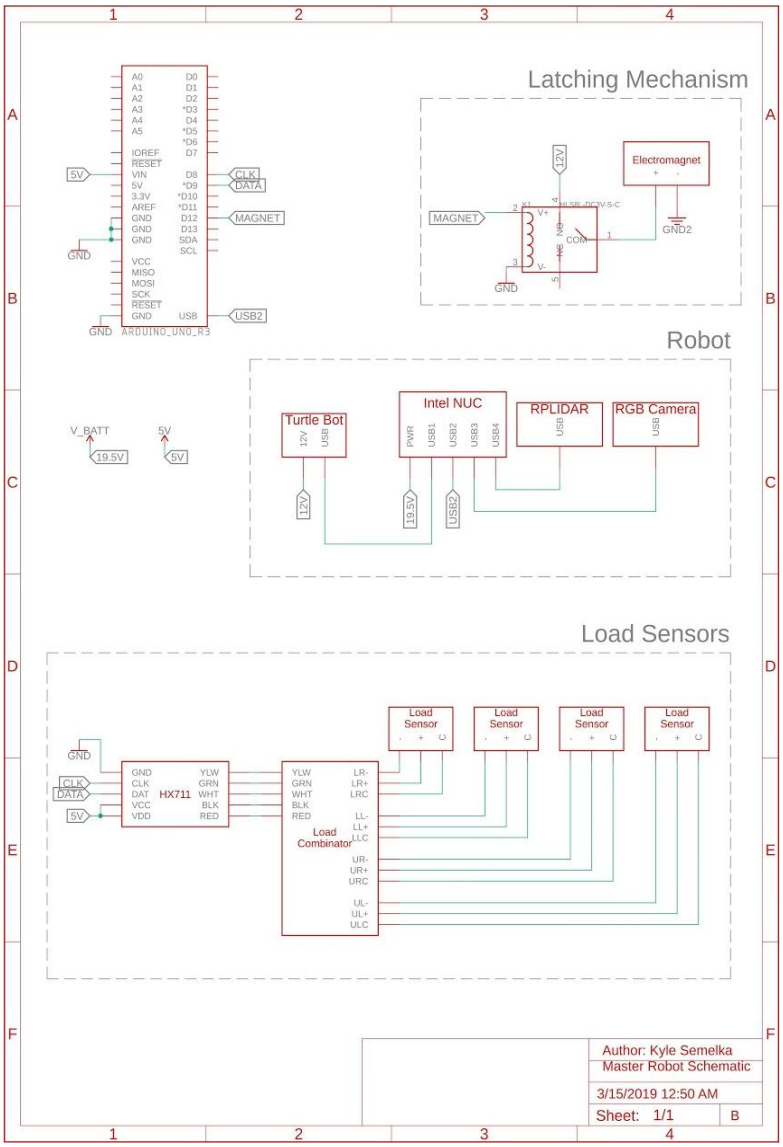


Figure 27: Electrical Schematic

7.1 Mechanical Design

7.1.1 Robot Chassis

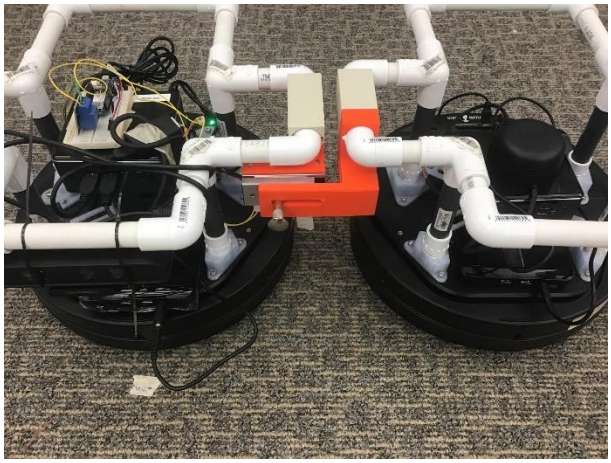


Figure 28: Robot chassis.

7.1.2 Chassis Mounts

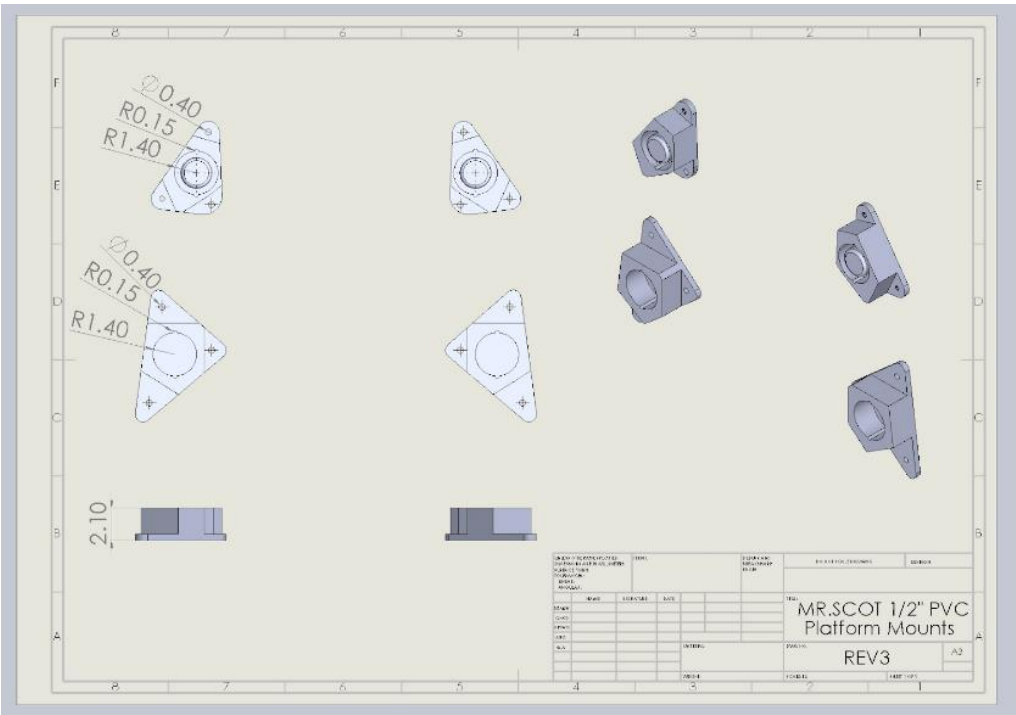


Figure 29: Chassis mounts mechanical drawing.

Figure 30: Mounts connecting robot chassis to TurtleBot Robot.

7.1.3 LIDAR Sensor Mounts

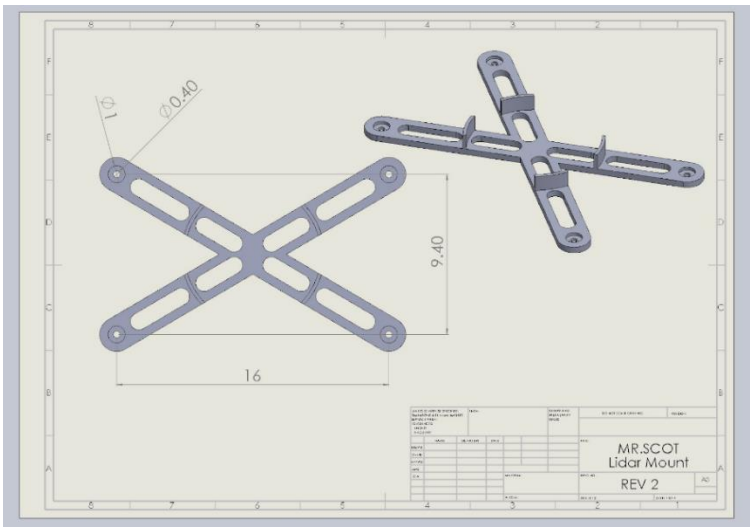


Figure 31: Mechanical design for LIDAR sensor mount.

Figure 32: LIDAR sensor mount attached to TurtleBot.

7.1.4 Magnet Mounts

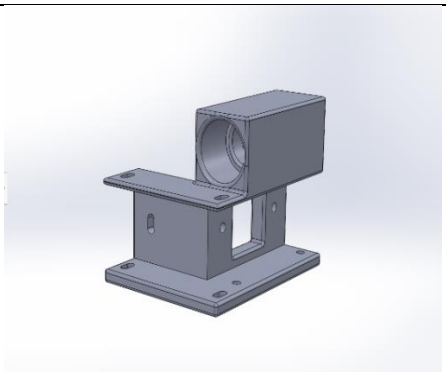


Figure 33: Mount for electromagnet.

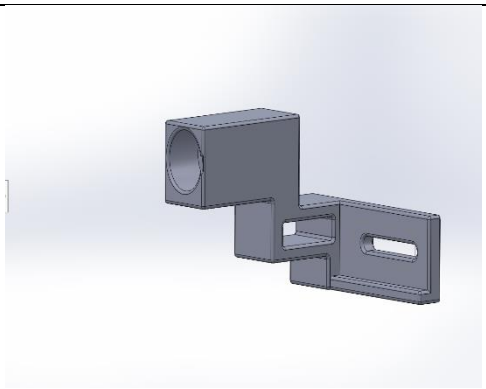


Figure 34: Mount for metal bar.

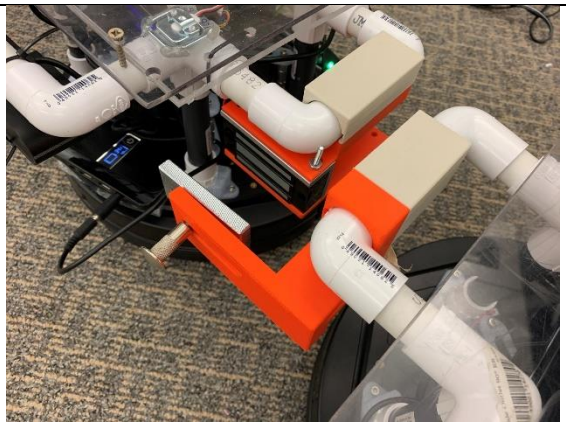


Figure 35: 3D printed mounts.

7.2 Load Cells

The load cells are used to measure the weight on top of the robot. We use four load cells: one for each corner of the square platform. Each load cell is equidistant from one another on the platform for uniform calculation of load, no matter where the load is placed on the platform. This module is so that we know if an object placed on the robot is too heavy.

Kyle is responsible for this module.

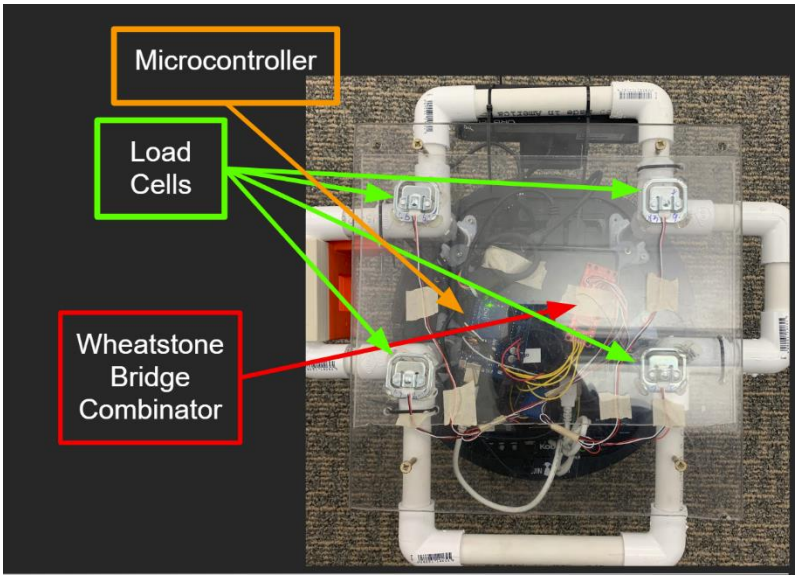


Figure 36: Load cell configuration.

7.2.1 Processing narrative for Load Sensors

The load sensors work using a strain gauge mechanism. The load cells are combined using a Wheatstone Bridge Combinator, then connected to the HX711 Load Cell Amplifier so that the Arduino Uno can read the value given by the load cells.

