Roombotics

James Deere, Taylor Mosser, Todd Van Klaveren

Concept of Operations

Version – 2.0
25 April 2023

CONCEPT OF OPERATIONS
FOR
Roomba Multi-Robot System

TEAM ROOMBOTICS

APPROVED BY:

_____
Project Leader Date

_____
Prof. Kalafatis Date

_____
T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
0.0	09/07/22	James Deere		First Draft
1.0	11/21/22	James Deere		Revision
2.0	04/25/23	James Deeree		Final

Table of Contents

List of Tables

List of Figures

1. Executive Summary

Our project sponsor wants to be able to deploy at least two Roombas in the same room or building and have them work together to efficiently clean the building. This means that no Roomba should cover the same ground as another and no Roomba should cover the same area twice. A Roomba does not work with any intelligent pattern, rather it makes random movements until the area is clean. What our project would entail is adding an intelligent pathfinding algorithm and a real-time progress report of what area has been covered. This would not be done by an individual Roomba, but by a single base station that commands each Roomba that is in use. The intelligent use of multiple Roombas will not only be more power efficient, but will also clean a given area much faster.

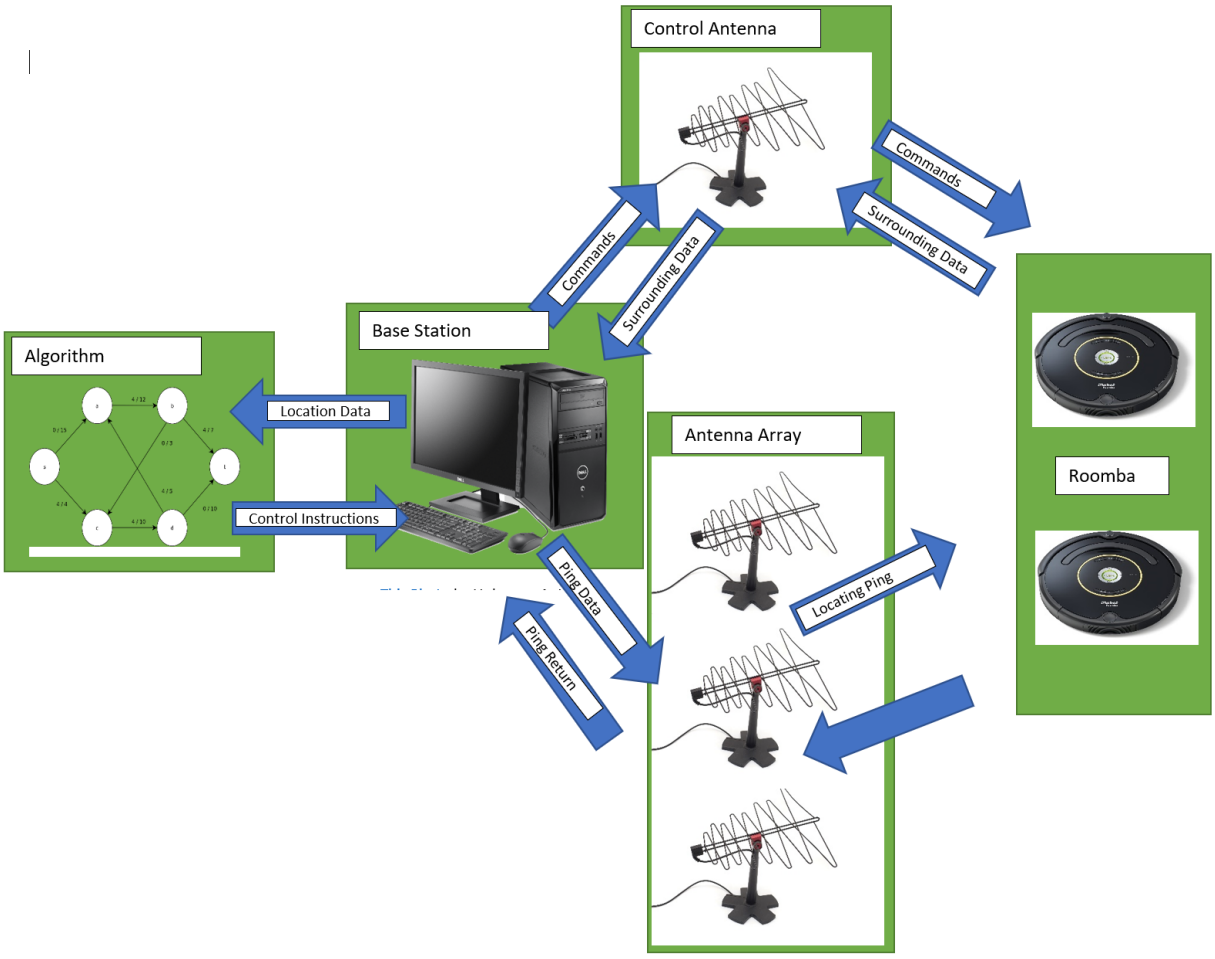


Figure 1: Representation of Roomba Multi-Robot System

2. Introduction

2.1 Background

Many families and businesses have turned to using robot vacuum cleaners instead of manually-operated vacuum cleaners for a more convenient cleaning process. One of the most popular robot vacuum cleaners is the iRobot's Roomba, a disk-shaped model that navigates the rooms and hallways of any building. The Roomba is effective at cleaning around and underneath furniture, and it quickly changes its path when it comes into contact with walls, doors, or sitting objects in a room. Despite all of its useful features, the Roomba's method of navigation and obstacle avoidance is not the most efficient. It will move around randomly, either in an outward spiral or a relatively linear path, and it will not sense an object in front of it without bumping into it. There is no guarantee that the Roomba will not cover the same spot twice or that it will cover every inch of the room. Additionally, Roombas are not capable of communicating with other Roombas in order to coordinate their efforts. Shown below is a typical pathing of a Roomba as it cleans a room. it starts forward until it finds a wall, and then begins to sweep across the room at slight offset angles. Note that this method has a large number of overlapping paths.

Figure 2: Light diagram showing Roomba's typical path

2.2 Overview

This multi-robot system design aims to create and simulate a pathing algorithm that can be programmed onto a Roomba so that it takes the shortest path between the starting point and destination. Instead of navigating the space in a random fashion, the Roomba will move across in a way in which it covers and cleans the entire space while still avoiding any obstacles. Through the use of an intelligent searching algorithm, the best path can be found in simulation and later implemented in the execution by the Roomba.

To shorten the amount of time it takes for a Roomba to clean a certain space, users may wish to add another Roomba to work in cooperation with the first. These two Roombas must work together in a strategic fashion to clean the space while not interfering with each

other's tasks. One will not only avoid collision with the other, but also not cross into the area in which the other has covered or is currently covering.

Roombas provide many advantages with its user-friendly functionality, but its navigation system must be greatly improved to operate more efficiently. The Roombas are controlled by a singular base station that will plan the paths for them. Localization is determined by both an IMU sensor and triangulation between the Roomba, the base station, and other antennas placed around the room. The Roombas will have a sensor suite that will be able to detect obstacles in the way of the Roomba's commanded path. Through the use of multiple infrared distance sensors the Roomba will be able to sense stationary objects (ex: boxes) and mobile ones (ex: humans). The base station receives data from the Roombas regarding where they are, and if there are any obstacles around it. Using this data and the already determined layout of the room, the base station will consult the algorithm and relay commands to the roombas as to what actions to take next.

The simulation of this process will be created and performed in Unity with the use of C# programming. Roombas will navigate through an established workspace using the A* pathing algorithm to cover all ground efficiently. The completed version of this simulation will consider the two Roombas to be operating at the same speed, but will also be functional to use with Roombas that vary in speed.

2.3 Referenced Documents and Standards

Document Title	Revision/Release Date	Publisher
Design and Implementation of Pathfinding Algorithms in Unity 3D	April 2022	International Journal for Research in Applied Science & Engineering Technology
Time-efficient A* Algorithm for Robot Path Planning	February 2016	Procedia Technology
"How Does My Robot Navigate"	November 2021	iRobot
"iRobot® Create® 2 Open Interface (OI) Specification based on the iRobot® Roomba® 600 "	April 2, 2015	iRobot

Table 1: Referenced Documents and Standards

3. Operating Concept

3.1 Scope

The Roomba multi-robot system discussed in this report is designed to optimize paths of a variable number of robots to cover a given surface area. Given that design, the multi-robot system is meant to efficiently path-plan for a task that requires a full surface area coverage, but in the future the system can be updated to perform other path-planning tasks such as delivering items, lawn mowing, and weed pulling.

The design and development of this system is divided into three different subsystems: pathfinding and simulation, control system and sensor suite, and robot localization. The exact deliverables are as follows:

- **Pathfinding and Simulation:**
  - Finds efficient paths between start and end points
  - Plans a cleaning path for each Roomba
  - Able to control multiple robots
  - Capable of handling a changing environment
- **Control System and Object Detection:**
  - Ability to accept commands via bluetooth
  - Base station sends commands to Roomba, which then properly executes them
  - Roomba detects the environment accurately to a distance of approximately 0.5 m
  - Roomba sends environment data coherently to the base station
  - Roomba avoids collision when detected
- **Robot Localization**
  - Wirelessly locate each Roomba
  - Each location must be properly attributed to a given Roomba
  - Location accuracy of within a 35 cm circle (less than one roomba in size)
  - Use a minimally invasive communication standard
  - Have support for multiple (more than 3) locating beacons

3.2 Operational Description and Constraints

Users will first upload the floor layout to the multi-robot system. The user will then place at least three antennas and the base station in different corners of the room, and then upload the accurate locations of these stations. Once the antennas are placed and the base station connected, the Roombas can be placed anywhere in the room to begin their tasks.

This system is not meant to be used in outdoor environments or on uneven surfaces such as thick carpet. Roombas perform best when placed on harder surfaces such as concrete, tile, or short carpet. Rougher surfaces can affect the Roomba's ability to clean and navigate across the room.

Due to limitations on hardware cost, the maximum room size for proper localization is 30ft x 30ft. More beacons would be needed to extend this range. The Roombas will be battery powered and will have to contend with battery charge state. They will also need to consider vacuum bin capacity levels when deciding to continue cleaning or return to dock.

While the Roomba multi-robot system will be designed to function in any workspace, the simulations will focus on resembling the real-life controlled environments of a university

lab that will be used to test the robots. This means a generally smaller room size, but with higher levels of foot traffic and variability of obstacles.

3.3 System Description

Pathfinding Algorithm in Unity Simulation:

One subsystem of this project is the simulation that will be performed in a 3D environment in Unity. A pathfinding algorithm that can be used for one Roomba and copied onto multiple Roombas will be implemented in the simulation. The space in which the robots move will resemble the real-life workspace that will be used to test the actual robots. The Roombas will first be tested with equal speeds and navigate the workspace through the use of the A* algorithm. This involves calculating the lowest-cost path from one starting point to another by evaluating all possible paths that the Roomba can take.

Ultimately the Roombas will coordinate in cleaning by covering their own spaces and avoiding interference with each other, while following a predetermined path.

Roomba Control and Sensors:

The second subsystem of this project regards the control of Roomba functions (movement, brush speed, turning) and the sensors that are placed on the Roomba. For the main control, a microcontroller will be integrated into the Roomba. Sensors and external modules on the Roomba (besides the microcontroller) are of three types:

- Proximity Sensors: these are used to detect obstacles
- DC Buck Converters
- Localization antenna and chip. This will be provided by the localization subsection.

Localization:

The localization subsystem uses radio communication in order to perform a Time of Flight measurement of how far the robots are from each locating station. The base station then uses this data to determine the location of each Roomba. The location stations will be DWM1000 modules wired to the base station and placed at measured distances from the base station. The Roombas will each have a DWM1000 module on them to receive and respond to pings from the location stations. This method uses UWB to determine the location of a module in relation to the others. Once the initial technology is built and tested, further testing will be done in order to determine the optimal refresh rate of the localization subsystem.

3.4 Users

The Roomba multi-robot system is an excellent tool for users who want to cover large spaces such as offices, building lobbies, hallways, laboratories, and classrooms. The use of one Roomba in these kinds of environments would be highly inefficient and inconvenient due to its random navigational methods and the large amount of time it would

take to clean an entire space. However, it can be too great a task to assign one person with the duty of cleaning these areas.

This system would permit users to place multiple Roombas that work together to reduce the amount of time it takes to cover the space while avoiding interference with any obstacles or tasks. Users would not have to wait until a room or hallway is empty to deploy the Roombas because the system accounts for any moving objects that cross through the determined path.

3.5 Support

As Roombas are an item typically marketed to individuals with minimal technical knowledge, a very high-level and detailed instruction set must be provided. Most of the operation is automated, but some user setup is needed. This includes placing the base station and antennas around the room, as well as uploading the floor layout of the room. To ensure little confusion, an instruction pamphlet will be provided with the Roomba multi-robot system.

4. Scenarios

4.1 A New Roomba Is Deployed During Operation

If an additional Roomba is turned on mid-operation, the base station will send the new Roomba to an area that has not yet been covered, and the paths of all Roombas will be adjusted to maintain an efficient coverage of the surface area.

4.2 Obstacle Detected In Path of Roomba

When an object has been detected by the sensors on the Roomba, the Roomba will stop until the obstacle is no longer detected in the peripheral sensors. If the obstacle has been stationary for 5 seconds, the obstacle will be registered as a permanent obstacle and the Roomba will go around it.

5. Analysis

5.1 Summary of Proposed Improvements

The Roomba multi-robot system will reduce the amount of time required to clean an entire space. When working alone, the Roomba will take a calculated path about the room that allows it to clean the entire room as quickly as possible.

The use of only one Roomba may take too much time because of its small size and limited speed for a large room or workspace. Using two or more Roombas would allow for more space to be covered at one time. These Roombas would work in cooperation so that they each clean separate areas. This means that there would be no chance of the same spot accidentally being cleaned twice.

5.2 Disadvantages and Limitations

One possible limitation to this system is that it may not be able to detect when a static obstacle has been displaced. If a Roomba approaches a nonmoving object, such as an office chair, it will recalculate its path to move around or beside it. If the chair is moved shortly afterwards during the cleaning process, the Roomba will not be able to detect that this change has happened and continue on its predetermined path. The Roombas will be able to navigate around static obstacles and moving obstacles, but with its determined path it will not return to space that has already been covered to check if anything has been moved since the start.

An inherent issue with robotic cleaners is the finite capacity of a given robot in regards to its vacuum container. In a particularly dirty environment, the roomba could potentially fill its container before it completes a room. In this case, the roomba would not effectively clean after the container was full. This limitation could be avoided by having the roomba measure its container fill level and return to base station for emptying when it is nearly full.

5.3 Alternatives

An alternative way of localizing would be to only use dead reckoning with an IMU. A very big pro of this would be that triangulation would not be used, so antennas would not have to be placed around the room. This would make the multi-robot system more user friendly. The disadvantage of this would be the possibility that the base station could end up with a large miscalculation of where the Roombas actually are.

5.4 Impact

The Roomba multi-robot system plans paths that are more strategic than the paths taken by individual Roombas. This leads to less energy wasted, much shorter duration of noise in the surrounding environment, and an overall cleaner floor. The system will also lead to less required manual cleaning and higher efficiency. Our system is just one application for exploring and making advancements in path planning, leading to a future in which there can be further dependency on robots to perform difficult tasks.