7.2.2 Load Sensors interface description

State Inputs

Subscriber topics:

a. N/A

State Outputs

Publisher topics:

1. ***/robot_load***
    a. Data type: Float32
    b. Description: Stores the current load on the robot.
    c. Users:
        i. State machine
        ii. Graphical User Interface

7.2.3 Load Sensors processing details

The HX711 Load Cell Amplifier which we use has a default rate of 10 samples per second (SPS). There is a selectable option for 80SPS, however, it's unnecessary for our application. The HX711 has a

precision of 24-bit. The load cells themselves are rated at 50 kg which is much higher than the load which our robot can support.

On the Arduino, readings from the HX711 were taken continuously at a rate of 10Hz. Readings are stored in a global ring buffer of floats called “measurements”. Multiple sizes of ring buffer were tested, with a size of 5 being optimal for our application. The readings are reduced by a factor of 100 to prevent overflow, and a tare value is subtracted by it. This tare is a created during the startup sequence after a settling time of 400ms defined in the HX711 datasheet and after the ring buffer is filled for the first time. To control the output rate of the module, we keep track of how many times the ring buffer is filled. After every n times, the module will publish the average value of the last 50 readings to our ROS topic, “/robot_load”. In our application, we published every 2 runs, or 1 second.

```
# Get tare value
repeat 50 times:
  get hx711 reading
  add reading to ring_buffer
tare_value := average(ring_buffer)

repeat forever:
  # Get load reading and scale it down
  get hx711 reading
  reading /= 100
  reading -= tare_value
  add reading to ring_buffer
  if iteration >= iterations_per_run:
    # Publish to ros_topic “/robot_load”
    publish(average(ring_buffer))
    iteration := 0
    continue
  iteration++
```

Figure 37: Load Sensor Processing Pseudocode

7.3 Electromagnet

The Electromagnet module consists of a 12V electromagnet, a 5V relay, an Arduino Uno, and the 12V power supply of the TurtleBot. This electromagnet is attached to the mechanical frame of our robot in order to create a joint between multiple robots when needed.

Kyle is responsible for this module.

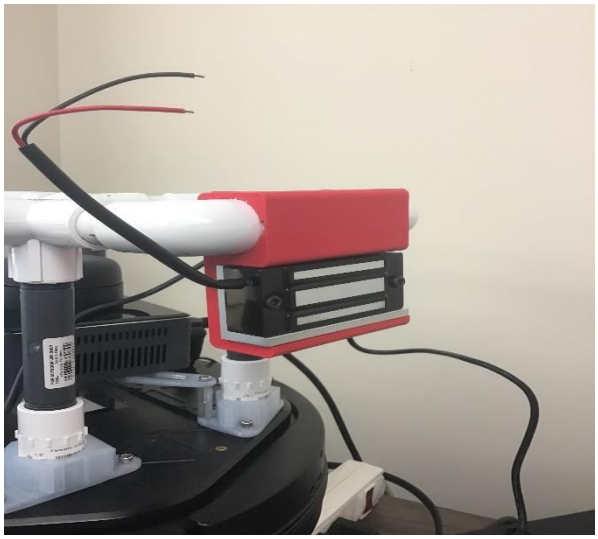


Figure 38: Electromagnet.

7.3.1 Processing narrative for Electromagnet

The electromagnet is controlled by a relay connected to the 12V power supply of the TurtleBot. The relay is controlled by an Arduino which turns the magnet on or off depending on the message which is published to the "/toggle_magnet" topic on ROS.

7.3.2 Electromagnet interface description

State Inputs

Subscriber topics:

1. ***/toggle_magnet***
    a. Data type: Bool
    b. Description: On or off signal for electromagnet
    a. Users:
        i. State machine
        ii. Graphical User Interface

State Outputs

Publisher topics:

1. N/A

7.3.3 Electromagnet processing details

The module uses a ROS Subscriber listening to the "/toggle_magnet" topic. ROS Arduino. This subscriber then calls a callback function which pulls a digital I/O pin on the Arduino to high or low depending on the ROS message passed.

7.4 State Machine

7.4.1 Processing narrative for State Machine

The State Machine was created using the SMACH ROS package which provides a framework for state machine programming.

Gustavo and Vincent are responsible for the state machine.

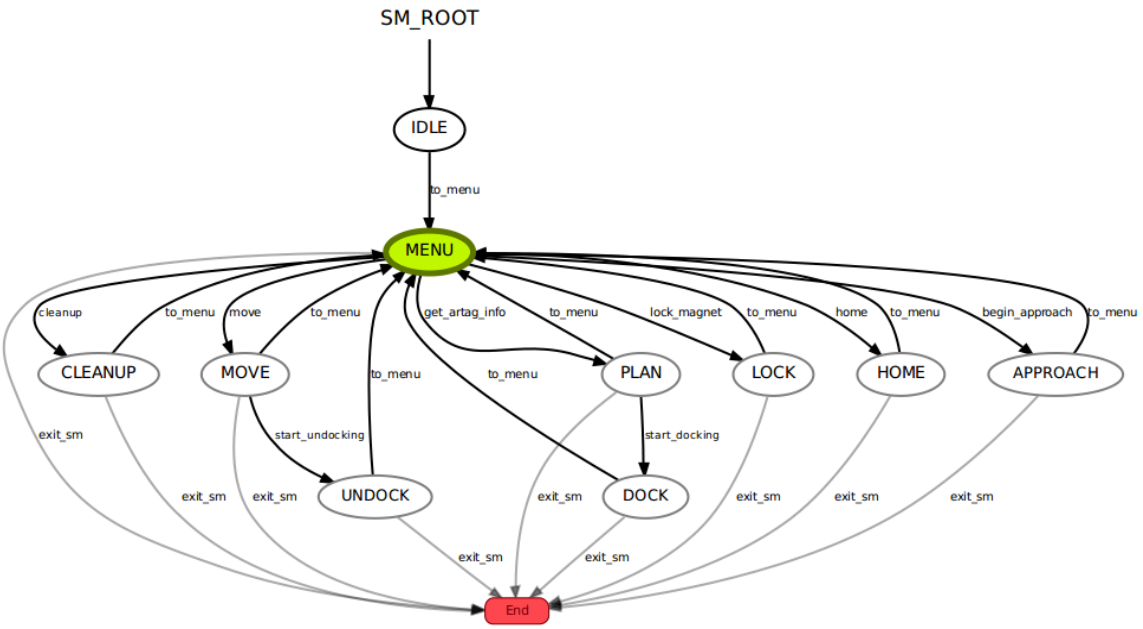


Figure 39: SMACH State Machine

7.4.2 State Machine interface description

Inputs

The main inputs to the state machine include data from sensors on the robot. Please look at the following sections to view the inputs of each state in the state machine.

Outputs

The main output from the state machine includes velocity commands for the robots, commands to engage the electromagnet, and output for data visualization.

7.4.3 State Machine processing details

The following is a template for a state in the state machine. Each state consists of a Python class where subscriber and publisher functions connect to data topics being updated in ROS. The class demonstrated below is the outline of the MENU state which leads to the other states in the state machine.

```
# define state MENU
class Menu(smach.State):
   def __init__(self):
      smach.State.__init__(self, outcomes=['begin_approach','lock_magnet','get_artag_info','start_un-
dock','exit_sm','move', 'home', 'cleanup','face_control'],
                           input_keys=['status_in'],
                           output_keys=['status_out'])

      # INIT SUBSCRIBERS

      # INIT PUBLISHERS
      self.sm_state_publisher = rospy.Publisher('/sm_state', String, queue_size = 5)


   def execute(self, userdata):

      self.sm_state_publisher.publish('MENU')
      rospy.loginfo('MENU STATE: In Menu')

      user_input = raw_input("Choose Option: \n (1) Demo Mode (2) Demo Mode with User Input \n (3)
Approach (4) Magnet Control (5) Output AR Tag (6)Move (7) Home (8) Cleanup (9) Face Control (10)Un-
dock (11) to exit state machine: ")

      if   user_input == '1':
                  return 'begin_approach'
      elif user_input == '2':
                  return 'begin_approach'
      elif user_input == '3':
                  return 'begin_approach'
      elif user_input == '4':
                  return 'lock_magnet'
```

```
    elif user_input == '5':
                return 'get_artag_info'
    elif user_input == "6":
        return "move"
    elif user_input == "7":
        return "home"
    elif user_input == "8":
        return "cleanup"
    elif user_input == "9":
        return "face_control"
    elif user_input == "10":
        return "start_undock"
    else:
        return 'exit_sm'
```

7.5 State Machine: *Menu State*

7.5.1 Processing narrative for *Menu State*

Gustavo and Vincent were responsible for this state.

The Menu State in the state machine contains the terminal style menu that accepts user input and takes the user to the corresponding state in the state machine. No computation or data processing occurs in this state.

7.5.2 Menu State interface description

State Inputs

Subscriber topics:

1. N/A

Data from previous state: N/A

State Outputs

Publisher topics:

1. **/sm_state**
    a. Data type: String
    b. Description: Stores the current state of the state machine

Data to next state: N/A

Next possible states: Approach, Lock, Plan, Move, Undock, Clean Up, Exit

7.5.3 Move State processing details

Pseudocode:

1. user_input = input "Choose Option:
    (1) Demo Mode (2) Demo Mode with User Input
    (3) Approach (4) Magnet Control (5) Output AR Tag
    (6)Move (7) Home (8) Cleanup (9) Face Control (10) Exit"
2. Go to corresponding state according to user_input

7.6 State Machine: *Approach State*

7.6.1 Processing narrative for *Approach State*

The *Approach State* in the state machine is utilized to send waypoints to robot_1 using the Navigation Stack and MoveBaseGoal() messages.

7.6.2 Approach State interface description

State Inputs

Subscriber topics:

1. N/A

Data from previous state: N/A

State Outputs

Publisher topics:

1. ***/robot_1/move_base***
    a. Data type: MoveBaseAction
    b. Description: Stores the waypoint being sent to robot_1
2. ***/sm_state***
    a. Data type: String
    b. Description: Stores the current state of the state machine

Data to next state: N/A

Next possible states: Menu, Exit

7.6.3 Approach State processing details

Pseudocode:

user_input = input "Insert x and y coordinates: "

send robot_1 to the user_input x/y coordinates with navigation stack

7.7 State Machine: *Plan State*

7.7.1 Processing narrative for *Plan State*

The Plan State in the state machine obtains the position of robot_2 relative to robot_1 using an AR Tag Detection and Tracking ROS package. The raw output of the AR Tag ROS package is then sent over to the Dock State.

Gustavo was responsible for this state.

Figure 40: Robot 1 detecting AR tag on Robot 2.

7.7.2 Plan State interface description

State Inputs

Subscriber topics:

1. **/robot_1/ar_pose_marker**
    a. Data type: AlvarMarkers
    b. Description: Stores the pose and orientation of AR tags seen on the screen relative to the camera.

Data from previous state: N/A

State Outputs

Publisher topics:

1. **/robot_2/ar_tag_x_pos**
    a. Data type: Float32
    b. Description: Stores the x value of the AR Tag pose.
    a. ***/robot_2/ar_tag_y_pos***
    c. Data type: Float32
    d. Description: Stores the y value of the AR Tag pose.
    b. ***/sm_state***
    e. Data type: String
    f. Description: Stores the current state of the state machine.

Data to next state: The pose and orientation of the AR tag.

Next possible states: Undock, Menu, Exit

7.7.3 Move State processing details

N/A

7.8 State Machine: *Dock State*

7.8.1 Processing narrative for *Dock State*

The Dock State in the state machine contains the procedure to dock robot_1 to robot_2. This state receives the AR Tag location from the Plan state and sends two waypoints to robot_1 to dock to robot_2. A PID controller is used for waypoint navigation.

Gustavo and Vincent were responsible for this state.