Roombotics
James Deere
Taylor Mosser
Todd Van Klaveren

Functional System Requirements

Version – 2.0
25 April 2023

Functional System Requirements
for
Roombotics

Prepared by:

_____
Author Date

Approved by:

_____
Project Leader Date

_____
John Lusher, P.E. Date

_____
T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
0.0	09/28/22	James Deere		Draft Release
1.0	11/21/22	James Deere		Revision
2.0	04/25/23	James Deere		Final

Table of Contents

List of Tables

List of Figures

1. Introduction

1.1. Purpose and Scope

This report contains information about the functionality requirements for the Roomba multi-robot system. The goal of this project is to determine and implement a pathfinding algorithm onto an iRobot Roomba to improve speed and efficiency in cleaning a room. The Roomba will work in cooperation with one or more Roombas to cover an area in a strategic fashion. The algorithm will first be tested and simulated in an environment in Unity that consists of static obstacles, moving objects, and any other factors that may interfere with a Roomba's movement and completion of tasks. After performing multiple simulations, the algorithm shall be implemented onto a pair of actual Roombas and tested in a controlled workspace.

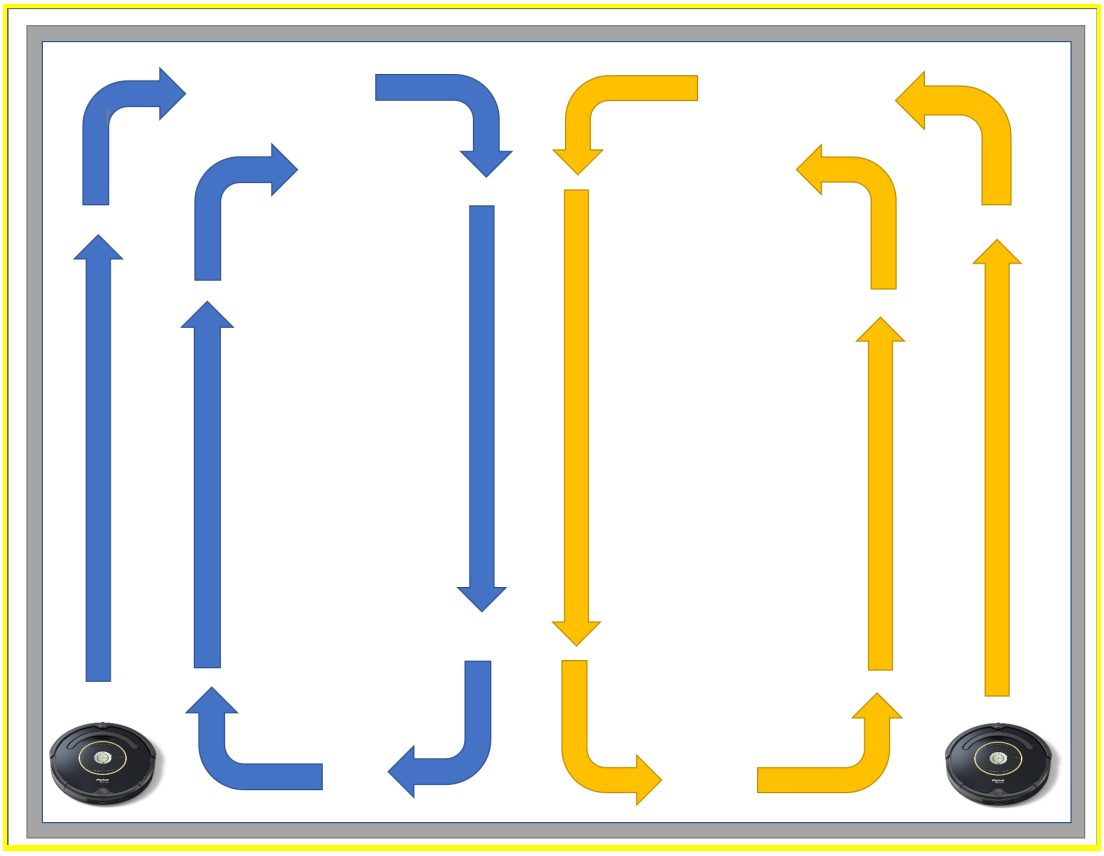


Figure 1. Diagram of Simulation and Functionality of Roomba Multi-Robot System

1.2. Responsibility and Change Authority

Todd Van Klaveren is the team leader and is therefore responsible for making sure all requirements and deadlines are met in a reasonable manner. Eric Robles, the sponsor of this project, must be consulted for any changes in performance requirements. Todd has

authority over implementation and can approve implementation changes. The breakdown of subsystems is as follows:

- Path Planning - Taylor Mosser
- Roomba Control - James Deere
- Localization - Todd Van Klaveren

2. Applicable and Reference Documents

2.1. Applicable Documents

The following documents, of the exact issue and revision shown, form a part of this specification to the extent specified herein:

Document Number	Revision/Release Date	Document Title
	9/23/2022	Unity User Manual 2021.3 (LTS)
	4/15/2017	A* Pathfinding Project Documentation
	9/29/2022	A Tour of the C# Language
	4/2/2015	iRobot® Roomba 600 Open Interface (OI) Specification

Table 1. Applicable Documents

2.2. Reference Documents

The following documents are reference documents utilized in the development of this specification. These documents do not form a part of this specification and are not controlled by their reference herein.

Document Number	Revision/Release Date	Document Title
	2018	Algorithm - iRobot Roomba
	7/8/2021	What Is a Roomba and How Does It Work?

Table 2. Reference Documents

2.3. Order of Precedence

In the event of a conflict between the text of this specification and an applicable document cited herein, the text of this specification takes precedence without any exceptions.

All specifications, standards, exhibits, drawings or other documents that are invoked as “applicable” in this specification are incorporated as cited. All documents that are referred to within an applicable report are considered to be for guidance and information only, except ICDs that have their relevant documents considered to be incorporated as cited.

3. Requirements

This section defines the minimum requirements that the development item(s) must meet. The requirements and constraints that apply to performance, design, interoperability, reliability, etc., of the system, are covered.

3.1. System Definition

The system is divided into three subsystems: pathfinding simulation, control and interface, and localization. The simulation of the Roomba multi-robot system shall be designed to test the A* pathfinding algorithm on the individual robots in an environment that resembles a real-life workspace. This algorithm will later be implemented onto the physical Roombas inside a controlled area. The Roomba's will have an MCU and multiple sensors on them to control the Roomba's movements and to sense any nearby obstacles. In conjunction with the MCU, the Roomba will have an UWB module which in conjunction with 3 or more anchor units will allow the Roomba to be located in the room. This position will be passed to the base station and the pathfinding algorithm.

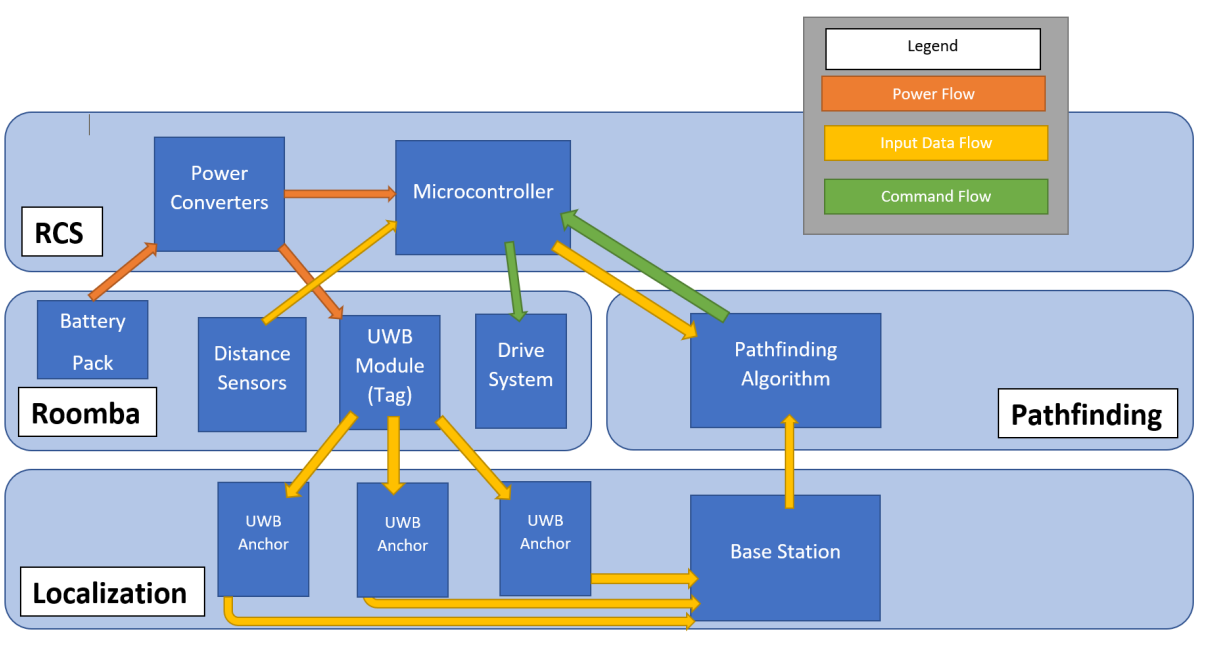


Figure 2. Block Diagram of System

Pathfinding Subsection: The simulated environment will be a large room containing a large rectangular obstacle in the center. The hallways surrounding this obstacle will each have the same width, but their lengths will depend on location. Smaller static obstacles will be placed randomly on the floor of the hallways and the simulated Roombas shall calculate their paths around these objects. Using the A* algorithm, each Roomba will evaluate all possible paths from one starting point to a specific ending point and determine which is the most efficient. After arriving at the ending point, this location will be set as the new starting point and a different location will be set as the new ending point. This process

will be repeated until the Roomba has covered all of the space that it has been assigned to clean. Two Roombas in a space will not interfere with each other's tasks.

Roomba Control System (RCS) Subsection: The Roomba control system will consist of one MCU and a distance sensor. The MCU being used is the ESP32 and the sensors are infrared distance sensors. The Roomba battery will be used to power the ESP32 and the sensors. I2C will be utilized to communicate between the MCU and the sensors, and UART will be used to communicate between the MCU and the Roomba. With this setup the Roomba shall be able to follow commands from the pathfinding subsystem, and stop for obstacles that are in the path.

Roomba Localization Subsection: The Roomba Localization System (RLS) will consist of a set of UWB modules configured as anchors or tags. Each Roomba will have a UWB module configured as a tag. The system will have at least one tag and at least 3 anchors. At a set polling rate, each tag will attempt to contact the anchors. The anchors will wait a predetermined amount of time, and then respond. Using a TOF method of calculation, the distance from tag to anchor will be determined. The base station will perform a calculation and triangulate the exact position of the tag module in the room. The base station will then relay this information to the control program running the pathing algorithm.

3.2. Characteristics

3.2.1. Functional / Performance Requirements

3.2.1.1. Simulated Pathfinding Algorithm

The A* algorithm will be used in the Unity simulation to determine the navigational strategy of each Roomba. The algorithm will be implemented through a script written in the C# programming language and must adapt to static or moving obstacles in the environment.

3.2.1.2. Delegated Area of Operation

Each Roomba will be assigned its own area of the workspace in which it will operate. The size of each area will depend on the speed in which the Roomba moves and the number of obstacles present.

Rationale: A Roomba will not travel into a space that has already been assigned to or covered by another Roomba.

3.2.1.3. Cleaning Time

The time required for the Roombas to clean the environment is a variable dependent on the speed at which the Roombas operate. Any space that is covered by at least one Roomba shall not be covered again. Cleaning time ends when all space has been covered by one of the Roombas in the system.

Rationale: Roombas will not be set to operate for a specific duration of time. They will return to their base stations once they have covered all areas of the workspace.

3.2.1.4. Roomba Charging

The Roomba's battery has a nominal voltage of 14.4 volts. When the battery is depleted by around 80% of nominal value, the Roomba shall stop where it is and go to the charging station.

Rationale: Roombas cannot operate with a heavily depleted battery therefore when the battery is in a depleted state but still functioning, the Roomba battery needs to be recharged.

3.2.1.5. Roomba Control

The MCU will deliver UART packets to the Roomba. The UART packets delivered to the Roomba will indicate the proper movements to be taken. A 3.3V to 5V level shifter is used to facilitate communication between the MCU and the Roomba.

Rationale: Through the mini-din7 connector, Roomba's take UART communication protocol to control the motors. The Roomba uses UART at 5V logic, but the ESP32 uses UART at 3.3V logic so a level shifter is required.

3.2.1.6. Object Detection

Objects shall be detected within 7-15 centimeters. We are limited to the front of the Roomba for object detection for monetary reasons, but considering that the Roomba has a 0 degree turn radius and therefore does not need to go backwards, this is not an issue. Once an object is detected the Roomba will stop moving for 5 seconds, and if the object is redetected after that downtime the Roomba will move around the object and have it marked as a static obstacle.

Rationale: 5 seconds is long enough for a dynamic obstacle to move out of the way, but short enough to not be too time demanding.

3.2.1.7. Localization

The Roomba shall have a localization tag on it, that which in conjunction with 3 or more anchor devices, will be able to determine the location of the Roomba within the workspace. The accuracy of the system is to be accurate within one Roomba's width. This equates to about 13" or 34cm. This is the maximum distance allowed.

Rationale: The distance of 13" the likely grid size for the pathing algorithm; thus the closest grid square the roomba appears to be in is a safe estimation without exceeding the accuracy of UWB ranging ability.

3.2.2. Physical Characteristics

3.2.2.1 Mass

The mass of each Roomba in the multi-robot system shall be around 3.5 kg.

Rationale: Almost all models and versions of the iRobot Roomba have the same measurement of mass.

3.2.2.2 Volume Envelope

Each Roomba has a diameter of 34 cm and a height of 9 cm.

Rationale: Almost all models and versions of the iRobot Roomba have the same volume.

3.2.2.3 Transition Height

The maximum height that a Roomba can raise to when traveling across a changing surface is 1.6 cm.

Rationale: Each Roomba can slightly adjust its height to accommodate small changes in surface level.

3.2.3. Electrical Characteristics

3.2.3.1 Inputs

a. No sequence of commands from the base station shall damage the Roomba control and localization systems, reduce its life expectancy, or cause any malfunction.

Rationale: Error and malfunction shall be minimized by the design of the system

3.2.3.2 Power Consumption

a. The maximum peak power of the combined Roomba control and localization system shall not exceed 8 watts. Power for these systems will be consumed from the Roomba battery.

Rationale: Power is consumed in known quantities by two DC-DC converters utilized by the control system.

3.2.3.3 Input Voltage Level

The input voltage level for the MCU and sensors is 3.3V and 5V for the UWB module. These voltages will be supplied by the Roomba battery through two separate DC-DC converters.

Rationale: These voltages are the factory set options for the sensors and control modules we will be using. They do not have any adjustability.

3.2.3.4 Input Noise and Ripple

All power supplied is DC and ripple will be minimized with bypass capacitors on power rails.