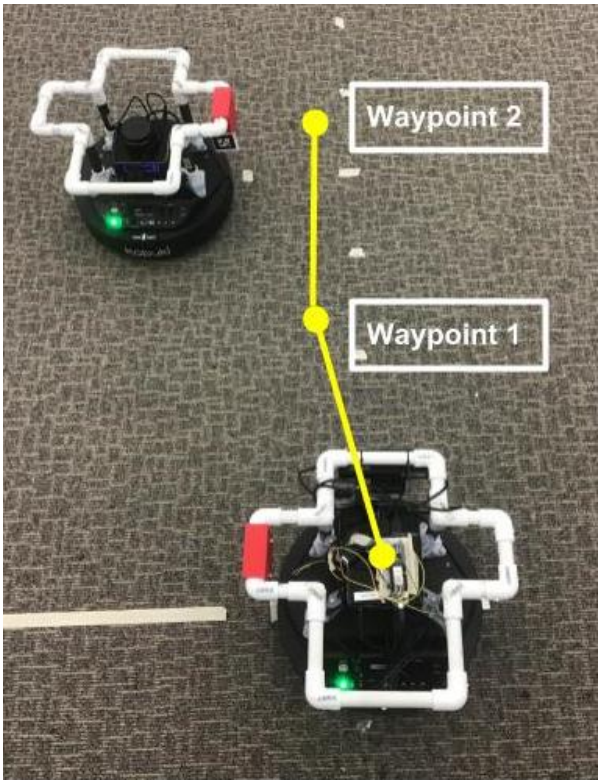


Figure 41: Illustration of waypoints.

7.8.2 Dock State interface description

State Inputs

Subscriber topics:

1. ***/robot_1/odom***
    a. Data type: Twist
    b. Description: Stores the odometry information for robot_1 relative to global map.

Data from previous state: AR Tag pose and orientation relative to robot_1

State Outputs

Publisher topics:

1. ***/toggle_relay***
    a. Data type: Bool
    b. Description: Stores the on or off state of the electromagnet
    a. ***/robot_1/mobile_base/commands/velocity***
    c. Data type: Twist

d. Description: Stores the linear and angular velocity sent over to robot_1

b. ***/sm_state***

e. Data type: String

f. Description: Stores the current state of the state machine

Data to next state: N/A

Next possible states: Menu, Exit

7.8.3 Dock State processing details

Pseudocode:

Turn off electromagnet

Xar = ar tag x position

Yar = ar tag y position

Xna = 0.16 // x distance from AR tag to bottom right corner of robot_2

Xrc = 0.20 // x distance from camera_link to center of robot_1

Xad = 0.01 // x distance from AR tag to docking point on robot_2

Xoffset = 0.03 // x offset determined from testing

Ymr = 0.20 // y distance from magnet on robot_1 to center of robot_1

Yad = 0.01 // y distance from AR tag to docking point on robot_2

Yoffset = 0.05 // y distance from robot_2 metal to robot_1 lock_magnet

First waypoint to align robot_1 to robot_2 before docking

W1x = Xar - Xrc - Xna

W1y = Yar - Ymr - Yoffset

Second waypoint to align robot_1 to robot_2 docking point

W2x = Xar + Xad + Xrc - Xoffset

W2y = Yar - Ymr - Yoffset

Turn off electromagnet

move_to_point(W1x,W1y) // Position robot_1 behind robot_2

move_to_point(W2x,W2y) // Dock robot_1 to robot_2

Function move_to_point()

adjust orientation to face waypoint

move to waypoint

Performance Issues:

While using the move_to_point() function, we run into issues where the PID controller takes substantial time to reach a steady state. This is due to the docked robot cause an opposing force and preventing robot_1 from reaching the desired waypoint.

7.9 State Machine: *Move State*

7.9.1 Processing narrative for *Move State*

The Move State in the state machine contains a differential drive controller that models the multi-robot system as one unit. A master linear and angular velocity is mapped to linear velocities controlling both robots in the system. A figure-8 motion or teleoperated movement can be demonstrated in this state. Finally, a publisher publishes the linear velocities to the corresponding robots.

Gustavo and Vincent were responsible for this state.

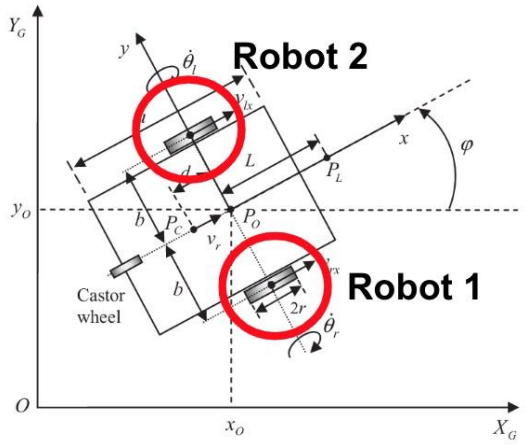


Figure 42: Differential drive model of MR.SCOT

7.9.2 Move State interface description

State Inputs

Subscriber topics:

1. ***/multi/mobile_base/commands/velocity***
    a. Data type: Twist

b. Description: Takes the linear and angular velocity input from a teleoperation controller

a. ***/multi/mobile_base/commands/motor_power***

c. Data type: MotorPower

d. Description: Enables/Disables the motor for both robots

Data from previous state: N/A

State Outputs

Publisher topics:

1. ***/robot_1/mobile_base/commands/velocity***
    a. Data type: Twist
    b. Description: Stores the linear and angular velocity sent over to robot_1
    a. ***/robot_2/mobile_base/commands/velocity***
    c. Data type: Twist
    d. Description: Stores the linear and angular velocity sent over to robot_2
    b. ***/sm_state***
    e. Data type: String
    f. Description: Stores the current state of the state machine

Data to next state: N/A

Next possible states: Undock, Menu, Exit

7.9.3 Move State processing details

Pseudocode:

```
user_input = input "Press (1) Move robots (2) Teloperation (3) Menu (4) Exit:  "
if (user_input == “1”)
diff_drive(‘Figure-8’) //Robots move as a single unit on a planned path
elif (user_input == “2”)
        diff_drive(‘Teleoperation’) //Manual control of the robots
elif (user_input == “3”)
Return to menu
elif (user_input == “4”)
Exit state machine


function diff_drive()
```

```
center_distance = 0.432 //distance between center of mass of both robots
        V_xy = keyop_velocity.linear.x
        omega = keyop_velocity.angular.z
          if omega != 0:
             icc_radius = V_xy / omega # Obains the ICC radius
             V_l = omega * ( icc_radius - center_distance / 2) # Robot_2 tangential velocity
             V_r = omega * ( icc_radius + center_distance / 2) # Robot_1 tangential velocity
          else:
             V_l = V_xy # Robot_2 tangential velocity
             V_r = V_xy # Robot_1 tangential velocity

          robot_1_vel.linear.x = V_r
          robot_2_vel.linear.x = V_l

          robot_1_vel.angular.z = 0
          robot_2_vel.angular.z = 0

          self.robot_1_velocity.publish(robot_1_vel.linear,robot_1_vel.angular)
          self.robot_2_velocity.publish(robot_2_vel.linear,robot_2_vel.angular)
```

Performance Issues:

With this differential drive controller, we face an issue where the "center_distance" will change every time the robots dock together.

7.10 State Machine: *Undock State*

7.10.1 Processing narrative for *Undock State*

The Undock State in the state machine frees the two robots by unlocking the electromagnet connection. The two robots move 1 meter in opposite x directions to show that both robots now move independently of one another.

Gustavo and Vincent were responsible for this state.

Figure 43: Robots undocking during demo.

7.10.2 Undock State interface description

State Inputs

Subscriber topics: N/A

Data from previous state: N/A

State Outputs

Publisher topics:

1. ***/robot_1/mobile_base/commands/velocity***
    a. Data type: Twist
    b. Description: Stores the linear and angular velocity sent over to robot_1
    a. ***/robot_2/mobile_base/commands/velocity***
    c. Data type: Twist
    d. Description: Stores the linear and angular velocity sent over to robot_2
    b. ***/sm_state***
    e. Data type: String
    f. Description: Stores the current state of the state machine

Data to next state: N/A

Next possible states: Go home, Menu, Exit

7.10.3 Undock State processing details

Pseudocode:

user_input = input "Press (1) Go Home (2) Menu (3) Exit: "

if (user_input == “1”)

unDock()

Return to Home State
elif (user_input == “2”)
Return to menu
elif (user_input == “3”)
Exit state machine

function unDock()

start_time = now //Set current time as start time
publish_time = 2 //Set amount of time to publish a velocity

electromagnet_state = 1 // Publish 1 to turn magnet off
arduino_publisher.publish(self.electromagnet_state) // Sends electromagnet command to Arduino

while (now - start_time) < publish_time: // While current time is less than publish time
rospy.loginfo(now - start_time)
robot_1_vel = Twist()
robot_2_vel = Twist()

robot_1_vel.linear.x = -0.2 // move robot_1 at -2m/s
robot_2_vel.linear.x = 0.2 // move robot_2 at -2m/s

robot_1_vel.angular.z = 0
robot_2_vel.angular.z = 0

self.robot_1_velocity.publish(robot_1_vel.linear,robot_1_vel.angular)
self.robot_2_velocity.publish(robot_2_vel.linear,robot_2_vel.angular)

7.11 State Machine: *Home State*

7.11.1 Processing narrative for *Home State*

The Home State in the state machine sends both robots back to their original position by using the navigation stack. The navigation stack sends velocity commands to each robot.

Gustavo and Vincent were responsible for this state.

7.11.2 Home State interface description

State Inputs

Subscriber topics:

1. N/A

Data from previous state: N/A

State Outputs

Publisher topics:

1. ***/robot_1/move_base***
    a. Data type: MoveBaseAction
    b. Description: Stores the waypoint being sent to robot_1
2. ***/robot_2/move_base***
    a. Data type: MoveBaseAction
    b. Description: Stores the waypoint being sent to robot_2
3. ***/sm_state***
    a. Data type: String
    b. Description: Stores the current state of the state machine

Data to next state: N/A

Next possible states: Menu, Exit

7.11.3 Home State processing details

Pseudocode:

user_input = input "Insert x and y coordinates: "

send robot_1 to its origin (0,0) with navigation stack

send robot_2 to its origin (0,0) with navigation stack

Performance Issues:

With the navigation stack we face accuracy problems with sending the robots back to its place of origin. When the robots are not within the threshold of its goal, it may spin in place trying to meet its goal.

7.12 State Machine: *Cleanup State*

7.12.1 Processing narrative for *Cleanup State*

The Cleanup State in the state machine reads each individual robot's odometry in quaternion and converts the data to euler form. It sends an angular velocity command to each robot to reorient the robots to its initial orientation. The electromagnet is turned off as well.

Gustavo and Vincent were responsible for this state.

7.12.2 Cleanup State interface description

State Inputs

Subscriber topics:

1. ***/robot_1/odom***
    a. Data type: Odometry
    b. Description: Odometry
2. ***/robot_2/odom***
    a. Data type: Odometry
    b. Description: Odometry

Data from previous state: N/A

State Outputs

Publisher topics:

1. ***/robot_1/mobile_base/commands/velocity***
    a. Data type: Twist
    b. Description: Stores the linear and angular velocity sent over to robot_1.
2. ***/robot_2/mobile_base/commands/velocity***
    a. Data type: Twist
    b. Description: Stores the linear and angular velocity sent over to robot_2.
3. ***/sm_state***
    a. Data type: String
    b. Description: Stores the current state of the state machine

Data to next state: N/A

Next possible states: Menu, Exit

7.12.3 Cleanup State processing details

Pseudocode:

```
electromagnet_state = 1 # Publish 1 to turn magnet off

robot_1_yaw = euler_from_quaternion(robot_1_odom) // converts robot 1's odometry
robot_2_yaw = euler_from_quaternion(robot_2_odom) // converts robot 2's odometry

while(robot_1_yaw != original_orientation) // corrects robot 1's orientation to its initial pose
    robot_1_velocity.publish(robot_1_vel.linear,robot_1_vel.angular)
while(robot_2_yaw != original_orientation) // corrects robot 2's orientation to its initial pose
    robot_2_velocity.publish(robot_2_vel.linear,robot_2_vel.angular)
```

Performance Issues:

Because we did not know what value to look for after we converted to euler, we created a threshold for the robot's orientation to be as close to its initial orientation.

8 Technical Problem Solving

8.1 The SLAM Problem

The problem we run into is localizing our robots within the global map. We need to capture the robot's pose within the map at every second. There must be a transformation between the coordinate frames of the map and every joint and link of the robot. Otherwise there would be no correlation between the Lidar data and robot's odometry, which results in generating a map that does not update with the environment.

The Kobuki kit with open-source software allows users to work with their predefined packages and tutorials. However, this is very insufficient, hence their launch file "minimal.launch" which is only able to communicate between nodes, such as teleoperation. At this point, there is still no communication going on between any joints or links. There is no time-tracked relationship between coordinated frames, which would allow a user to transform vectors, points, etc between any two or more possible coordinate frames on or off the robot at any point in time.

The Lidar would continue publishing new data, but with its original starting coordinate frame as the robot is moving around with it. The robot's frame would be updating in time, however the robot system views the lidar as standing still, but with updated values as it were actually moving with the robot. This is a big problem because if we wanted to properly use the data from the lidar, the calculations to translate or rotate would be completely wrong.

8.2 The SLAM Solution

In order to have proper communication in time the robot system must contain two things. The robot system must contain a Unified Robot Description Format (URDF), which is an XML format for representing a robot model. Secondly, the robot system must contain something to keep track of the changes of multiple coordinate frames over time, which in ROS can be done by using the tf package.

The URDF contains the specifications and dimensions of each and every joint and link relative to each other on the robot. The Kobuki package luckily provided users with a predefined URDF that contains the dimensions of each joint and link, and their locations. We added onto the URDF additional joints and links, for the additional tools we used. Those tools include Lidar, AR camera, and the magnet.

In order to have a proper transform between the Lidar's reference frame and the robot, I first added the lidar to the URDF. I then used the static transform publisher from the tf package to publish a coordinate transform to the tf tree between the base link of the robot to the Lidar's base.

After lidar was successfully integrated onto the robot, we used it to perform laser-based SLAM (Simultaneous Localization and Mapping) with the help of a ROS package called slam_gmapping. This package creates a 2-D grid map from laser and pose data collected by a mobile robot. Thanks to our URDF and tf tree, we met the required transforms to produce a map, which were a tf transform between the robot's base link to Lidar scan, and between the robot's base link and its wheel odometry.

However, when producing the map it was messy. The walls of the environment were drawn very thick, and the map looked out of line. It wasn't until I went and configured some parameters. I changed the max range of the lidar to 5.5 meters which helped the package not draw a wall further than the range of the lidar. I also changed the linearUpdate parameter to 2 meters because it didn't need to continuously update linearly, especially if it is looking at the same wall. However, I did change the angular update to update twice as fast as its default value in order to keep up with the robot's odometry better as it is rotating.

8.3 The GMapping Problem

We encountered a problem where ROS's Gmapping package was creating a map far too large for the size of the room that we were in. The result was a map with 95% empty space, with only 5% actually containing useful mapping information. This caused multiple problems. Firstly, since the size of the map was in the gigabytes range the GUI map element was displaying the map with too much empty space. This caused the map to be way too large of a file, which reduced the rate at which the system operated. In addition, this caused the GUI to crash, because ROS was trying to send the whole map file over the Wi-Fi network to the laptop which was running the GUI.

8.4 Solving the GMapping Problem

Since this issue was first noticed from the GUI crashing, we attempted to solve this problem on the GUI side. We found a solution online where people were converting the map file first from its native format (.pgm), to an image file like .png, then sending that over to the GUI and displaying it as a highly compressed image. However, this rabbit hole ended by me scrapping the idea, since there were only deprecated and outdated libraries which offered this functionality and would not work with the rest of our ROS packages.

We solved the issue by altering parameters in the GMapping params file. The parameters we altered were "xmin", "xmax", "ymin", and "ymax". These were set to default values of -100m, 100m, -100m, and 100m respectively. We changed the values all to 0 so that the map would start as a 0m x 0m map, then grow as needed. This resulted in reducing the size of the map dramatically, and getting rid of much of the empty space that was originally in the map file. The size of the map file went from 1.5GB down to around 250MB.

8.5 The Load Sensor Problem

After attaching our load sensors to our robot, we found the readings to be very erratic and noisy. These problems weren't observed before we attached the sensors onto the robot, so we knew the problem was from the operation of the robot and not the actual load sensors or code. We determined the problem to be our LIDAR. Since the LIDAR was rotating while the robot was on, it was causing the load sensors to move slightly. Also, when the robot moved, the load cells would experience forces in different directions and it would affect the output also.

8.6 Solving the Load Sensor Problem

First, I tried making sure the LIDAR and load sensors were all securely attached to the frame. However, they were already very securely attached. I decided that the best way to solve this issue was in the software. I created a ring buffer in the Arduino code to store the last 50 readings that came in. Then, I averaged these 50 values to get a smooth output. The result was very positive. The readings went from fluctuating +/-500g, to fluctuating +/-10g. This was more than accurate enough for our application, so I considered the problem solved.

8.7 The System Scalability Problem

Initially, we wanted to have a system of four robots therefore we knew we had to make our system scalable. The idea was that we can have multiple slave robots to all perform the same action when called. Because the slaves would be mechanically identical, we wanted to be able to launch the robots in any order and number and not worry about which robot was what. We also wanted to easily be able to add an additional N number of robots as well.

8.8 Solving the System Scalability Problem

A simple yet brilliant solution I found to easily scale the system and launch an N number of robots was to include an argument tag in the URDF and every single launch file in every package we used. This allows us to launch each robot and its component under its own namespace such as "/robot1". I created a launch file that contains the command to launch several packages under the same namespace. Throughout this report, you will notice many argument tags such as "$(arg scot_name) in many launch files.

8.9 The Coordinate Transform from Lidar to Robot Problem

The lidar came with a package called “rplidar_ros”. It easily integrates the lidar into ROS and provides a launch file to get the lidar running. Once the lidar is up and running, the information coming out of it is not yet relative to the robot, regardless if placed on the robot. The lidar would continue publishing new data, but with its original starting coordinate frame as the robot is moving around with it. The robot’s frame would be updating in time as it moves, however the lidar’s frame is standing still, but with updated values as it were moving with the robot. This ultimately causes a big problem because if we wanted to properly use the data from the lidar, the calculations to translate or rotate would be completely wrong. For example, as we navigated our robot around the room, the first instance of the map was being dragged across, instead of updating the map with the new environment.

8.10 Solving Coordinate Transform from Lidar to Robot Problem

In order to have a proper transform between the lidar’s reference frame and the robot, I first added the lidar to the URDF. I then used the static transform publisher from the tf package to publish a coordinate transform to the tf tree between the base link of the robot to the lidar’s base. Now when the robot moves with lidar attached to it, the static transform publisher sends the transform between those two frames in time at 10hz. The system now knows where the lidar’s frame is relative to the robots odometry and can update the map accordingly to what it sees.

8.11 The Docking Problem

In order to dock both robots together, we had to constrain the direction of docking. Docking is still a pending problem in academia that requires further development. .

8.12 Solving the Docking Problem

Our solution for docking was to implement a local PID controller so that we could better control the waypoints that were assigned to robot_1 which docked to robot_2. Once achieving the PID controller which as developed with the mentorship of graduate students in the ARCS Lab, the second achievment was redesigning a latch mechanism so that the robots could still dock given the tolerance of the waypoint navigation.

8.13 The Coordinated Movement Problem

In order to move the multi-robot system, a different approach had to be taken to assign velocities to both robots given the constraint that they are physically attached by a joint in the center of both robots.

8.14 Solving the Coordinated Movement Problem

Our solution for moving both robots was to implement a differential drive controller which modeled both robots as one body. Once this model was created, a controller was implemented to assign a global linear velocity and map it to linear velocities for each robot.

9 User Interface Design

The user interface was made in a web page using HTML5, CSS3, and JS.

9.1 Application Control

The Javascript libraries used were roslibjs for interfacing with ROS, ros2djs for the occupancy grid client used to display the map, nav2djs to allow for navigation from the GUI, JQuery 3.3 to allow for easier HTML document manipulation, and Bootstrap 4.0 for general styling.

Thanks to Bootstrap 4.0, the design is responsive, meaning it will scale the website properly on any sized device. To run this website on our phones, we used The Apache HTTP Server ("httpd") on our main laptop and connected to the server IP using our mobile phones. This allows any user to control MR.SCOT from a mobile device given that they are on the same Wi-Fi network.

9.2 User Interface Screens

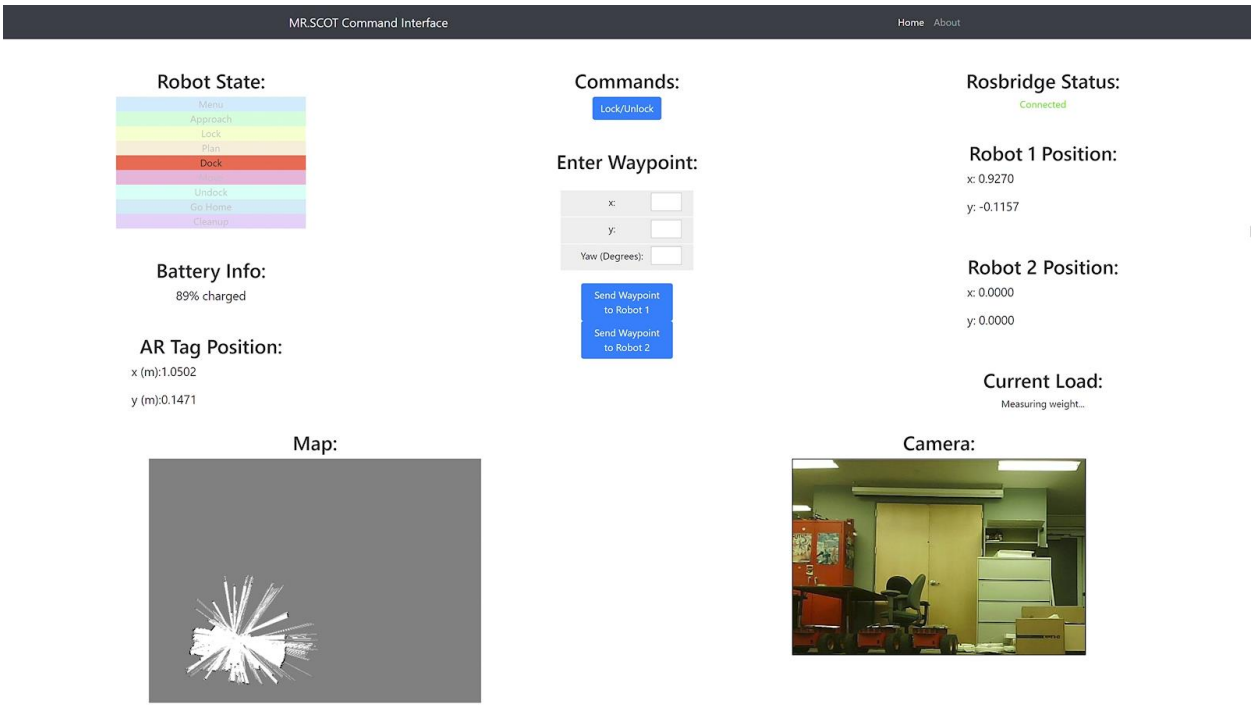


Figure 44: GUI during robot docking state.

In the top left, it shows the Robot State. As the robot goes through each state, the corresponding state box will turn active to show which state the robot is currently in.

The top right shows a status indicator for Rosbridge, the connection between the website host and the robots. This will change color to red and show an error message if there isn't a connection between the GUI and ROS. It will also show green if there is an established connection.

At the bottom left of the screen is the map. This map is updated continuously by the robot as it does SLAM.

The bottom right of the screen is for the camera. This is a live camera feed from robot 1. It updates at a rate of 8 fps and uses jpg compression.

On the left and right columns of the interface are sections for the current positions of the AR Tag, robot 1, and robot 2. These positions are displayed in meters and are taken from the global map used by the robots. There are also sections which display battery info of robot 1 and current load on robot 1. These sections are updated continuously.