Rationale: It is good practice to put a capacitor on power rails to minimize output ripple.

3.2.3.5 Outputs

3.2.3.5.1 Data Output

The Roomba control system shall include an interface compatible with the Roomba and the base station.

Rationale: This interface will carry commands from the algorithm to the Roombas, and carry obstacle data from the Roomba to the base station.

3.2.3.6 Connectors

The MCU will use a 7 pin mini-DIN connector to interface with the Roomba. This connector has two positive voltage pins from the Roomba battery, two ground pins, one device detect pin, and a Tx and Rx pin for UART communication. Each UWB module will have a micro-USB 2.0 connector used for power input and programming. Power input will be 5v @ 0.5A for each UWB module.

3.2.4 Software Requirements

The required software for this project is listed below.

3.2.4.1 Unity Real-Time Development Platform

The simulation of the multi-robot system should be designed using Unity 2021.3.8f1 on the Microsoft Windows operating system. Assets and packages that are compatible with the platform may be imported from external sources such as Github or the Unity asset store. Scripts must be written in C# programming language and must compile correctly for use in a Unity environment.

Rationale: Unity platform will host the designed environment for testing the pathing algorithm on the Roombas.

3.2.4.2 Visual Studio IDE and Code Editor

All C# scripts will be edited in the Microsoft Visual Studio 2019 development environment. Visual Studio is compatible with the Microsoft Windows operating system and can be instantly accessed through the script inspector in Unity.

Rationale: C# is a programming language compatible with the Unity platform and can be used to make additions and changes in the development of the simulation.

3.2.4.3 MCU Configuration

C++ will be the programming language utilized in the MCU configuration. For direct interface with the Roomba, UART serial communication will be programmed. UART, SPI, and I2C serial packets will be sent and read by the MCU.

Rationale: C++ is compatible with the ESP32 and is a good language for serial communication.

3.2.4.4 Roomba Control Software

The Roomba Control Software will be programmed in C++11, with all necessary libraries to be deployed on an up to date Windows 10 operating system, running on the base station.

Rationale: C++ is the primary programming language for Windows 10 application development.

3.2.4.5 UWB Module Firmware and Programming

The UWB modules will run the latest, stable version of the Zephyr firmware. This firmware is validated to work on the DWM1001 dev boards and provides for the programming and use of the UWB chipset and the onboard bluetooth module.

Rationale: The DWM1001 is supported in use with zephyr as an alternative to creating our own bluetooth library from scratch.

4. Support Requirements

4.1.1 Base Station Computer Requirements

The base station computer shall be a computer with at least a dual core processor, having a method of bluetooth communication, and have an updated version of Windows 10 installed on it.

Rationale: The base station computer must be able to run the control program, communicate to the Roombas with bluetooth, and simultaneously make pathing decisions.

4.1.2 Localization Anchors and Tags

4.1.2.1 Quantity of Anchors

The workspace must be set up with at least, but not limited to, three (3) UWB modules set up as anchors.

Rationale: Triangulation requires at least 3 distances from known points to accurately determine location of the tag module.

4.1.2.2 Location of Anchors

The anchor modules must be placed at least 6 ft from each other, preferably on opposing corners of the workspace.

Rationale: The distance between known points must be large enough that the tag does not often find a space of each individual distance from tag to anchor being similar. This is to keep the accuracy of triangulation high.

4.1.2.3 Reporting of Location of Anchors

The location of each anchor relative to the base station must be accurately input into the base station before running the Roomba Control System program.

Rationale: In order for triangulation to be calculated by the base station, it must know the exact position of each anchor module.

4.1.2.4 Quantity of Tags

Each Roomba shall have an UWB module installed on it, programmed as a tag. Each tag will have a unique identifier assigned to it.

Rationale: Unique identifiers ensure the location of each Roomba is individually attributed to the correct Roomba.

4.1.3 Mounting Modules onto Roomba

The PCB, UWB module, and sensor modules shall be mounted onto the Roomba in a sleek and elegant way, allowing for no blindspots.

Rationale: A mounting method needs to be used to contain the modules on the Roomba.

Appendix A: Acronyms and Abbreviations

cm	Centimeter (0.01 meters)
DC	Direct Current
Dev	Development
kg	Kilograms (1,000 grams)
MCU	Microcontroller Unit
MRS	Multi-Robot System
OS	Operating System
UWB	Ultra-Wide Band

Appendix B: Definition of Terms

Operating System	The program or software used to run the applications on a computer.
Pathfinding	A strategic method of determining the shortest possible path between one point to another.

Roombotics

James Deere
Taylor Mosser
Todd Van Klaveren

Interface Control Document

REVISION – 2
25 April 2023

Interface Control Document
for
Roombotics

Prepared by:

_____
Author Date

Approved by:

_____
Project Leader Date

_____
John Lusher II, P.E. Date

_____
T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
-	09/30/22	Taylor Mosser		Draft Release
1.0	11/21/22	Taylor Mosser		Revision
2.0	04/25/23	James Deere		Final

Table of Contents

List of Figures

1. Overview

The Interface Control Document (ICD) for the Roomba Multi-Robot System will detail the specifications for designing and using each subsystem. The different components of the physical and electrical interfaces, as well as the communication protocols, will be listed and described in the following sections. This document also contains information on the way in which the subsystems will function individually and when integrated together in the entire system.

2. References and Definitions

2.1. References

iRobot CREATE - OPEN INTERFACE Manual
2006 Edition

2.2. Definitions

Asset	Item such as a 3D model or audio file that can be added into a game or simulation design.
cm	Centimeters (0.01 meters)
Game Object	Objects such as obstacles or characters that can placed into a game with a specific functionality.
IDE	Integrated development environment
kg	Kilograms (1000 grams)
MCU	Microcontroller Unit
Scene	The space that contains the game environment, objects, and overall simulation in Unity.
UWB	Ultra-Wide Band

3. Physical Interface

3.1. Weight

3.1.1 Weight of Roomba Device

Each Roomba vacuum will weigh 3.5 kg. Most models weigh 3.5 kg, no matter the version or year. The simulated game objects in Unity will be tested with this exact weight. The modules that are being added onto the Roomba (MCU, UWB module, distance sensors) will have a negligible effect on the weight of the Roomba (about 5% or less).

3.2. Dimensions

3.2.1. Dimension of Roomba Device

A Roomba has a diameter of 34 cm and a height of 9 cm. Most models are designed with these exact dimensions so that they are large enough to cover and clean several centimeters of space at a time, while also being small enough to avoid interfering with activities occurring in the environment. The simulated Roomba vacuums will be designed to have these measurements. The external modules will add a small amount of height to the Roomba (less than an inch), however there will still be a nearly negligible change in the height clearance.

3.3. Mounting Locations

3.3.1 Sensor Mounting and Obstacle Detection

The Roomba's sensors shall be able to detect an obstacle approaching without any blind spots. The sensor mounting poses an interesting situation. At $12.90 a piece, it is crucial to not use any more than necessary. That is why only 4 sensors will be used in a criss-cross pattern as shown below. Utilizing this method, there will be no blind spots and no resources will be wasted. This placement will be on the top of the front of the Roomba to detect any objects that are suspended at about the height of the Roomba. Obstacles will be detected approximately 6 inches in front of the Roomba.

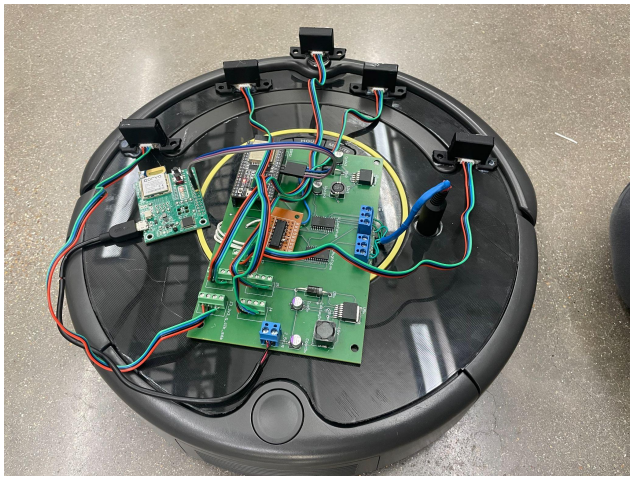


Figure 1: Sensor Placement

3.3.2 Control and Localization Modules

The control system uses an MCU (ESP32), 2 DC-DC converters, and a logic level shifter (3.3V to 5V for serial communication) all on a single board. The localization system will have an UWB module that communicates with the control system's MCU and is on the same board. This will be small enough to discreetly be mounted onto the top of the Roomba, while adding a negligible height.

4. Software Interface

4.1. Development Platforms

4.1.1. Simulation Environment Design

The simulation of the Roomba multi-robot system will be created in the Unity 2021.3.8f1 game engine. Assets and game objects will be added into the scene to represent an environment similar to the real-life workspace in which the system will be tested. Physical characteristics such as device dimensions, rigidity, speed, color, and material can all be set and adjusted in the Unity scene design.

4.1.2. Simulation Script Design

All scripts used for development in Unity will be written in the C# programming language. Edits will be made using the Microsoft Visual Studio 2019 IDE and scripts will recompile in Unity automatically. The functionality of game objects in a scene will depend on the scripts in which they are referenced. Initial values of game objects such as speed and 3D coordinates will be set in a C# script. The pathfinding algorithm will be designed in a C# script and programmed onto the Roomba game objects for simulation.

4.1.3. Base Station Operations Program

The base station operations program will be programmed in C++ with required libraries to run on Windows 10. This software will handle calculations for localization, pathfinding algorithms, and determination of commands given to each Roomba. This program will interface with the onboard Bluetooth module for communication.

The localization subsystem will produce coordinates of each Roomba relative to the base station, where the base station is at the origin, and each Roomba is at a cartesian grid with a spacing of ¼ the width of a Roomba.

The pathfinding algorithm will derive commands for each Roomba in terms of move forward x grid squares, turn left, turn right, or stop. These commands will have specific sets of movements assigned to them for the Roomba to execute.

The Roomba will send obstacle reports when it detects an obstacle in the form of “Object detected x cm in front of robot”. The robot will stop and wait for further commands from the pathfinding algorithm.

5. Electrical Interface

5.1. Primary Input Power

The input power will come directly from the Roomba's Battery. The Roomba has a 14.4 V DC battery which will be used to support two DC-DC converters that will have an output voltage of 3.3 V at 1A and 5 V at 750 mA. The 7 pin mini-Din connector used to interface with the Roomba has 2 positive voltage pins connected to the Roomba battery and 2 ground pins which will be used to supply the DC-DC converters. The total power supplied from the battery will be approximately 7 watts.

5.2. Control and Localization System Power

From the DC-DC converter with 3.3 V output, the MCU and infrared distance sensors will be supplied power. The DC-DC converter with 5 V output will be used to supply the UWB module (tag)with power. Both the 3.3 V and 5 V converters will be used to supply the logic level shifters. The UAB modules (anchors) will each have their own wall warts producing 5v that will be permanently installed with the module in the workspace. The UWB module requires 5v with 500mA of current.

6. Communications / Device Interface Protocols

6.1. I2C

Communication between the MCU and the distance sensors will be conducted in the I2C protocol with the MCU in the master role and the sensors in the slave role on the same bus.

6.2. SPI

The SPI protocol will no longer be used.

6.3. UART

UART with a baud rate of 115200 will be the protocol used for the MCU to directly interface with the Roomba. The MCU uses UART at a logic level of 3.3 V, while the Roomba uses UART at a logic level of 5 V. To solve this, two 3.3 V to 5 V level shifters will be used: one between the MCU TX and Roomba RX, and the other between the MCU RX and Roomba TX.

The UART protocol with 115200 baud rate will be used to communicate between the MCU and the UWB tag module. The UWB will send position data to the MCU.

6.4. Bluetooth

The base station will send commands via bluetooth to the MCU. The MCU will acquire location data from the UWB module, and will in turn send location and sensor data to the base station.

6.5 UWB (Ultra Wide Band)

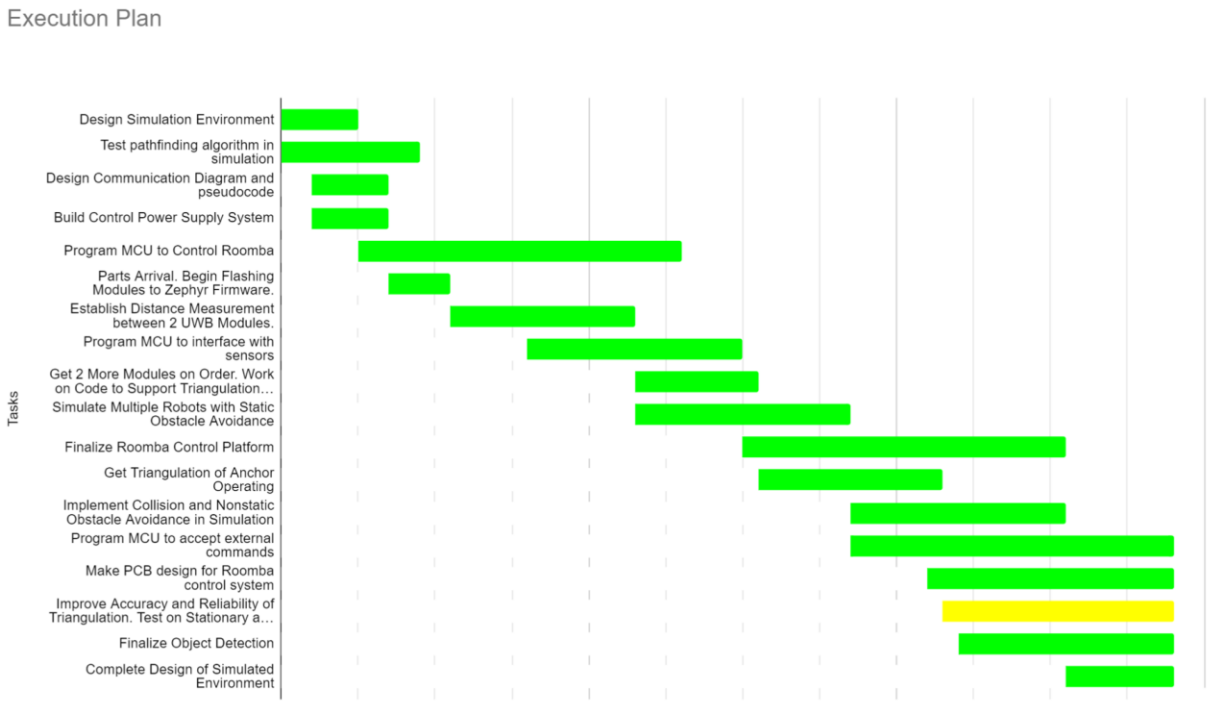
The UWB module on the Roomba will communicate with each anchor module to determine the distance between each. The UWB communication is used to determine the distance of the anchors from the Roomba.