The center column is meant for actions that the user can take. The user is able to lock and unlock the electomagnet, send a waypoint to robot 1, and send a waypoint to robot 2.

10 Test Plan

10.1 Test Design

The following tests were conducted to achieve the design objectives of the project.

Test Case 1: Waypoint Navigation with Navigation Stack, *Conducted by Gustavo and Vincent and Kyle*

1. The objective for this experiment is to test the accuracy of waypoint navigation where we send waypoint goals to the Navigation stack and the navigation stack outputs velocity commands for the robots to follow. **This experiment measures the ability for the robots to use SLAM for navigation and move to specific points on the generated map.**
2. The setup for the experiment consisted of marking points on the ground with masking tape which the robot had to go to. The robot would start at position (x=y=0) then receive commands to go to those points.
3. We collected data by both measuring the x/y position of the robot in respect to the origin point as well as reference the determined position retrieved from odometry.
4. We expect the robots to accurately arrive at the waypoints.

Test Case 2: Load Sensors, *Conducted by Kyle*

1. This experiment tests how accurate the load sensors are at measuring a known weight.
2. The experiment was setup by having the load sensors on a table which was still, stable, and flat, and putting an iPhone XS on top of the platform.
3. First, the load sensors module is tared when there is nothing on top of the platform. Then, once the output is settled at 0.0g, the iPhone XS is placed in the center of the platform and the output is recorded. The experiment is repeated for each corner of the platform.
4. The expected result is 177 grams, the weight of an iPhone XS as defined in the datasheet released by Apple.

Test Case 3: AR Tag Joint Calibration, *Conducted by Gustavo and Vincent*

1. The objective of this experiment is to fine tune the location of the AR tag join in respect to the camera on robot_1. This experiment helps to dock the robot_1 to robot_2 more accurately.
2. The robots are placed at a fixed distance to determine the offsets.
3. Data will be collected by carefully measuring the joint distances as shown below in Figure 24.

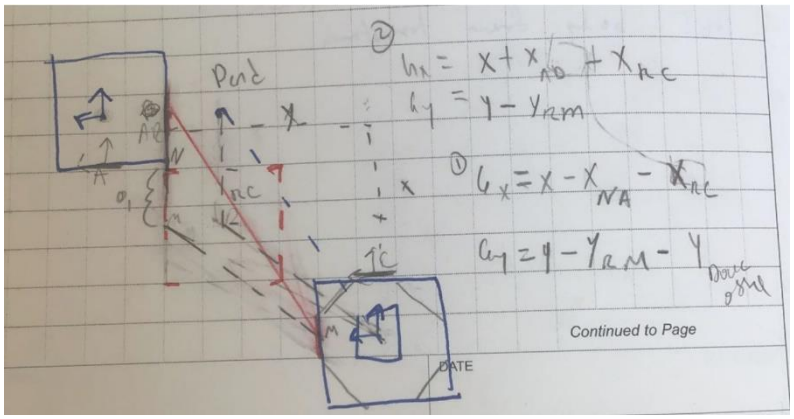


Figure 45: Offset Calculation

4. We expect further tuning to be required once the robots perform the docking procedure.

Test Case 4: AR Tag Detection Range, *Conducted by Gustavo and Vincent*

1. The objective of this experiment is to determine the viewing range which the camera on robot_1 can detect the AR tag placed on robot_2. This experiment helps to measure the **docking design objective.**
2. The photos below illustrate the experimental setup where the robots were placed to determine the left, center, and right maximum detection angles.

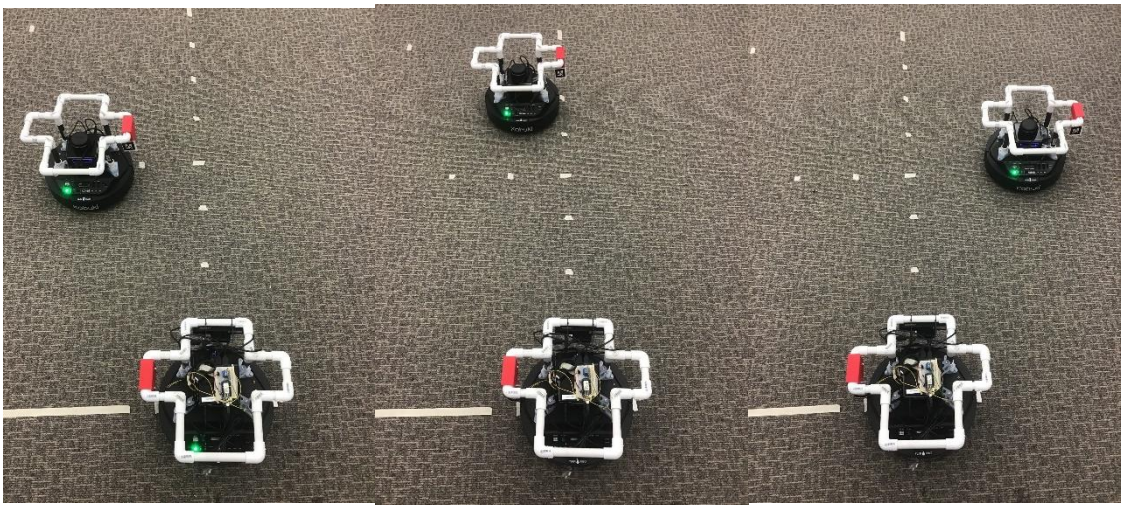


Figure 46: AR tag offset tuning.

3. For the left, center, and right general locations, the robot on top is moved by hand to the extreme points until the AR tag isn't detected. The extreme position at which the AR tag is still detected is recorded.

4. We expect the robot on the bottom (robot_1) to better detect AR tags when the robot on top (robot_2) is to the left of robot_1.

Test Case 5: Docking Procedure Tuning, *Conducted by Gustavo and Vincent*

1. The objective of this experiment is to fine tune the offsets which are used to find the location of the AR Tag together with fine tuning the PID controller used for local waypoint navigation. This experiment helps to measure the feasibility of docking robots together.

2. The experiment was setup in the default position for docking as shown in the figure below.

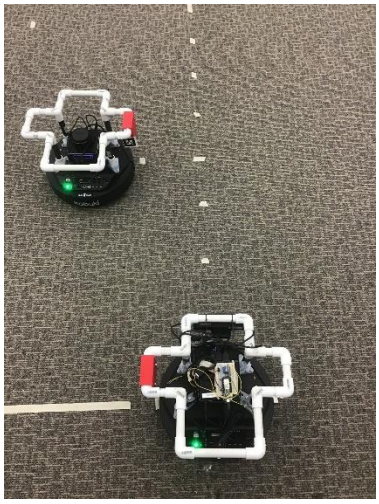


Figure 47: Docking test setup.

3. We perform the docking procedure consisting of two waypoints to practice docking the robots together. We collect data by observing the output and plotting the trajectory that the robot took.

4. We expect the PID controller to work properly after a few iterations of testing.

Test Case 6: Navigation Stack Tuning, *Conducted by Gustavo and Vincent*

5. The objective of this experiment is to fine tune the parameters of the navigation stack. The navigation stack sends way points for our robots to arrive to, and we need its path planning to reach its goal to be as accurate as possible. This experiment helps to measure the feasibility of docking robots together.

6. The experiment was setup in the default position for docking as shown in the figure below.

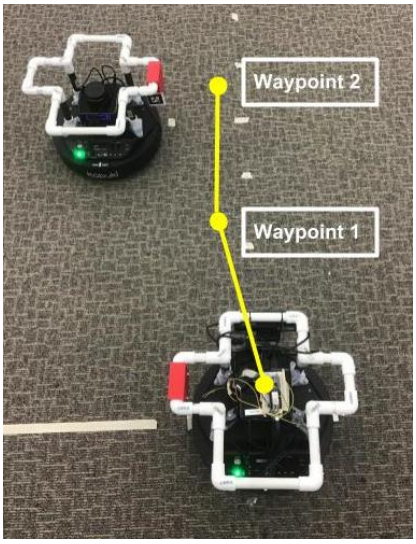


Figure 48: Navigation tack test setup.

7. We perform the docking procedure consisting of two waypoints to practice docking the robots together. We collect data by observing the output and plotting the trajectory that the robot took.

8. We expect the navigation stack to accurately send the robots to its waypoint goals.

Test Case 7: Outdoor Tests, *Conducted by Gustavo and Vincent*

5. The objective of this experiment is to test the system in an external environment.

6. The setup and procedure of this experiment was the same setup as the demo used for the experiment.

7. We collected results by observing the output of the demo and expected the demo to work the same as if it were run indoors.

10.2 Bug Tracking

Notes of bugs and issues we had were taken in our lab notebooks and were noted in the weekly progress reports.

The following are some of the issues we encountered:

Table 4: Bug Tracking Notes

Date	Notes
Week 7 11/12/2018	• Testing mapping a room using the LIDAR created issues • GMapping package in ROS requires tuning in order to map correctly
Week 8 11/19/2018	• Combined Lidar with navigation stack, but navigation was not accurate. • To fix this, we altered parameters in the navigation stack, as detailed in Section 11, Navigation Stack Tuning.
Week 14 1/28/2019	• Issue: Docking robots together is impossible without better precision • To solve this issue, we changed the electromagnet mount to not need as much precision during docking
Week 15 2/4/2019	• Issue: Coordinated movement is too jerky. • To solve this, we implement the differential drive mechanism.

10.3 Quality Control

Each test is performed many times in order to ensure a stable system. When the tests passed or failed, we noted results in our lab notebooks and reported them during the weekly progress reports.

10.4 Identification of critical components

The Intel NUC is a critical component of our robots. Since the NUC is the "brain" of the robot, many negative consequences can happen if it crashes or shuts down during operation. As a result, we always remotely operate the robots and monitor their status from the command line when testing.

10.5 Items Not Tested by the Experiments

The accuracy of the odometry was not tested. Given that the robot is also localized using LIDAR and the AR tag, we determined that the odometry was not a critical component which needed testing.

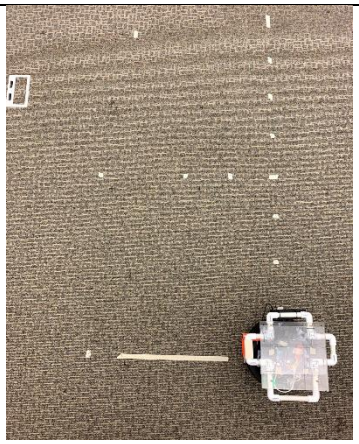
11 Test Report

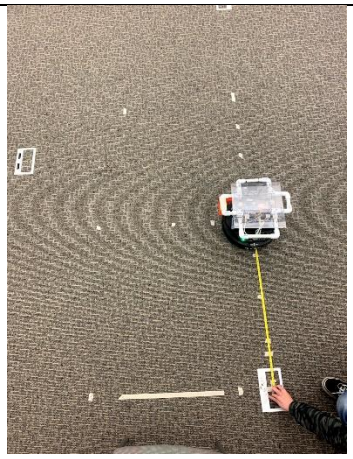
The following sections detail the results of the tests proposed in Section 10.

11.1 Waypoint Navigation

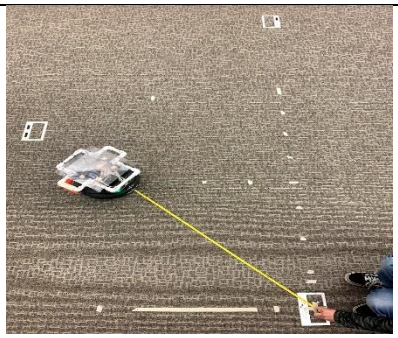
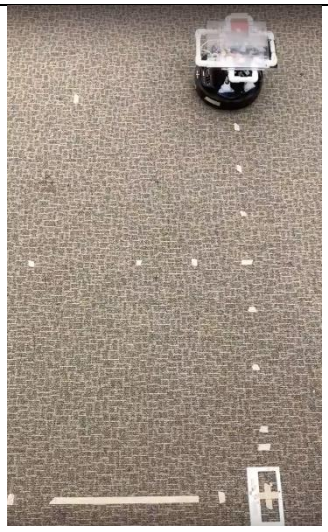
Vincent, Kyle, Gustavo performed the experiment

1. Before tuning the navigation stack, the accuracy of the robots was very off, up to 6% misalignment in both x and y positions.