Roomba Multi-Robot System

James Deere
Taylor Mosser
Todd Van Klaveren

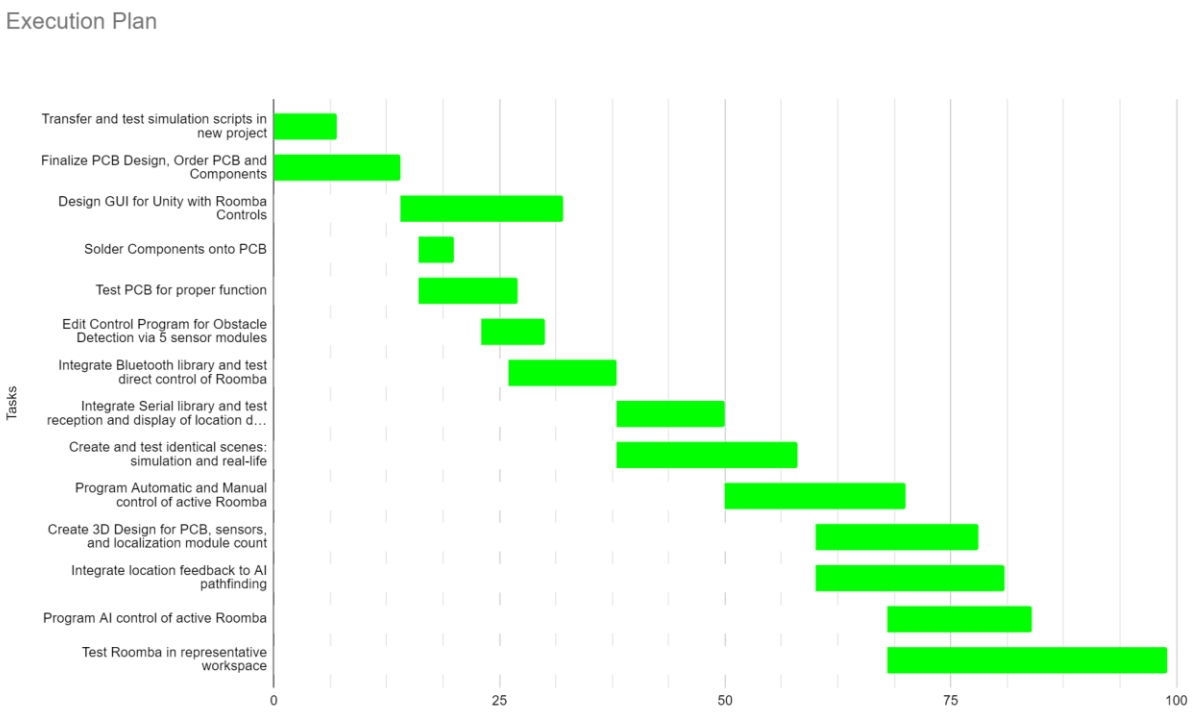
EXECUTION PLAN

1. 403 Execution Plan



Behind	Complete	In Progress	Not Started

2. 404 Execution Plan



Behind	Complete	In Progress	Not Started

Roomba Multi-Robot System

James Deere
Taylor Mosser
Todd Van Klaveren

VALIDATION PLAN

1. 403 Validation Plan

FSR	Test Name	Success Criteria	Methodology	Status	Responsible Engineer(s)
3.2.1.5	Command Response	Roomba can receive commands and respond accordingly.	A set of known commands will be flashed to the MCU through a USB connection from a laptop.	SUCCESS	James Deere
3.2.1.6	Object Detection	Roomba can detect an object in front of it..	Obstacle will be placed in front of Roomba.	SUCCESS	James Deere
3.2.1.6	Object Response	Roomba stops and sends alert of obstacle's presence.	Obstacle will be placed in Roomba's commanded path, Roomba should stop once obstacle is detected and send bluetooth signal that there is an obstacle.	SUCCESS	James Deere
3.2.3.3	Voltage Converters	The two converters used output 3.3V at 1 amp and 5V at 750 mA respectively	A multimeter is used to check the output voltage and current levels.	SUCCESS	James Deere
3.1	External Commands	MCU is able to accept external commands and move the Roomba appropriately	A bluetooth command terminal will be used to send commands to the MCU.	SUCCESS	James Deere
3.2.1.1	Multi-robot pathfinding	The A* algorithm can be used on two different Roombas until both robots have covered each of their assigned points.	Roomba game objects tested in simulation. Each takes the lowest-cost path and begins recalculation with each ending point that it reaches. Recalculation of path to new end point begins.	SUCCESS	Taylor Mosser
3.1.2.2	Obstacle and Collision Avoidance	The two Roombas avoid any obstacles and also avoid interference with each other.	Roombas avoid walls or obstacles during calculation of best path and while moving in the simulation. Roombas do not touch each other nor scrape against any obstacles.	SUCCESS	Taylor Mosser
3.2.1.2	Moving Obstacle Avoidance	The Roombas stop movement or recalculate paths when a moving object enters the space.	A Roomba will pause operation or move in a different direction to avoid collision with a moving obstacle, such as a person, in simulation.	SUCCESS	Taylor Mosser
3.2.3.6	Data Output	UWB module outputs location data to UART communication bus.	Once location data is calculated, the tag module must output the calculated location data once per second over UART. This will be connected to the Roomba Control System in the future.	SUCCESS	TVK
4.1.1, 3.2.4.4	UWB Module Communication	Base station able to issue commands to UWB module	Base station must command anchor units to initiate ranging pings. UWB must send a ranging ping on command from the base station and await a reply. Commands must be executed faithfully 95% or better.	Success, modified. Anchors all respond appropriately to a tag requesting distance. Working at 0.25 second per request.	TVK

3.2.1.6	2 Point Ranging	UWB module can determine range between	Base station will use anchor UWB module to send a ping and await a response. Tag UWB module will respond accordingly. Base station must use the time between send and receive to determine the distance between modules and output it to Bluetooth. Accuracy must be within 13” at 15 ft.	Success, Accuracy is +- 5 inches. Recommend averaging to reduce drift.	TVK
3.2.1.6 (Full implementation)	Triangulation	UWB modules all output ranges, and the base station performs triangulation.	Base station must perform 2 way ranging for each anchor to the tag module. Base station must output coordinates of the tag module over Bluetooth. Accuracy must be within 13”.	Success, Tag requests ranges from anchor and performs trilateration math to find position.	TVK

2. 404 Validation Plan

FSR	Test Name	Success Criteria	Methodology	Status	Responsible Engineer(s)
3.2.1.6	Object Detection	Roomba can detect an object in front of it (PCB and 5 sensor configuration)..	Obstacles will be placed in front of Roomba at various sizes and angles to determine if there are blindspots.	SUCCESS	James Deere
3.2.3.3	Voltage Converters	The two converters used on the PCB output 3.3V at 1 amp and 5V at 750 mA respectively	A multimeter and E-load is used to check the output voltage and current levels.	SUCCESS	James Deere
4.1.3	Mounting	The PCB, UWB module, and sensor modules shall be mounted onto the Roomba in a sleek and elegant way, allowing for no blindspots.	Mount adds less than 50% to the height of Roomba and allows for no blindspots.	SUCCESS	James Deere, Todd Van Klaveren
3.2.1.5	Command Response	Roomba can receive commands from the base station and respond accordingly.	A set of known commands will be flashed to the MCU from the base station via Bluetooth	SUCCESS	Team
3.2.1.6	Object Response	Roomba stops and sends an alert of the obstacle's presence to the base station.	An obstacle will be placed in Roomba's commanded path, Roomba should stop once obstacle is detected and send bluetooth signal alerting the base station that there is an obstacle.	SUCCESS	Team
3.2.1.7	Localization	Localization system reports Roomba location to Base Station	Base Station Application will output location data. Data will be compared to measurements of Roomba. Roomba must be within 13" circle of reported.	SUCCESS	Taylor Mosser, Todd Van Klaveren
3.2.1.5	Roomba Control	Base station manual control is able to issue commands to Roombas individually.	Roomba must execute commands being sent from base station application inputs made using the GUI.	SUCCESS	Team
3.2.1.1	AI Control	Pathfinding AI is able to issue commands to Roombas individually.	Pathfinding script can be used in multiple scenes without interference. Commands to real-life Roombas can be given through the use of Unity GUI.	SUCCESS	Taylor Mosser
3.2.1.1, 3.2.1.2, 3.2.1.6, 3.2.1.7, 3.2.4.4	In use Testing	Roomba navigates the test environment cleanly (no bumps) and updates path according to discovered obstacles.	A successful test will have no collisions between Roombas, obstacles, or room boundaries. The roomba will report any found obstacles to the base station. The pathfinding algorithm will adjust and execute an appropriate path.	SUCCESS	Team

Performance on Execution Plan

All tasks in the execution plan were completed on schedule or ahead of schedule. The original execution plan was edited to include additional tasks for each subsystem depending on the needs of the sponsor. All requirements for the design progress in both 403 and 404 have been met.

Performance on Validation Plan

Each characteristic and challenge listed in the validation plan has been tested various times to ensure complete functionality and success. The validation plan was edited to remove or add tasks according to the requirements of the sponsor and the overall project.

Roomba Multi-Robot System

James Deere
Taylor Mosser
Todd Van Klaveren

Subsystem Reports

SUBSYSTEM REPORTS
FOR
Roomba Multi-Robot System

TEAM ROOMBOTICS

APPROVED BY:

_____
Project Leader Date

_____
John Lusher, P.E. Date

_____
T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
0.0	11/21/22	Taylor Mosser		First Draft

Table of Contents

List of Tables

List of Figures

1. Introduction

The Roomba Multi-Robot System addresses the navigational and efficiency issues that arise when using current iRobot Roomba devices. The system tracks the location of each Roomba in use and its proximity to obstacles in the environment. Based on this data, a pathfinding algorithm is implemented to determine the lowest cost path each Roomba should take to reach each endpoint in the space. There are three different subsystems in the overall design: Roomba controls, localization, and pathfinding. The purpose and implementation details for each subsystem can be found in the FSR and ICD. Key tasks for designing and testing each subsystem can be found in the execution and validation plans, as well as the status of these tasks.

2. Roomba Controls Subsystem Report

2.1 Roomba Control Subsystem Introduction

The Roomba Control subsystem provides a platform for the Roomba to receive external commands, react accordingly, detect and avoid obstacles, and send an alert when obstacles are present. The subsystem is fully powered by the Roomba's battery via two buck converters. Validation was performed on the buck converters, infrared distance sensors, bluetooth command acceptance, and microcontroller's ability to control the Roomba.

2.2 Roomba Control Subsystem Details

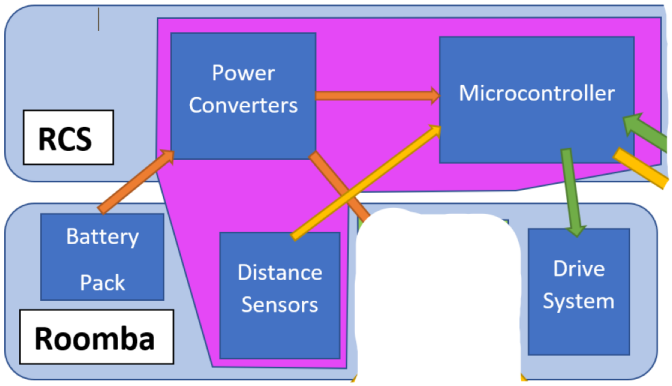


Figure 1: Diagram of Roomba Control Subsystem

The Roomba Control subsystem starts with the Roomba's battery supply (which has a 14.4V output) being fed through two different buck converters: a 3.3V 1A output and a 5V 750mA output. The 3.3V 1A signal is used to power the microcontroller (ESP32), the distance sensors, and a 3.3V to 5V logic level shifter. The 5V 750mA signal is used to supply the logic level shifter and to power the ultra-wideband module that will be integrated in ECEN 404. The ESP32 controls the Roomba via UART serial communication. The level shifter is used to shift 3.3V UART logic (output by the ESP32) to 5V UART logic for the Roomba (a logic HIGH for the Roomba is only triggered by 5V). The infrared distance sensors are used to detect obstacles, stop the Roomba when an obstacle is within 6 inches, and send an alert when an obstacle has been static for 5 seconds. The UART serial signals required for different actions are predetermined by the iRobot Roomba Open Interface.

2.3 Roomba Control Subsystem Validation

Both the 3.3V and 5V buck converters were tested for output stability at various loads using an e-load to make sure that the converters can handle various loads that might be put on it.

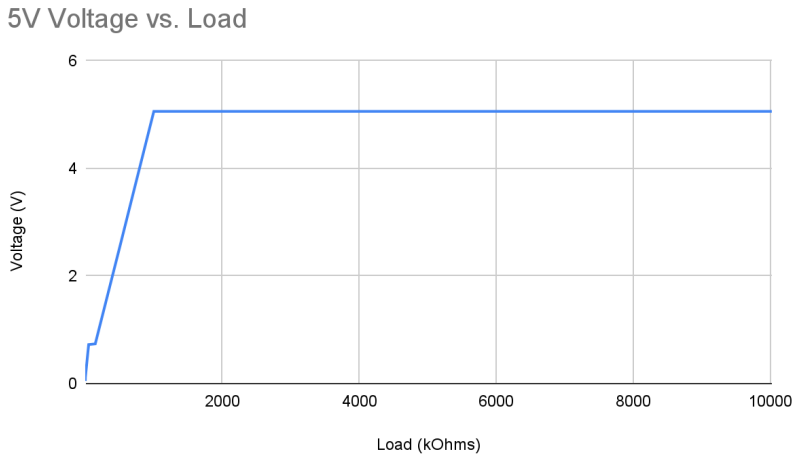


Figure 2: Voltage vs. Load Analysis for 5V Buck Converter

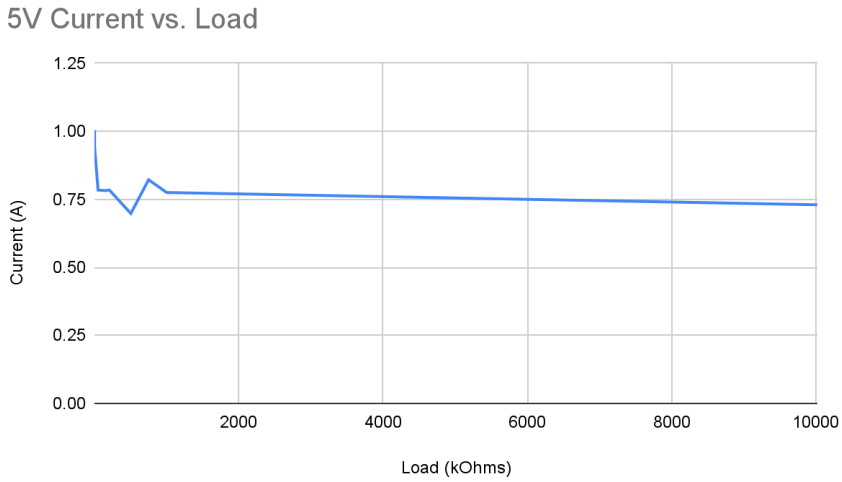


Figure 3: Current vs. Load Analysis for 5V Buck Converter

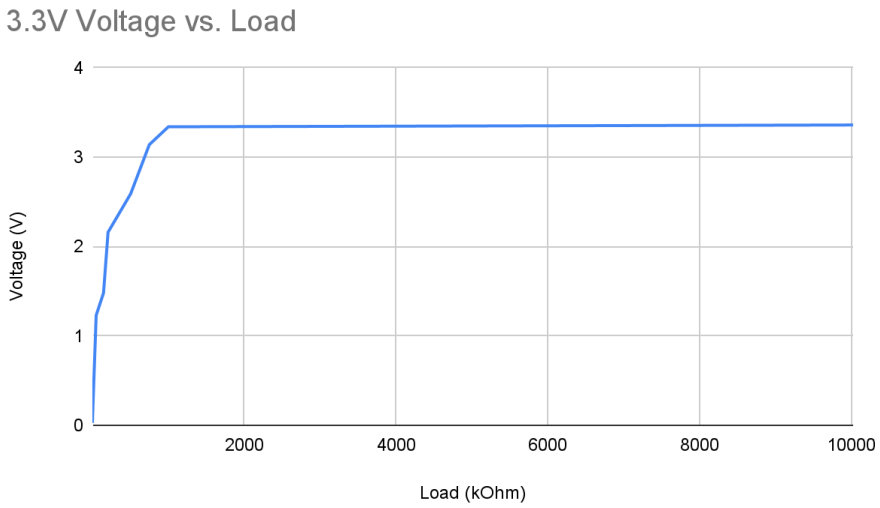


Figure 4: Voltage vs. Load Analysis for 3.3V Buck Converter

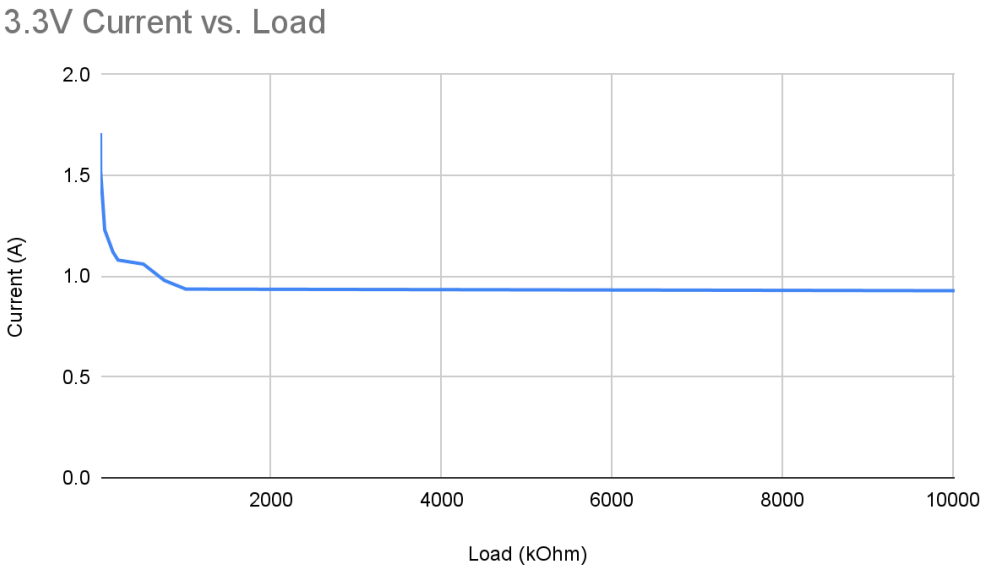


Figure 5: Current vs. Load Analysis for 3.3V Buck Converter

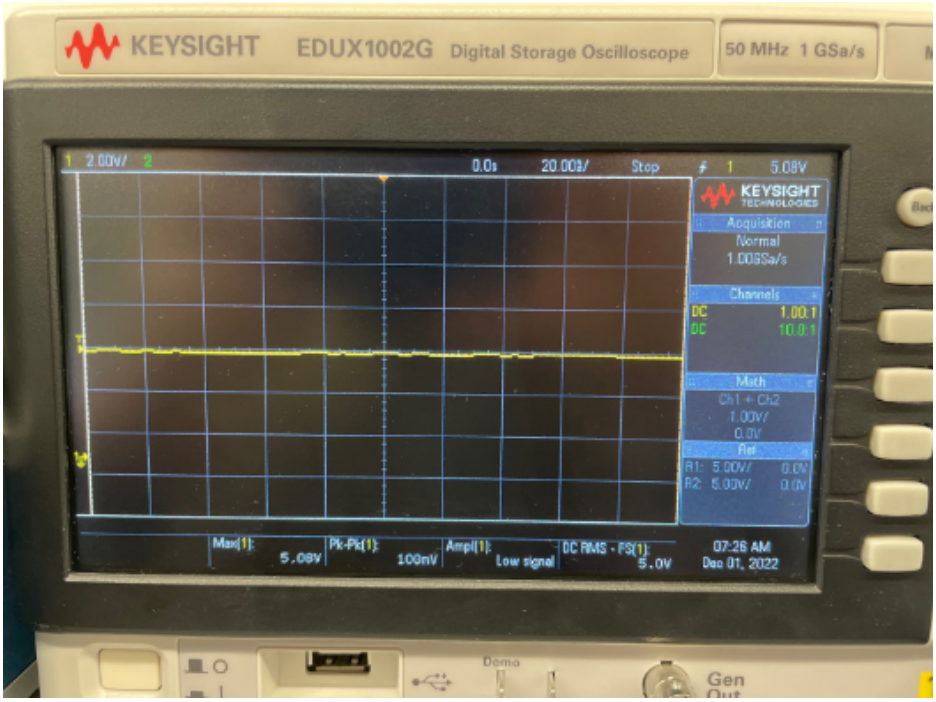


Figure 6: Oscilloscope Noise for 5V Buck Converter

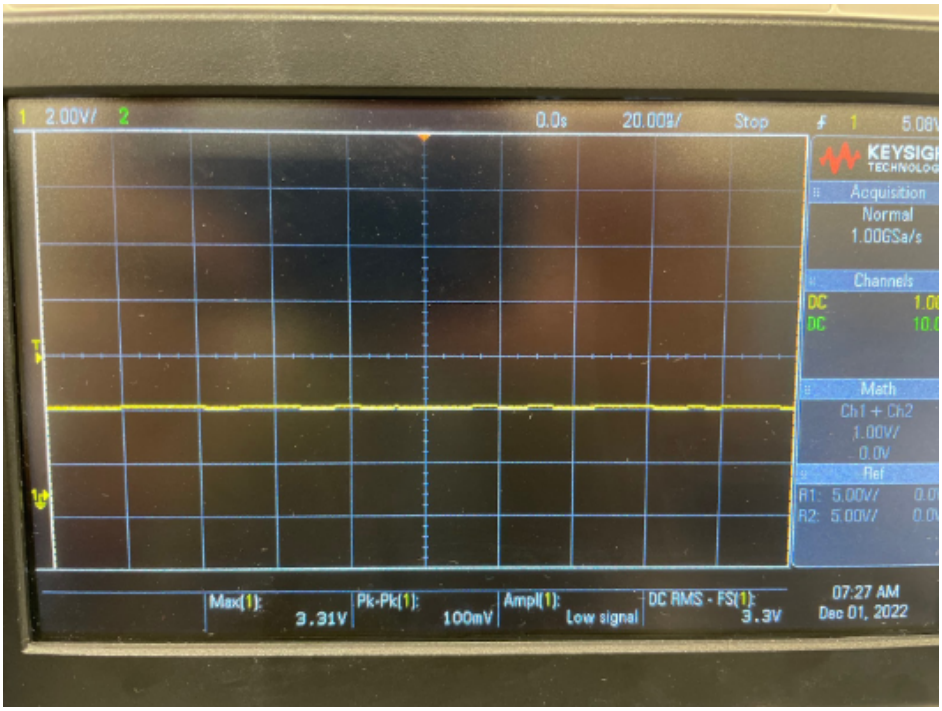


Figure 7: Oscilloscope Noise for 3.3V Buck Converter

As can be seen in figures 3 and 4, the 5V buck converter stabilizes quickly at the required 5V 750mA signal. The 3.3V buck converter stabilizes at a 3.3V 930mA signal. This is 70mA below the desired 1A signal, but it is still well within the satisfactory range to power

all required devices. In figures 6 and 7 oscilloscopes screens for both buck converters showing that the output signals have very low noise and negligible ripple. The buck converters have been shown to work on the circuit, powering the ESP32, distance sensor, and logic level shifter. Below, the current loads are shown for each DC converter. Required loads for the buck converters are shown in table 1 below.

Component	3.3V Buck Converter	5V Buck Converter
ESP32	700mA	
Distance Sensor (per unit)	6mA	
Level Shifter	< 1mA	< 1mA
Ultra-Wideband Module		500mA

Table 1: Current Load for Buck Converters

For object detection, the distance sensors were measured for accuracy at various lengths and on various surfaces. In the charts below, the raw distance and error are shown.

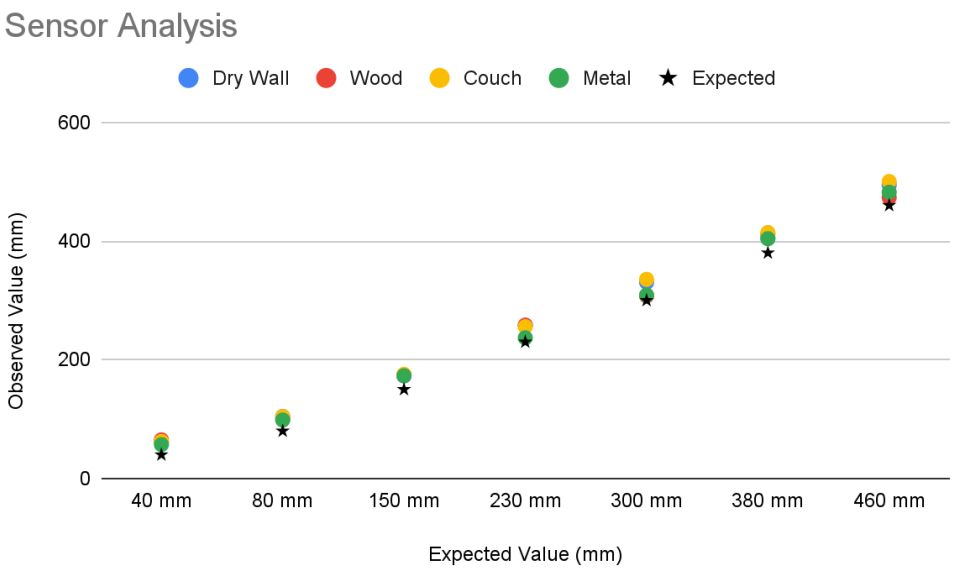


Figure 8: Distance Sensor Analysis

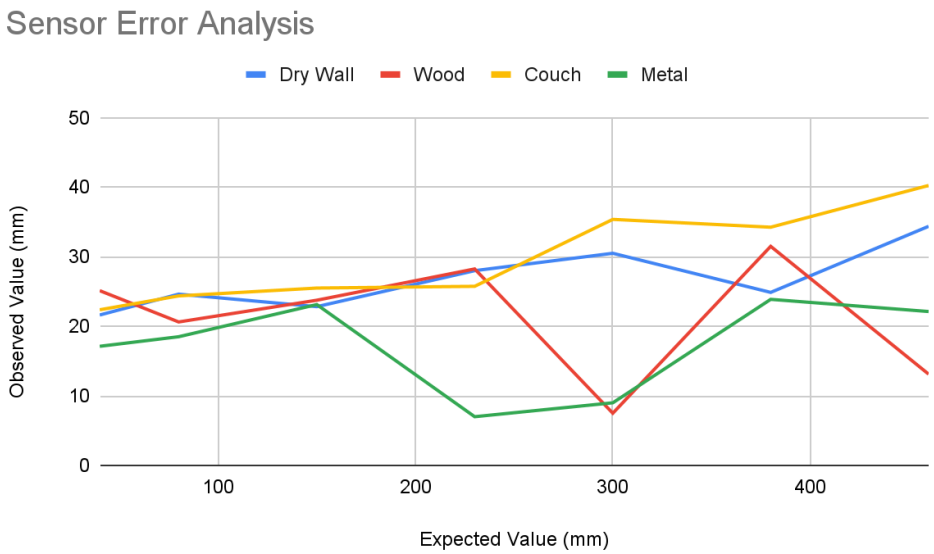


Figure 9: Distance Sensor Error Analysis

As can be seen, the sensors are relatively precise, just having an average error across distance and surface material of 23.8 mm. With this in mind, an additional 20-25 mm can be put on distances required to stop the Roomba and the error will be greatly attenuated. For validation of bluetooth external command reception, command response, and obstacle response, a bluetooth serial terminal (shown below) was used to connect to the ESP32, give commands, and receive warnings of obstacles in the way. The Roomba was monitored to ensure appropriate response to command and obstacles.

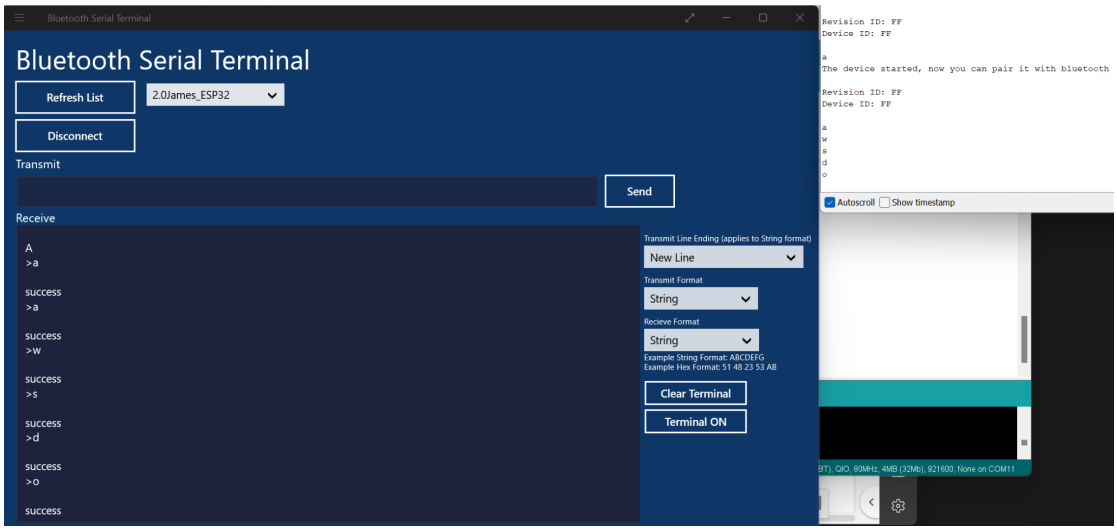


Figure 10: Image of Bluetooth Serial Terminal

```
Distance (mm): 2047.50
drive
Distance (mm): 2047.50
drive
Distance (mm): 2047.50
drive
Distance (mm): 129.25
stop
```

Figure 11: Image of Serial Monitor Response to an Approaching Obstacle

2.4 Roomba Control Subsystem Conclusion

The Roomba Control subsystem is capable of receiving external bluetooth commands, controlling the Roomba in accordance with commands given, and keeping the Roomba from hitting obstacles. In ECEN 404, the Roomba Control subsystem will be on a PCB, and be integrated to receive commands from the pathfinding algorithm. As of this report, only one distance sensor is being used, but a new mounting system will be designed for 5 sensors.