Starting Point Here we start each test at this point.	 **Figure 49**
First Test As you could see, the robot does not follow a straight path and ends up slightly to the left of the point (1,0). According to the robot's odometry, it was 6% misaligned to its goal.	 **Figure 50**

Second Test After tuning the navigation test, the robot traveled to the point (1,0) very accurately, with less than 1% error.	 **Figure 51**

2. As we continued conducting tests, the goal points achieved by the robots became consistently accurate with less than 2% error. Below are some more pictures.

Point (1,1)	Point (2,0)
 **Figure 52**	 **Figure 53**

3. Analysis of test results

   After configuring the navigation stack, way point navigation became accurate enough to proceed with using it in our state machine.

4. Corrective actions taken

   To see the configured navigation stack parameters, please look at Section 11.

11.2 Load Sensors

Kyle performed these experiments.

1. Test results

Starting Point The system is tared and settled. Initial Reading: **0.0g**
First Test The IPhone XS is placed in the center of the platform. Reading: **177.2g**
Second Test The IPhone XS is placed in the bottom left corner of the platform. Reading: **177.7g**
Third Test The IPhone XS is placed in the bottom right corner of the platform. Reading: **178.1g**
Fourth Test The IPhone XS is placed in the upper left corner of the platform. Reading: **176.6g**
Fifth Test The IPhone XS is placed in the upper right corner of the platform. Reading: **178.7g**

2. Comparison With Expected Results

   After the tests, we can conclude that the load sensors module operates at within a **1% error**, no matter where the load is placed on the platform.

3. Analysis of Test Results

   We conclude that the upper right corner of the platform shows the greatest deviation from the expected result. However, this deviation is reasonable, and within the needed range for our application.

4. Corrective Actions Taken

   No corrective actions were taken, since the error was within a reasonable range.

11.3 AR Tag Joint Calibration

Tests conducted by Gustavo

See Section 11.5 for Docking procedure tuning which tunes the AR tag joint offsets.

11.4 AR Tag Detection Range

Tests conducted by Vincent and Gustavo

1. Test Results

Left side of camera		Center of camera		Right side of camera	
X (m)	Y(m)	X(m)	Y(m)	X(m)	Y(m)
1.325	0.376	1.488	0.008	1.59	-0.764
0.96	0.425	1.294	0.007	0.993	-0.486
0.605	0.270	1.05	0.088	0.909	-0.462
0.438	0.150	0.824	0.070	0.659	-0.289
0.321	0.990	0.035	0.116	0.495	-0.240
0.204	0.080	0.495	0.0035	0.226	-0.069
0.136	0.32	0.363	0.0015	0.169	-0.053

2. Comparison with expected results

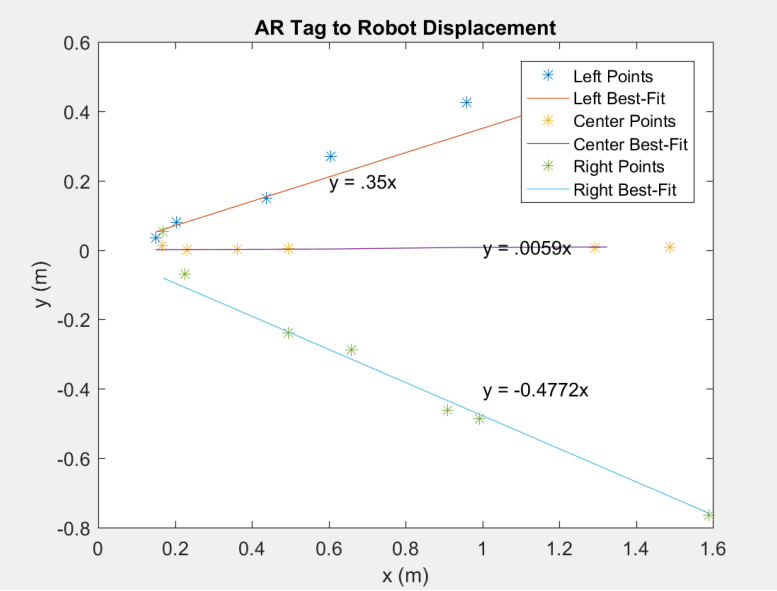


Figure 54: AR Tag to Robot Displacement

3. Analysis of Test Results

    As expected, we can't deviate too far from its angle of best fit. The RGB camera's field of view is not as wide as it seems. The AR tag can be seen in its live video capture, but if the AR tag is not in its best fit angle, then the AR tag will not be detected.

4. Corrective Actions Taken

    Instead of trying to change the camera settings, we decided to proceed with these results. To compensate, we made sure to make the robots align themselves as best as possible before trying to capture data from the AR Tag.

11.5 Docking Procedure Tuning

Gustavo and Vincent performed this experiment.

1. Test Results

    The following results demonstrate actions taken to improve the accuracy of the docking procedure.

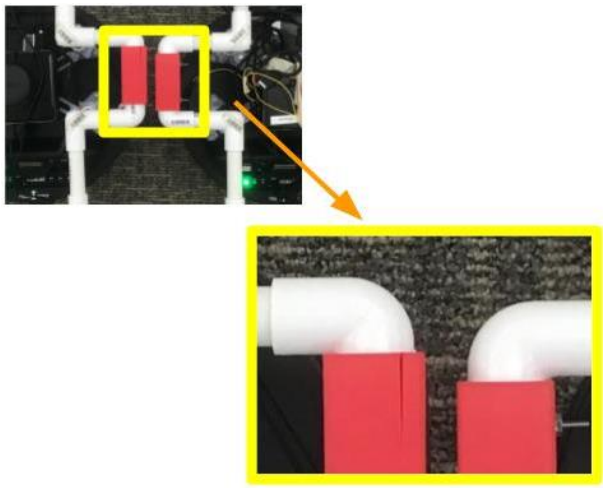


Figure 55: Closeup of docking test 1.

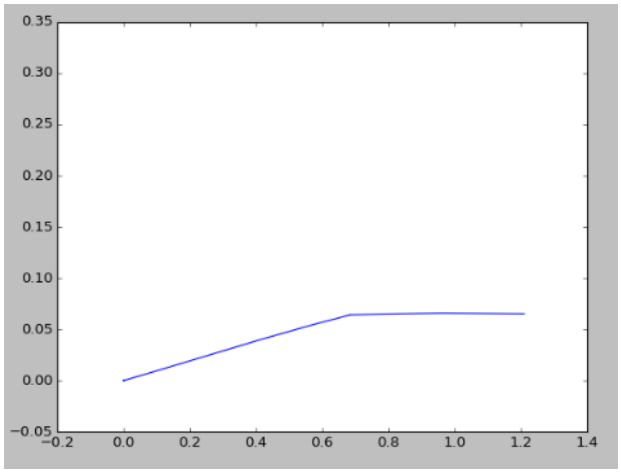


Figure 56: X vs Y robot displacement test 1 in meters.

Action: Increase X offset by 0.005 m

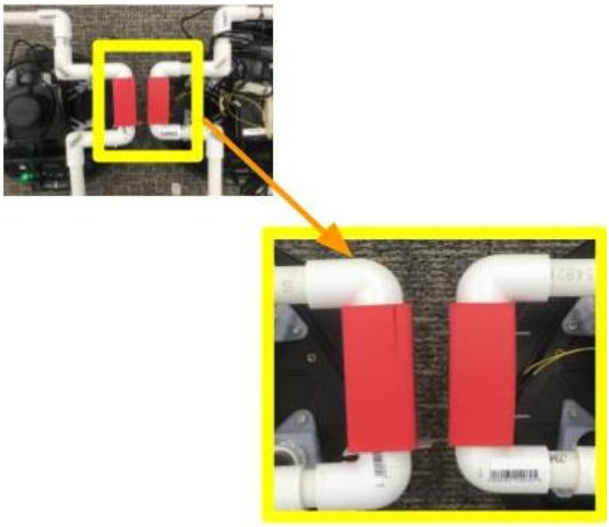


Figure 57: Closeup of docking test 2.

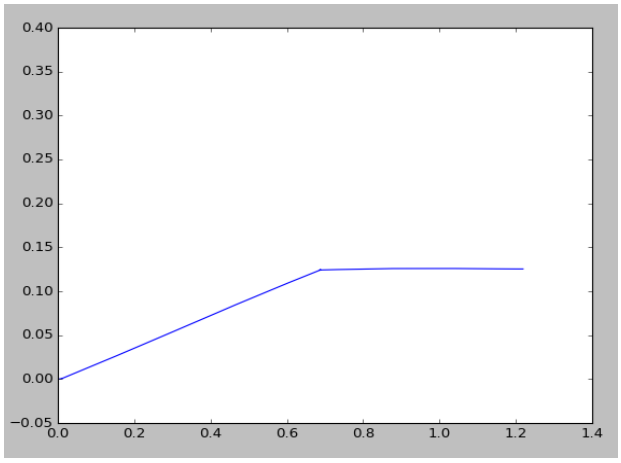


Figure 58: X vs Y robot displacement test 2 in meters.

Action: No further action is required. The robots were consistent from this point forwards.

2. Comparison with expected results

We were able to fine tune the docking procedure to remove the X offset of 0.005 m. We still had a Y offset of 0.003 m however we fixed this problem by redesigning the docking mount.

11.6 Navigation Stack Tuning

Vincent Tran performed this experiment

There are certain parameters in the navigation stack that affect the path and goal of the robot. Below are several tests conducted of each parameter that affected the movement of the robot's path.

Parameter Descriptions

yaw_goal: The tolerance in radians for the controller in yaw/rotation when achieving its goal

xy_goal: The tolerance in meters for the controller in the x & y distance when achieving a goal

vx-samples: The number of samples to use when exploring the x velocity space

vth_samples: The number of samples to use when exploring the theta velocity space

max_vel_x: The maximum x velocity for the robot in m/s.

max_rot_vel: The absolute value of the maximum rotational velocity for the robot in rad/s

Table 5: dwa_local_planner parameters.

Test #	yaw_goal tolerance (radians)	xy_goal tolerance (m)	vx_samples (m)	vth_samples	max_vel_x (m/s)	max_rot_vel (rad/s)
1	0.03	0.02	5	5	0.1	.2
2	0.05	0.04	10	10	0.2	0.3
3	0.1	0.06	15	15	0.3	0.5
4	0.2	0.08	20	20	0.4	0.7
5	0.3	0.10	25	25	0.5	0.8
6	0.35	0.12	30	30	0.6	1.0
7	0.4	0.14	35	35	0.7	1.4

Below are the suggested values for the parameters tested above from ROS.

Table 6: Suggested default parameters for ROS Navigation Stack.

Suggested Default Parameters in ROS

yaw_goal tolerance (radians)	xy_goal tolerance (m)	vx_samples (m)	vth_samples	max_vel_x (m/s)	max_rot_vel (rad/s)
0.05	0.10	3	20	0.55	1.0

1. Test Results

Table 7: Navigation stack tuning test results.