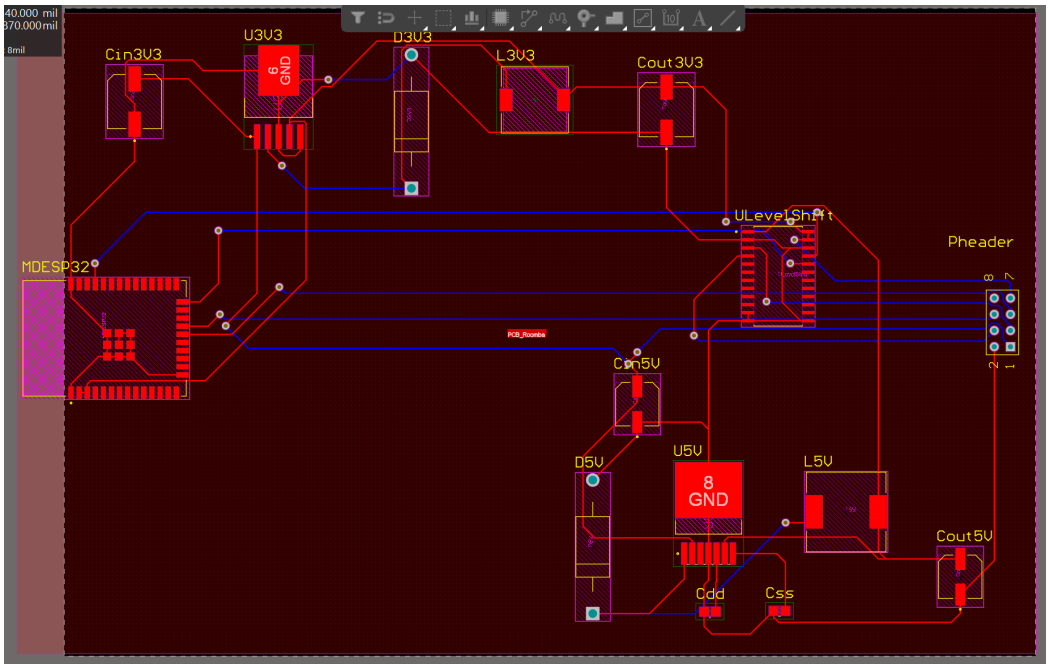


Figure 12: PCB Layout of Roomba Control Subsystem

3. Pathfinding Algorithm Subsystem

3.1 Pathfinding Algorithm Subsystem Introduction

The pathfinding algorithm is tested on the Roomba devices in a 3D simulation using Unity game engine and C# programming. The A* algorithm enables each Roomba to travel on the most efficient path from point to point, while avoiding collision with all obstacles.

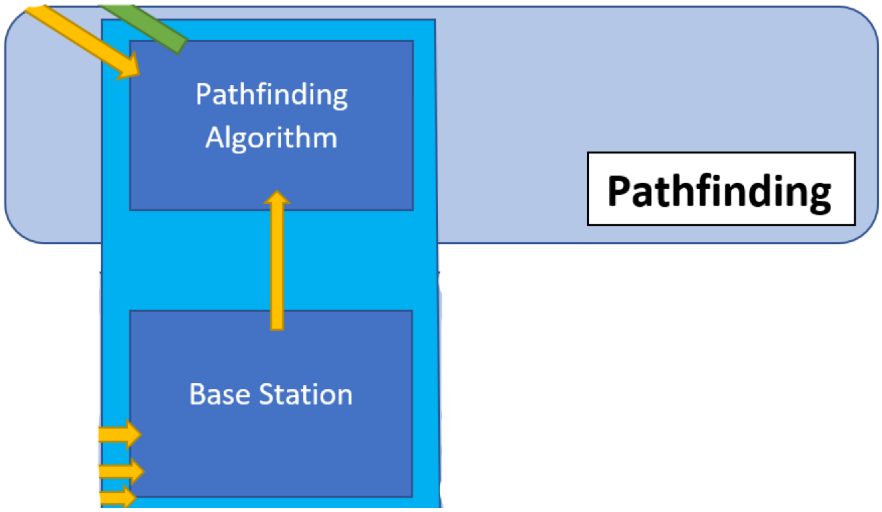


Figure 13: Diagram of Pathfinding Subsystem

3.2 Pathfinding Algorithm Subsystem Details

The Unity game engine is used to simulate a 3D environment that resembles the rectangular structure of WEB 156. The room consists of four connected hallways in which boxes and other obstacles are randomly placed. Objects such as walls, boxes, and the Roombas themselves are placed in the “Obstacles” layer so that when a path is calculated, there will not be a route traveling through them. The plane in which the room sits is placed in the “Ground” layer so that a graph of nodes or points can be formed on it.

The Roombas themselves are 3D game objects that operate independently of each other. Each one has a controller component, a seeker component, and an A* pathfinding script attached to its characteristics. The controller component enables the Roomba to move once a path has been calculated towards a target point. The seeker component performs the job of searching out all possible paths towards a target point and choosing the one of lowest cost. Under the A* pathfinding script, the user can edit the following public variables during simulation: speed, re-pathing rate, and distance to next waypoint. Private variables such as initial and additional target points are controlled through editing the script directly. If the user wishes to change or add target points for the Roombas to travel to, they can simply add another conditional statement for the desired point to the script itself.

Obstacle detection is handled by placing colliders on static objects and adding nav meshes to dynamic obstacles. Both types of components serve the purpose of creating an invisible barrier around an obstacle so that they will not collide with another. The dimensions of the barriers can be changed based on detection capabilities of real-life Roombas.

When the simulation begins, the grid graph of the plane is displayed and shows which areas are considered traversable by the Roombas. This graph is updated at the beginning of the simulation to account for any changes in the environment. A smooth path from the starting location to the first target point is displayed on the plane and the Roomba moves along the route until reaching the destination. If two or more Roombas are traveling on the same path through a narrow space, one will pass through while the others stand by. Once the first Roomba has passed, the others will proceed after it.

The path calculation is updated when an obstacle interferes with the current path. If the target point is out of range, the Roomba will simply travel to the closest coordinates that are in range. An error message appears in the log during game mode if path calculation is unsuccessful.

3.3 Pathfinding Algorithm Subsystem Validation

The subsystem successfully passed all validation tests to meet the requirements of the design. The following paragraphs contain brief descriptions of each key task.

Mapping of the Environment: The entire workspace can be mapped to determine which areas are traversable and which areas are considered unwalkable. Any object in the “Obstacles” layer is shown to be in the unwalkable areas. The grid graph of the entire room updates automatically as the Roombas travel across the space.

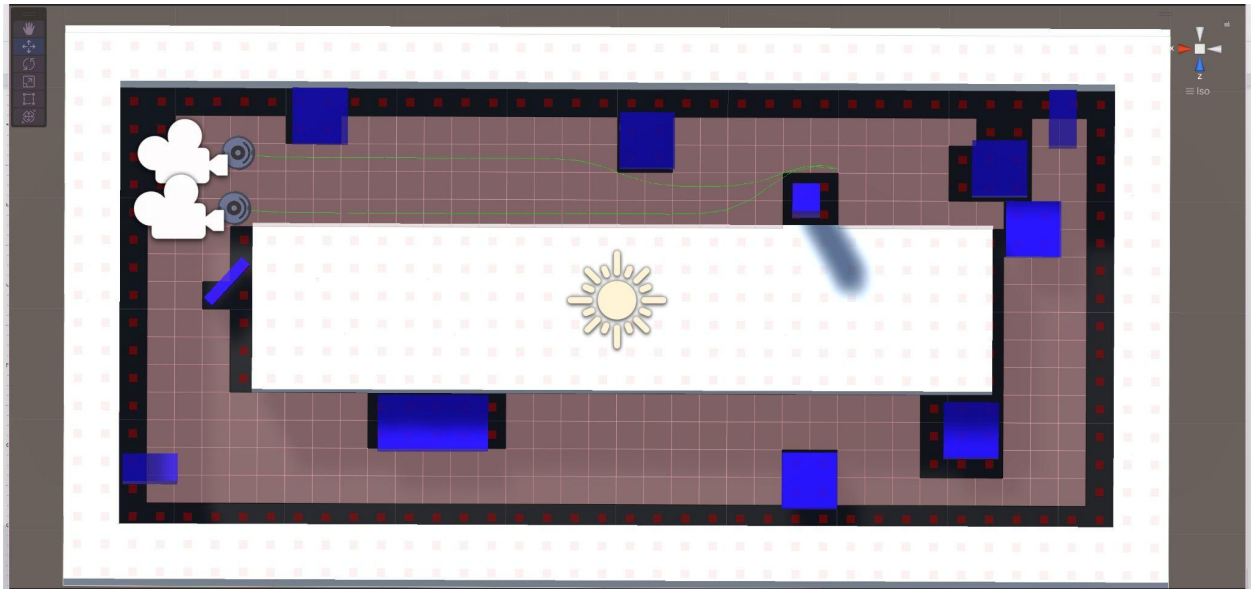


Figure 14: Path Calculation and Environment Mapping in Simulation

Single-target Pathing: This involves implementation of the A* algorithm to determine the optimal path from one point to another. The path is of lowest cost and is determined after evaluating all possible paths. The Roombas proceed on path once calculation is complete.

Static Obstacle Avoidance: Roombas do not collide with or calculate paths through obstacles in the environment.

Dynamic Obstacle Avoidance: Roombas cease movement or move around an object that is moving through the space. The Roombas avoid collision with each other, even if traveling on the same path or to the same target point.

Multi-target Pathing: Path calculation can be repeated for multiple points. Roomba can travel to a new target point after reaching an initial target point.

All tasks were completed and implemented into the simulation. The most challenging task proved to be the last, despite the simplicity of repeating the path calculation process for a single target. Currently, the target points are private variables in which a user would have to manually edit within the pathing script. Although the calculation of the paths to these points is controlled through conditional statements, a loop through a list of points may prove more effective in future implementation.

3.4 Pathfinding Algorithm Subsystem Conclusion

The A* pathfinding algorithm was successfully tested by simulating the Roomba devices in a 3D environment. The controls subsystem is represented by the collider and nav mesh components for detecting and avoiding obstacles. The seeker component and generated grid graph performs the duties of the localization subsystem by communicating the current location of each Roomba.

Figure 15: Localization - Current Position Data of Roomba in Simulation

For integration in ECEN 404, the process for implementing the algorithm onto the Roombas will be different, but Unity software will still be used. There will need to exist a

user interface in which the simulated algorithm can be transferred into use on the devices directly.

4. Localization Subsystem

4.1 Localization Subsystem Introduction

The localization subsystem is tasked with determining the position of a mountable “tag” relative to multiple anchors located around the workspace, and report this information to the Roomba Control Subsystem.

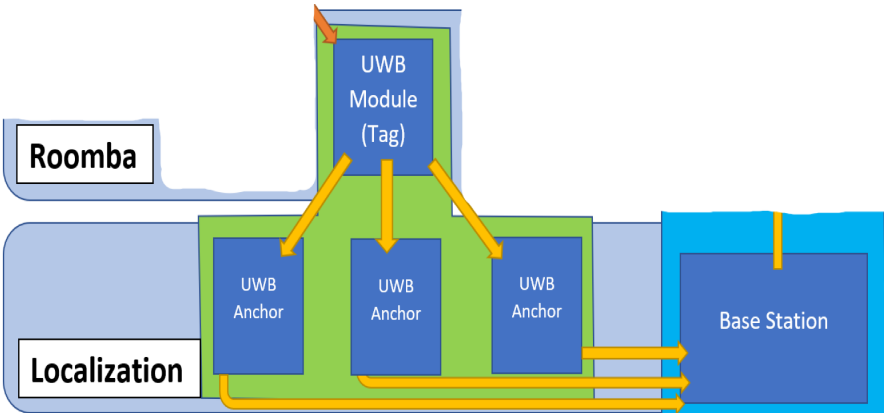


Figure 16: Localization Subsystem Block Diagram

4.2 Localization Subsystem Theory of operation

The localization subsystem consists of 3 anchor units and one tag unit. In the future, each Roomba deployed will have a tag unit attached to it. In addition, more anchors can be added if needed to increase coverage accuracy. The subsystem uses the UWB module on the tag to communicate with the UWB modules on the anchor in a way that allows for a determination of the distance between the two units. The tag does this for each of the anchors in the area (currently 3). The tag then performs trilateration to determine its position in relation to the position of the anchors and reports this to the base station through the Roomba Control Subsystem.

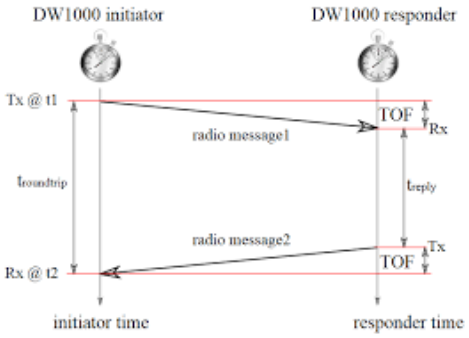


Figure 17: Two way Ranging using Time of Flight.

The distance determination is done by using a method known as time of flight. The tag unit will send out a request that will be received from each anchor that is listening. The anchor will wait for a given time period and then send a response with its id code. The tag can compute the distance from the anchor by taking the time between sending and receiving the message, subtracting the known wait period, dividing the remaining time by 2 and multiplying by the speed of light (as the signal will propagate at that speed in the air. This distance is found for each anchor response.

Once the distance to each anchor is found, the distance is put into a running average. This step is done to account for jitter in the position / distance reported. During testing of the distance measurement, I have found that the distance reported between the tag and anchor was prone to moving by about 3 - 5cm each time it updates. This jitter is worse when there is not a Line of Sight between the tag and the anchor. The running average reduces this jitter and tends to improve the accuracy. After the distance is averaged for each anchor, the tag attempts to perform trilateration using two different methods.

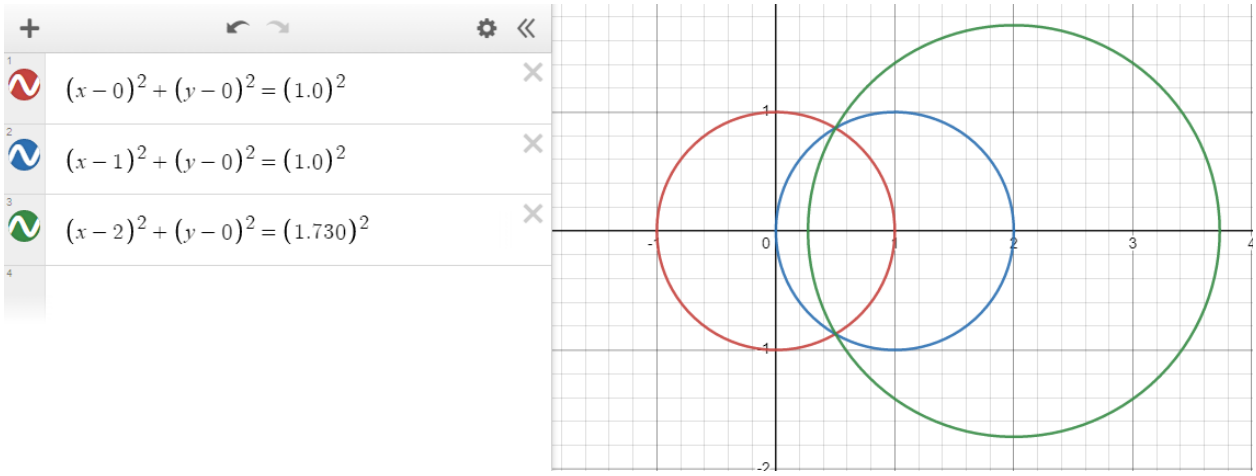


Figure 18: Perfect Trilateration with 3 points

The first method is mathematically perfect trilateration. This is simply the point where three circles intersect. The circles are of radius r where r is the distance from the respective anchor, and the center of the circle is at the known position of the anchor. The math behind finding the position x and y of the tag is simply the solving of three equations in the form of $(x - x1)^2 + (y - y1)^2 = d1^2$ where x1 and y1 are the known position of the anchor and d1 is the distance from tag to anchor. When solved for x and y, the position of the tag is found in the x,y plane. This method works if the anchors are extremely accurate and the circles do actually intersect at the same point.

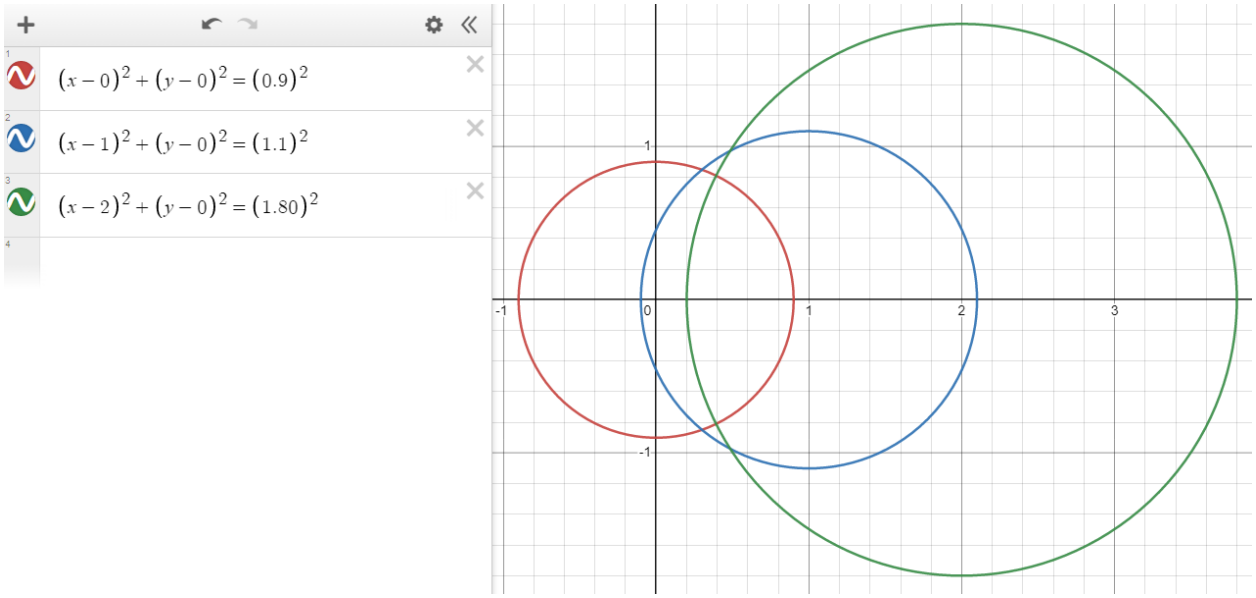


Figure 19: Imperfect Trilateration of 3 points

The second method of trilateration is one devised by myself. It is based on the same idea of intersection of circles, but it calculates the intersection of pairs of circles and then averages the calculated position from each. For each pair of anchors, such as 1 and 2, the calculation is similar to perfect triangulation in that is solves for x and y from the combination of the equation of a circle. It is then repeated for anchors 2 and 3 as a pair and anchors 1 and 3 as a pair.

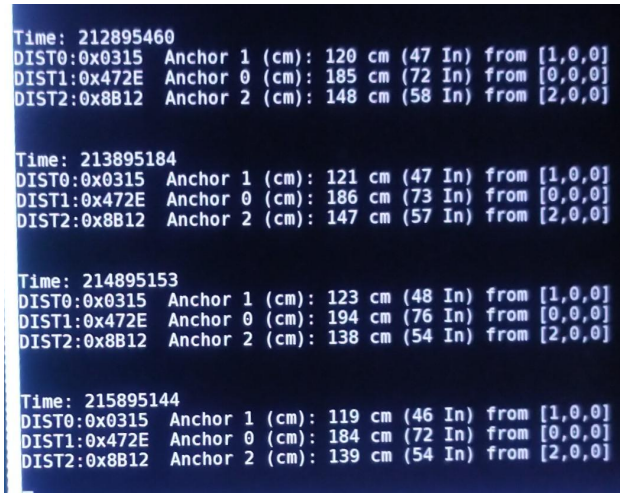


Figure 20: Serial Console output of Anchor Distance Data

After the location has been calculated by the tag module, the module then outputs the position to the Roomba Control Subsytem via UART. The Roomba Control Subsytem passes the data to the base station to update the position of the Roomba in the simulation environment. This is how we integrate the subsystem in 404. For testing, I have run the

UART output to a serial monitor which allows for viewing of the position data in a more readable format for testing. IT is seen above.

4.3 Localization Subsystem Validation

The validation for the subsystem came in two major parts. The first is the implementation and testing of two way ranging between a single anchor and tag unit. The testing simply checked the output on the Serial Console for the distance reported. It was found that the distance was within 7 cm of exact at distances from 3 ft to 20ft. This was tested outside in my driveway with clear line of sight. This proved the implementation of the distance measurement and the accuracy of the UWB Time of Flight methodology.

The second part of validation is the determination of an x y coordinate from 3 anchors and one tag. The system was first tested with perfect trilateration, but it was often unable to determine position. The imperfect method was able to reliably find a position of the tag with good accuracy. The tested accuracy was found to be 15 cm on average at 13 ft away. This test was done both inside and outside, both with clear line of sight between the anchors and tag.

4.4 Localization Subsystem Conclusion

The localization subsystem is able to determine the location of a tag module to well within the size of one Roomba. This will be able to keep track of Roombas as necessary and communicate their positions to the base station for tracking. The localization system is ready for integration with the Roomba Control Subsystem and base station.

Roomba Multi-Robot System

James Deere, Taylor Mosser, Todd Van Klaveren

Completed System Design (Final Report)

REVISION – 1.0
20 April 2023

COMPLETED SYSTEM DESIGN
FOR
Roomba Multi-Robot System

TEAM ROOMBOTICS

APPROVED BY:

_____
Project Leader Date

_____
Prof. Lusher Date

_____
Max Lesser Date

Change Record

Rev.	Date	Originator	Approvals	Description
0.0	04/20/2023	Taylor Mosser		Final Report

Table of Contents

List of Tables

List of Figures

1. Overview

The Roomba Multi-Robot System is an effective design to add intelligent pathing and control to an ordinary Roomba, allowing it to operate efficiently and quickly in a variety of environments and under a varying set of goals. The system consists of three subsystems which allow the system to operate as intended: Control System, Pathfinding, and Localization. When fully integrated, each system allowed the system to receive data, make path calculations, execute movements in the environment, and confirm the action was performed.

The Roomba was equipped with a set of 5 distance sensors, which would detect any objects in front of the robot and report them via the control system to the base station. The Localization system in turn determined and reported the location of each individual Roomba to the base station. The Pathfinding system, acting as the base station, received the data in from the Control System. Upon receiving the data, the Pathfinding system devised a path for the given task, and sent out a set of commands for the Roomba to execute; it would then check the data to ensure the movements were correctly carried out.

Once this system was devised, integrated, and implemented, we validated each of the abilities and functions of the system as per the Validation Chart we created.

2. Design and Execution

2.1 Project Design

2.1.1 On-Roomba Hardware

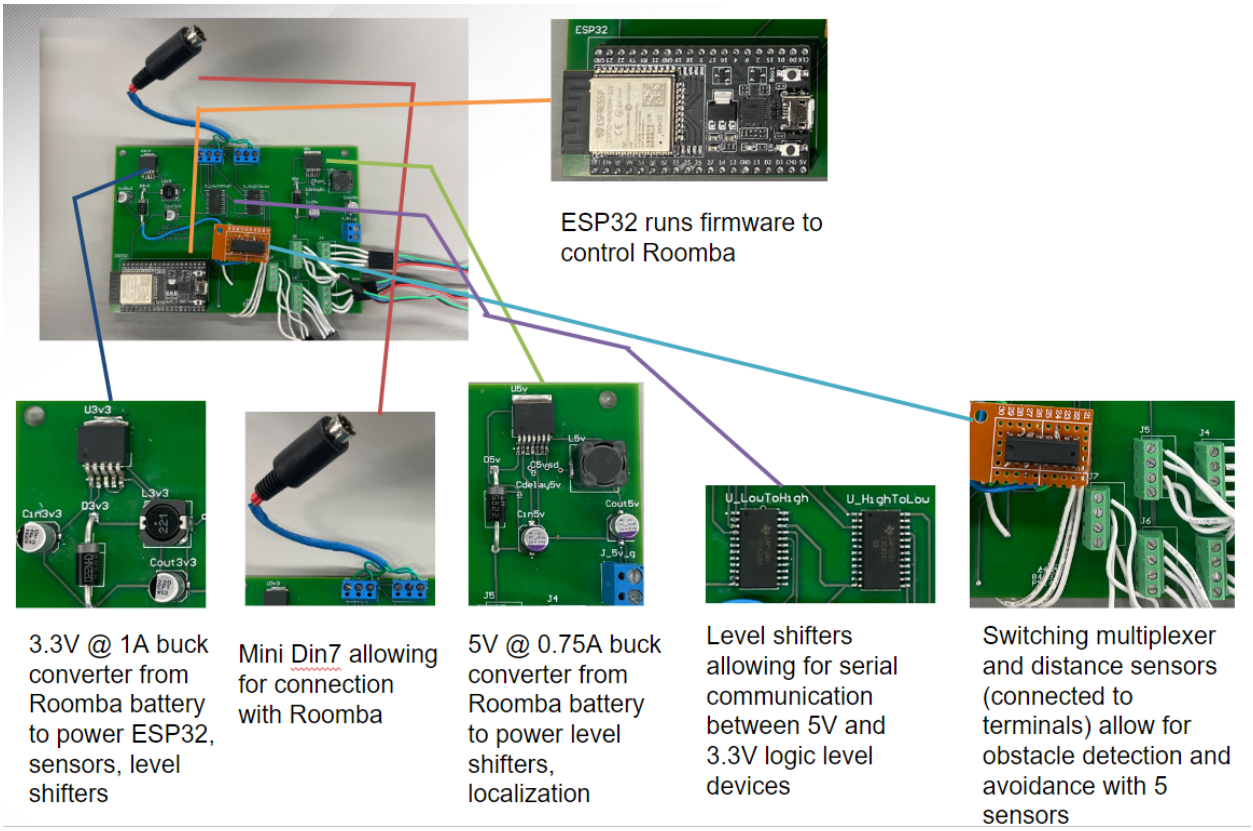


Figure 1: PCB Designed and Overview of Operation

As seen in ***Figure 1***, the circuit used in the Roomba control subsystem in 403 was made into a PCB for integration for 404. The only change was adding a multiplexer and 4 extra distance sensors. This was done to expand the area where obstacles can be detected. The multiplexer was added to make the sensors addressable on the I2C bus. These sensors allow the robot to fulfill validation point 3.2.1.6, which is shown in the “Validation” video, section Dynamic Obstacle Detection. The video is found on the project GitHub.

2.1.2 UI Design

For the purpose of integration, a user interface was designed in Unity in a separate project. The scene for this project is similar to the one used for simulation, but made much simpler so that it can represent the environment used in real life. In Unity, this scene measures 3m x 3m. There is no A* algorithm game object in this scene. Instead, all pathfinding is performed in a singular script called UpButton.cs. Sensor data from the Roomba and localization data

from the anchors is read directly into Unity, handled in the pathfinding script, and then commands are sent directly to the ESP32.

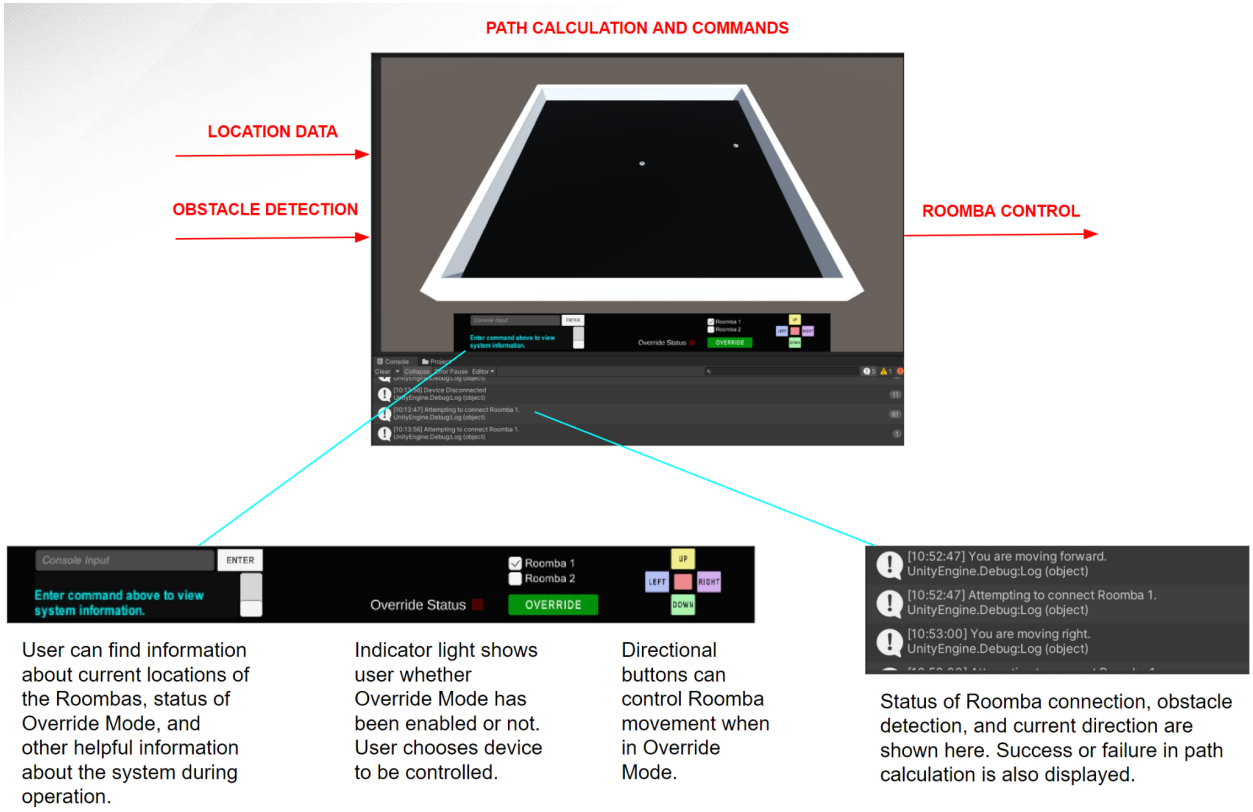


Figure 2: Integrated System Design in Unity Project

The key feature of the UI design is the game bar at the bottom of the screen. On the left of the game bar is a console in which the user can enter specific commands to receive information about the system during operation. These commands include:

- location: gives exact coordinates of Roomba from localization data
- override: shows status of Override Mode and which Roomba is being controlled
- reload: gameplay is restarted and all values are reinitialized
- help: displays a list of all commands that a user can enter.