Test	Results
yaw_goal tolerance (radians)	**Best Result: 0.3** The best execution time and accepted precision was at 0.3 radians. It did not take too long to find its goal, and we were able to work with its orientation at its goal. **Values < .3:** The yaw_goal tolerance was too precise, causing the robot to take a longer execution time while spinning in place to find its goal. **Values > 0.3:** The yaw_goal tolerance was too lenient. The robots would assume they found their goal, but would be a few radians away from the exact point depending how big we set the tolerance. to be.
xy_goal tolerance (m)	**Best Result: 0.05** Setting the xy_goal tolerance to 0.05 was not the most precise value, but its execution time to be within its goal was the deciding factor. We did not need the robots to navigate too precisely in these parts, because we had other controllers to realign the robot. **Values < .05:** The xy_goal tolerance was too precise, causing the robots to never reach a goal. The robots will keep trying to correct itself by moving back and forth until its x and y position are within the tolerated goal. **Values > 0.05:** The xy_goal tolerance was too lenient. The robots would assume they found their goal but would be too far off from the desired point depending on how big we set the tolerance to be.
vx_samples (m)	**Best Result: 20** The more samples we allowed the navigation stack to forward determine and simulate its projection, the longer the robots took to execute. It was not severely longer, but it was also not much more precise when reaching its goal. This could be due to friction on the wheels. Twenty samples were sufficient for our system.
vth_samples	**Best Result: 20**

(m)	Setting the sample size to 20 was sufficient, as the robot's path was smooth, and achieved its goal with its other configured parameters. We observed that having less than 15 samples would cause a rougher path for the robots to take. Any more than 20 samples was not noticeable.
max_vel_x (m/s)	**Best Result: 0.3** Having the robots move at 0.3 m/s was the best decision because the transforms kept in the tf tree between the robot's odometry and map would continue to line up correctly. Any slower would be unnecessary. **Values > 0.3:** With speeds greater than 0.3m/s would not allow the map to update fast enough and publish the correct transform. This would ultimately misalign the robots localization and cause the robot's goal to be in a completely wrong spot.
max_rot_vel (rad/s)	**Best Result: 0.7** Similarly to max_vel_x, having the robots rotate at 0.7 rad/s was the best decision because the transforms kept in the tf tree between the robot's odometry and map would continue to line up correctly. Any slower would be unnecessary. **Values > 0.7** With theta speeds greater than 0.7 rad/s would not allow the map to update fast enough and publish the correct transform. This would ultimately misalign the robots' localization and cause the robot's goal to be in a completely wrong spot.

2. Corrective Actions Taken

Selected Parameters					
yaw_goal tolerance (radians)	xy_goal tolerance (m)	vx_samples (m)	vth_samples	max_vel_x (m/s)	max_rot_vel (rad/s)
0.05	0.10	3	20	0.55	1.0

11.7 Outdoor Testing

We performed the same demo outdoors as we did indoors.

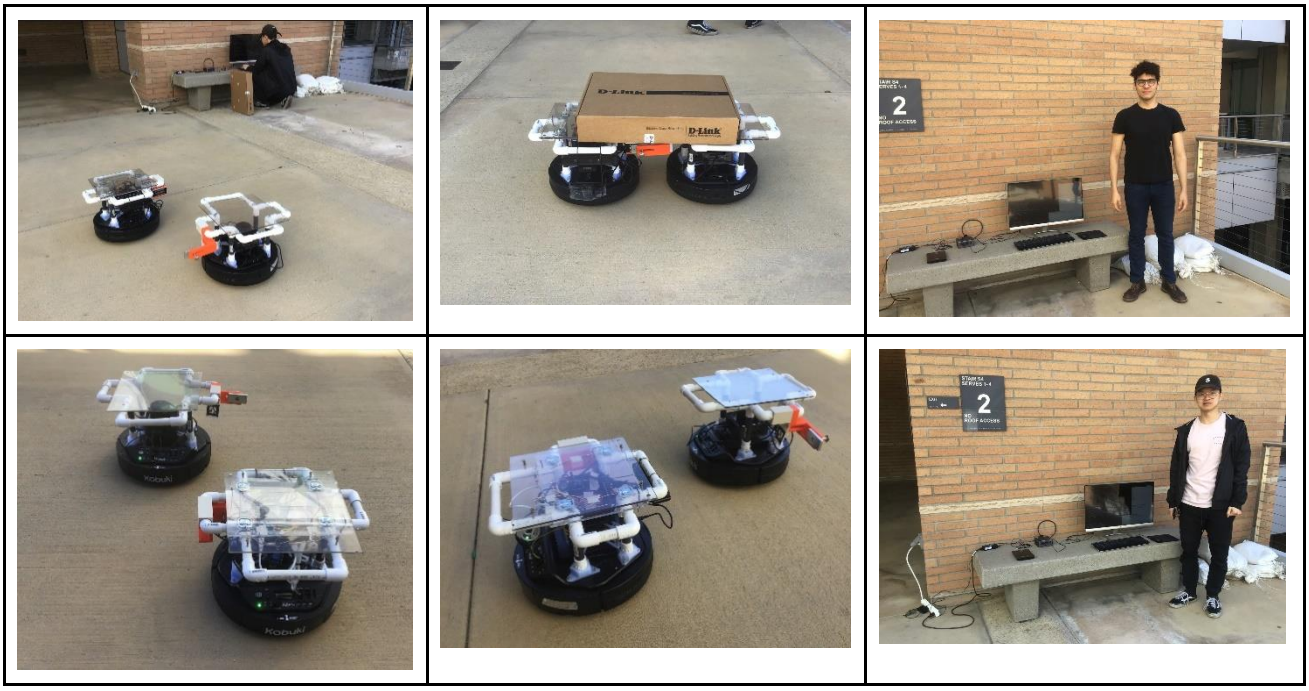


Figure 59: Outdoor test pictures.

1. Comparison with expected results

   As expected, we were able to demo the same project outdoors.

2. Analysis of Test Results

   There weren't any problems, except sometimes the Wi-fi connection may sometimes disconnect as it is outdoors. The lidar, RGB camera, and robots worked perfectly.

12 Conclusion and Future Work

12.1 Conclusion

Overall, we successfully completed our project and our demo worked out as planned. We met all of our own individual goals and came together to piece it all together. Learning how to use ROS (Robot Operating System) was an important task to complete this project.

Gustavo was responsible for the mechanical design and 3D prints of all the parts we have on our robot, such as the magnet mount, and stand for the robot. With Vincent's creation of the tf tree, Gustavo used the RGB camera to find the pose of the AR tag that corresponds to the location of the robot and electromagnet. Utilizing that information, he is able to dock both robots together by using a PID controller. He programmed the coordinated movement for when both robots are docked together by imitating the differential drive concept. Gustavo and Vincent both worked on the state machine to create the demo.

Kyle was responsible for the GUI (Graphical User Interface) of our robotic system. Inside the GUI displays the map produced by the Lidar, system diagnostics, way point navigation, state machine display, and first-person point of view camera from the RGB camera. He was responsible for all the schematics on our system, which consists of wiring together the robot latching mechanism and the load sensor module. He implemented the load sensor module to detect whether a second robot would need to come and dock together to transport the object. Kyle interfaced the Arduino with ROS and performed serial communication.

Vincent was responsible for making a scalable robot system to be able to launch an N number of robots. He was also responsible to integrate the Lidar onto the robot by creating the URDF for the robot. He created the tf tree in order for the system of robots to keep track of all coordinate frames at every second. Vincent set up the navigation stack to create a path and goal for the robots to arrive to with accurate precision. The navigation stack allows the robot to be able to return to its origin point as well. He performed SLAM to localize each robot and generate a map for both robots. Vincent and Gustavo worked on the state machine to create the demo.

We all learned that it takes time and patience to work with new equipment such as Lidar or RGB camera. It requires lots of designing and redesigning for schematics and hardware designs. It requires a lot of testing and tuning before everything can be put together. A lot of time has to be put into research to learn about new topics, such as robotics and understand how coordinate frames of every joint and link correlate, and how to do the math. We learned that in order to complete a big and complex project, each and every team member must contribute as much time and put in the same effort.

12.2 Future Work and Future Improvements

Distribution warehouses would be able to use our product in order to effectively replace most human workers in the warehouse, leading to reduced costs and therefore increased profits. Distribution companies like Amazon currently utilize teams of robots, each individually transporting package towers across warehouses. However, using teams of robots can lead to increased scalability of warehouse capabilities, can improve shipping efficiency, and can reduce strain on human operators.

For future research purposes, we would like to turn our project into a ROS package for others to use. We would also like to create our own robots from scratch and design our own PCB for it. We could use better sturdy parts to be able to carry a heavy load that Amazon uses for their robots in their distribution warehouses.

12.3 Acknowledgement

Autonomous Robots and Control Systems

UNIVERSITY OF CALIFORNIA UCRIVERSIDE

Advisors: Dr. Roman Chomko, Dr. Konstantinos Karydis, Manglai Zhou

ARCS graduate students: Keran Ye, Hanzhe Teng

13 References

[1] A. Saenz-Otero, D. W. Miller, "SPHERES: a platform for formation-flight research", Proc. SPIE 5899, UV/Optical/IR Space Telescopes: Innovative Technologies and Concepts II, 58990O (24 August 2005); doi: 10.1117/12.615966

[2] B. Gabrich, D. Saldana, V. Kumar, and M. Yim, “A flying gripper based on cuboid modular robots,” in IEEE International Conference On Robotics and Automation 2018, Brisbane, Australia, 2018.

[3] G. Li, B. Gabrich, D. Saldana, J. Das, V. Kumar, and M. Yim, “Modquad-vi: a vision-based self-assembling modular quadrotor,” in IEEE International Conference On Robotics and Automation 2019 (to be presented), Montreal, Canada, 2019.

[4] J. Alonso-Mora, R. A.Knepper, R. Siegwart, D. Rus. "Local motion planning for collaborative multi-robot manipulation of deformable objects.." Paper presented at the meeting of the International Conference On Robotics and Automation, 2015.

[5] Z. Wang, Yang, G., Su, X., and Schwager, M., “OuijaBots: Omnidirectional Robots for Cooperative Object Transport with Rotation Control Using No Communication”, Proc. of the International Symposium on Distributed Autonomous Robotics Systems (DARS). Springer, pp. 117-131, 2018.

[6] https://www.nasa.gov/offices/oct/home/roadmaps/index.html

[7] https://iclebo-kobuki.readthedocs.io/en/latest/specifications.html

[8] http://www.ros.org/

14 Appendices

Presents information that supplements the design specification, including:

Appendix A: Parts List

Table 8: Parts List

Item	Price	Qty.	Subtotal
TurtleBot 2.0 (Kobuki)	$320.00	2	$640.00
Intel Nuc	$400.00	3	$1,200.00
RPLIDAR A2	$320.00	2	$640.00
19V Portable Battery Pack	$50.00	2	$100.00
Arduino Uno	$25.00	1	$25.00
Orbbec Astra Pro 3D Camera	$160.00	1	$160.00
12V Electromagnet	$24.00	1	$24.00
5V Relay	$10.00	1	$10.00
Load Cell - 50kg	$11.00	4	$44.00
Load Cell Combinator	$2.00	1	$2.00
HX711 Load Cell Amplifier	$10.00	1	$10.00
Molex Connector for Kobuki	$0.45	1	$0.45
Crimp for Molex Connector for Kobuki	$0.15	1	$0.15
		Total	**$2,855.60**

Appendix B: Equipment List

3D Printer: Makerbot Replicator 2

Soldering Station: Weller WES51

Drill: Black and Decker LDX 120c

Dremel Tool: Dremel 4000

Laptop: Dell XPS 13 (Windows 10)

Appendix C: Software Repositories

https://github.com/gustavojcorrea/turtlebot_mrs

https://github.com/gustavojcorrea/ucr-ee-senior-design

Appendix D: Special Resources

We consulted with ARCS graduate students Keran Ye and Hanzhe Teng. We also consulted with Dr. Konstantinos Karydis, and were helped by Manglai Zhou of the UCR BCOE EE Shop.

Appendix E: User's Manual

Kobuki 2.0 Specs: https://iclebo-kobuki.readthedocs.io/en/latest/specifications.html

Appendix F: Mechanical Specs

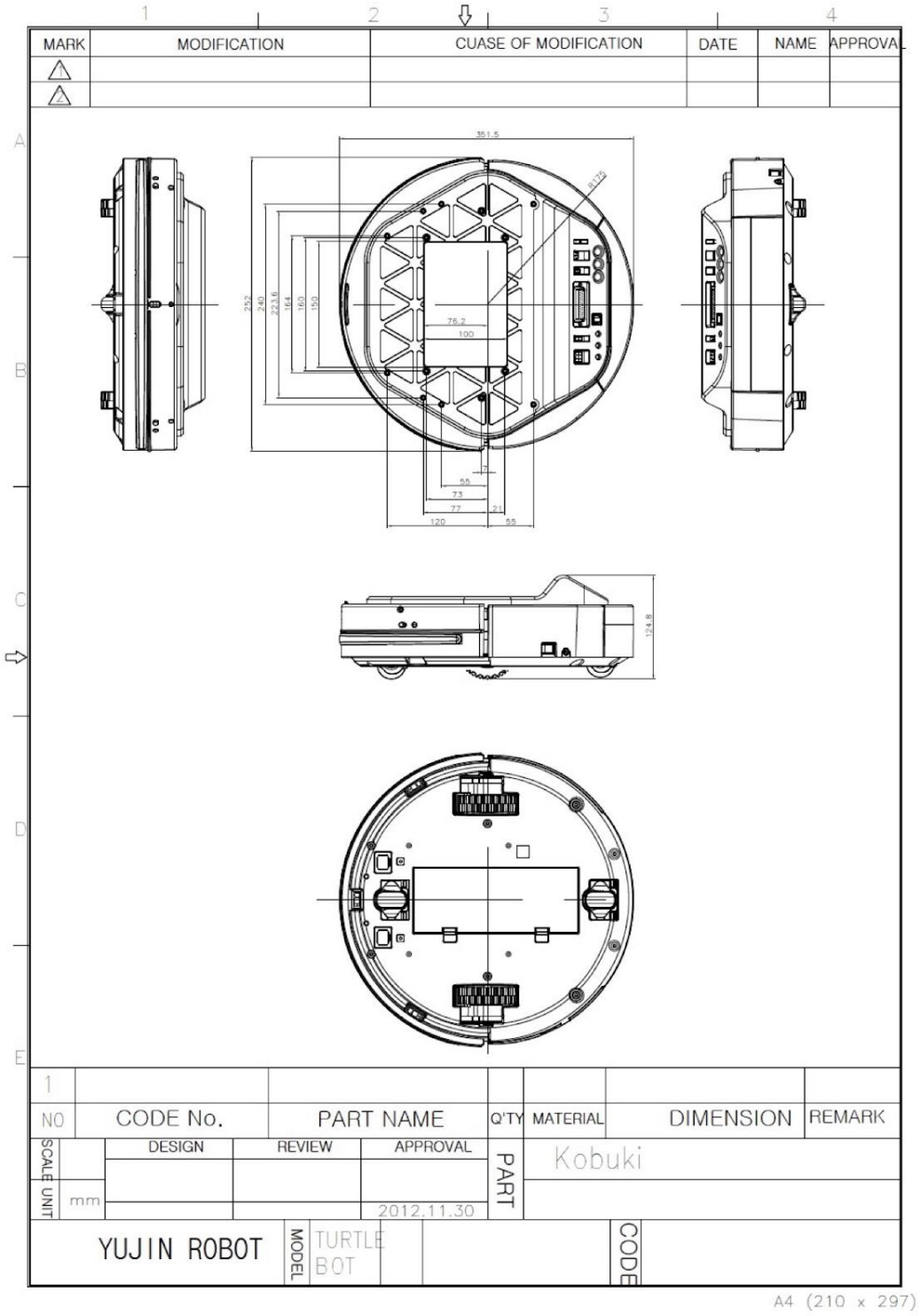
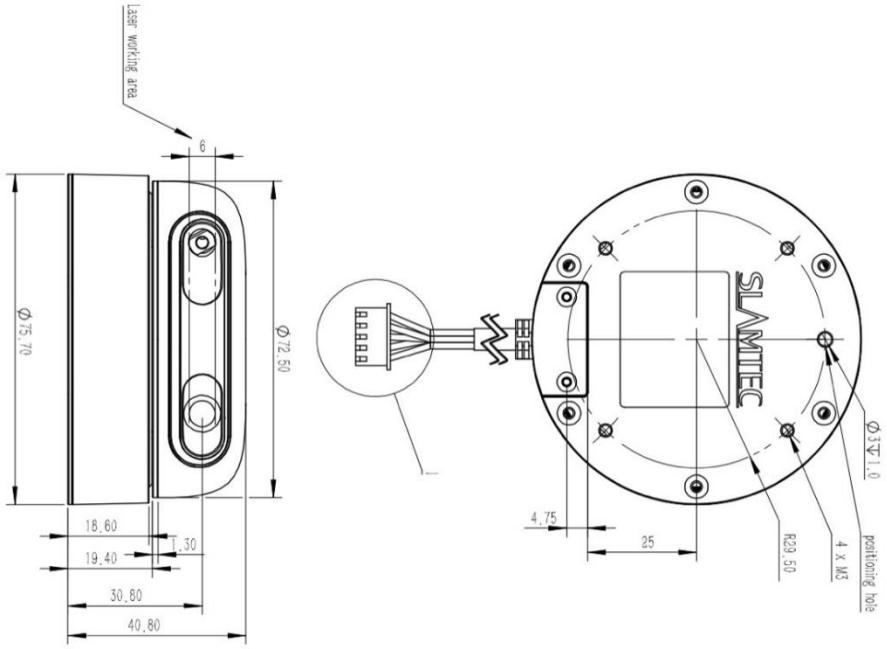
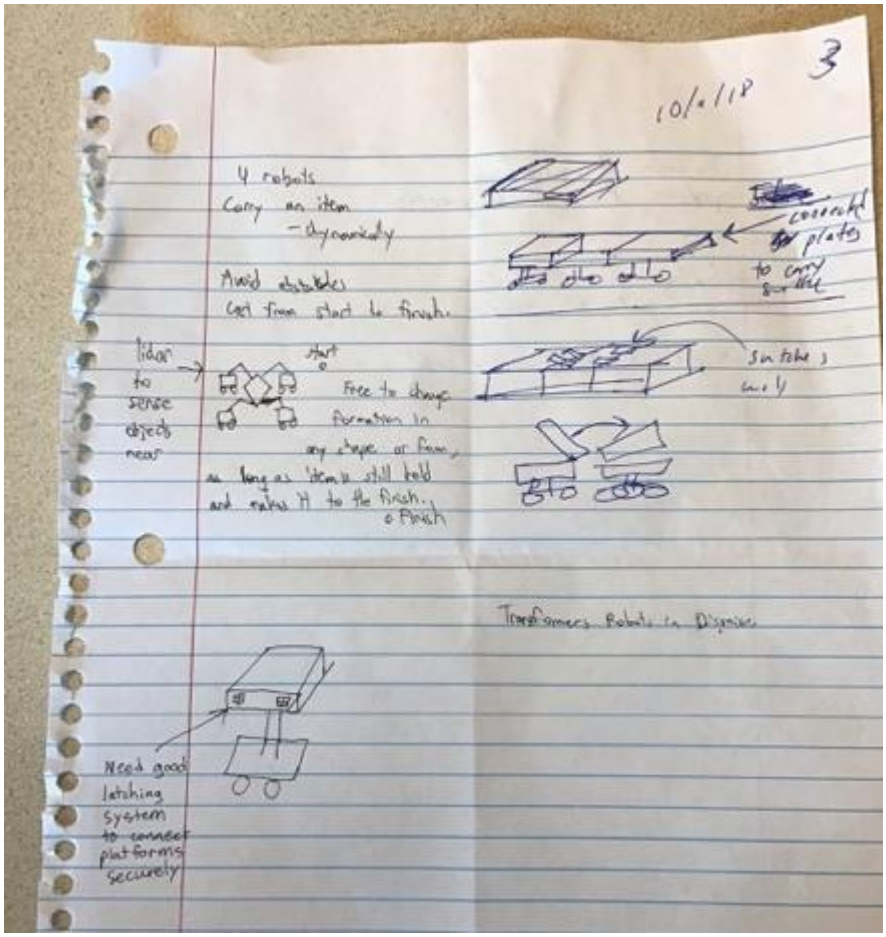


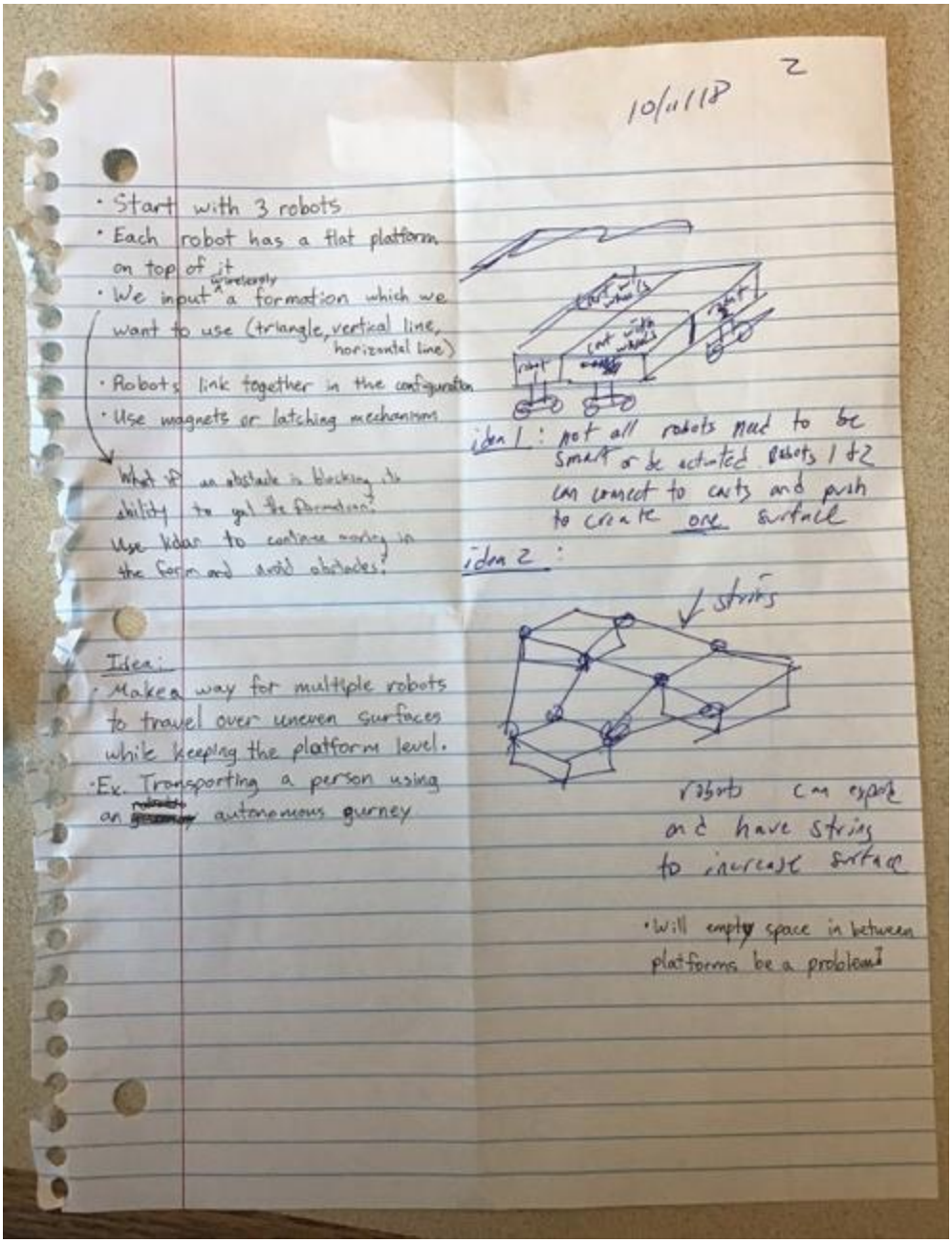
Figure 60: Kobuki TurtleBot Mechanical Drawing.

The mechanical dimensions of the RPLIDAR A2 are shown as below:



Appendix G: Hand-Drawn Design Notes




10/11/18
2
• Start with 3 robots
• Each robot has a flat platform on top of it
• We input a formation which we want to use (triangle, vertical line, horizontal line)
• Robots link together in the configuration
• Use magnets or latching mechanisms
What if an obstacle is blocking its ability to get the formation?
Use Kinect to continue moving in the form and avoid obstacles?
Idea:
• Make a way for multiple robots to travel over uneven surfaces while keeping the platform level.
• Ex. Transporting a person using an autonomous gurney
idea 1: not all robots need to be smart or be actuated robots 1 & 2 can connect to carts and push to create one surface
idea 2:
strings
robots can expand and have strings to increase surface
• will empty space in between platforms be a problem?