It is important to note that information displayed in the game bar console is also displayed in the project console outside of the scene. The project console displays information such as connection status, raw sensor data, and raw localization data.

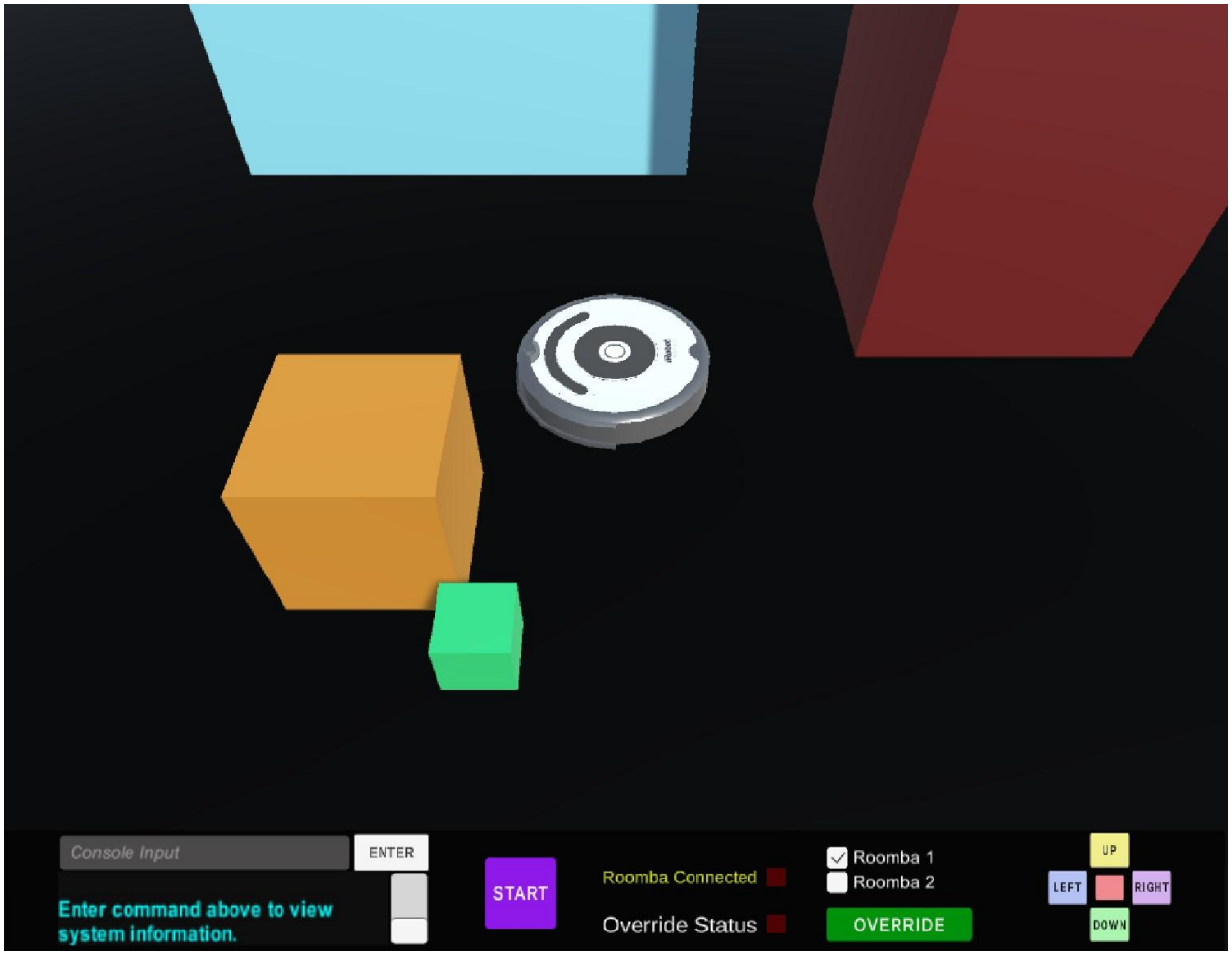


Figure 3: UI Design in Unity

Although gameplay begins after pressing the play button at the top of the project screen, pathfinding does not begin until the user presses “START” on the game bar. If no obstacles are detected, the Roomba will begin to travel in a direction based on current location. As soon as a Bluetooth connection has been established with the Roomba, the indicator light will turn green next to “Roomba Connected”.

The most important aspect of the game bar is the Override Mode feature. The user can select which Roomba they would like to control using the toggle. Once a Roomba has been selected, the user can press “OVERRIDE” to manually control the selected Roomba’s movement. The indicator light for “Override Status” will turn green when Override Mode is enabled. If the user presses “OVERRIDE” again to exit manual control, the indicator light will dim back to red and the Roomba will no longer receive direct commands from the UI. While in Override Mode, users can control the direction of Roomba with the buttons on the far right of the game bar. It is important to note that the Roomba does not travel backward when the “DOWN” button is pressed, rather, it makes two 90° turns and travels forward in the opposite direction.

The Roomba can transition smoothly between pathfinding and Override Mode easily. When “OVERRIDE” is pressed, the Roomba stops movement and waits for the user to send commands via the UI buttons. When exiting Override Mode, the Roomba begins to follow commands from the pathfinding script based on its current location. Override mode fulfills validation point 3.2.1.5, where the base station manually controls and has full command ability of the Roomba.

2.2 Validation and Results

All of the intended features of the UI functioned as intended. Below are examples of validation of the user console, Override Mode, and received localization and objection detection data for use in tracking and pathing the Roomba.

User Console: The following commands are entered into the user console and the output is shown in the images below. Commands can be added into the “Console Controller” script at any time to improve user experience.
override: informs user of Override Mode status

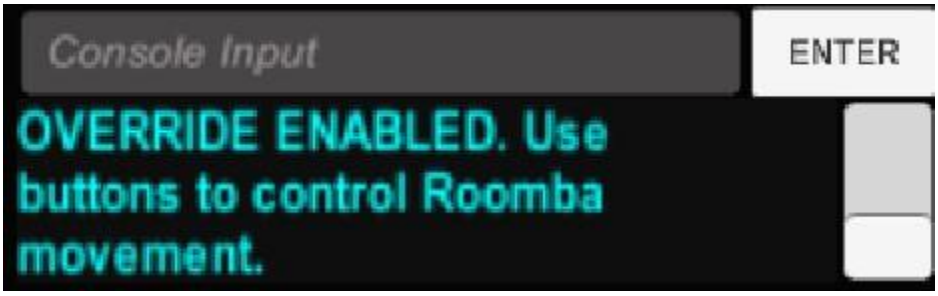


Figure 4: Output of “override” Command

help: gives the entire list of available commands

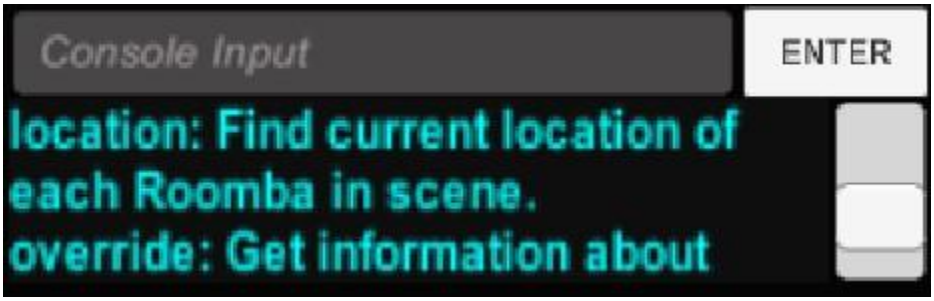


Figure 5: Output of “help” Command

Indicator Lights: When the Roomba is connected and also in Override Mode, both indicator lights will turn green.

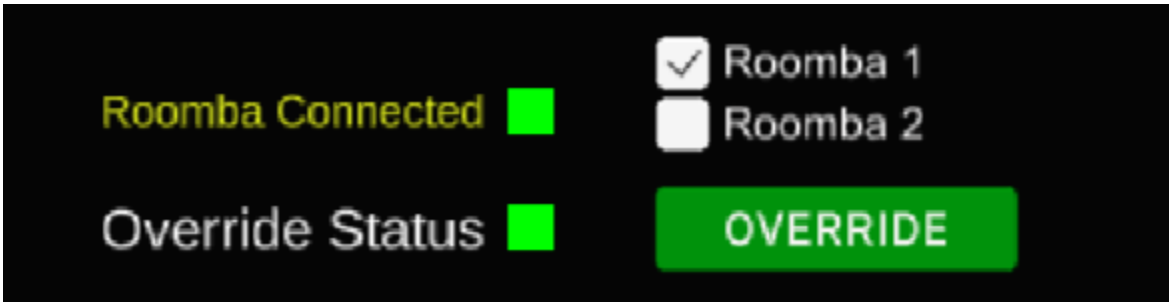


Figure 6: Indicator Lights for Bluetooth Connection and Override Mode

Sensor and Localization Data: Raw data from the five sensors and from the UWB anchors is received and read into the project console. Sensor data read in by is compared to the threshold set by a user; if the threshold is exceeded, a message is displayed indicating an obstacle's presence. This fulfills validation point 3.2.1.6. The location data is read in, and converted to numeric values, and placed in a publicly accessible container for each program to use. The display of these values fulfills validation point 3.2.1.7.

Figure 7: Sensor Data Read into Unity Project Console

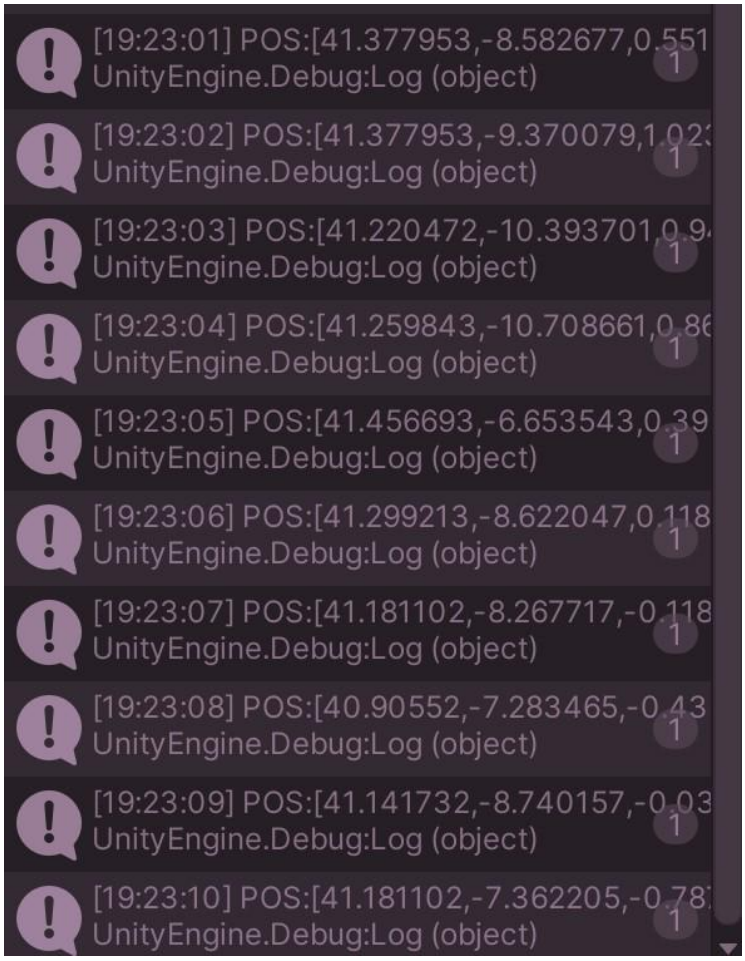


Figure 8: Location Data Read into Unity Project Console

	ESP32	IR Distance Sensor (5 used)	Ultra-Wideband Tag Module	Total Power Consumed
Power Consumed (mW)	1000	100	2500	3900

Table 1: Power Consumption

The power we are consuming from the Roomba battery is 3.9 watts. A Roomba battery supplies 48 watt-hours and the nominal battery life is 2.5 hours, leading to an average of 19.2 watts consumed by a normal Roomba. With our system consuming an additional 3.9 watts, the average battery life will go down to 2.078 hours, or about 83% of a normal Roomba run time per full charge. These converters were proven to fulfill validation point 3.2.3.3 in the charts found in the subsystem reports from 403.

As part of the validation process, a series of videos were made of the operation of the Roomba Multi-Robot System in each of the following modes: Room Cleaning, Static Pathfinding Execution, Dynamic Object Detection, and Override Mode. The video shows the system going through each task ascribed by validation points 3.2.1.1, 3.2.1.2, 3.2.1.6, 3.2.1.7, and 3.2.4.4. This video also shows the system operating as a whole, how it is intended to work per the design documents.

2.3 Conclusion

Our project was able to control a Roomba in various modes of operation including Room Cleaning, Direct Pathfinding, and Manual Override. The Room Cleaning mode instructs the Roomba to create a path back and forth across a room. This mode allows for multiple Roombas to be deployed, and they will each take a portion of the room in order to maximize efficiency. Direct Pathfinding allows the Roomba to implement the A* pathfinding to find the shortest route between two points and follow that path while checking its path for obstacles. In the event of an error state, the user can enter Manual Control mode using the UI of the base station. The commands in Manual Mode include cardinal directions, emergency stop, and resume original program. Also included in the UI is the output of data such as location and obstacle detection sensors to a console on the side.

The Roomba multi-robot system plans paths that are more strategic than the paths taken by individual Roombas. Some of the advantages of this system are

- Greater energy efficiency due to reduced amount of backtracking during cleaning.
- Faster cleaning rate due to pathing and ability to deploy multiple Roombas in the same room.
- Minimal cleaning overlap.
- Opportunities for delivery applications utilizing intelligent pathfinding in various environments.

Our system is just one application for exploring and making advancements in path planning, leading to a future in which there can be further dependency on robots to perform difficult tasks. Because our system relies simply on a UART command structure being sent via the control system, it could easily be adapted by attaching it to any robot platform that accepts commands by UART communication, and setting up the syntax for the specific application. The localization and pathfinding systems are designed to be scalable and transferable between implementations. The ideas of the project could be scaled up to perform parts delivery tasks in a machine shop or factory or could even be a part of an inventory management system for a warehousing company.

3. References

The video located at the following link is referenced in the document as the “Validation” video.

https://github.com/talimn/Roomba-MRS/blob/main/Capstone%20video%20location.txt