Beverage Service Robot

Gonzalo Tiscareno, Daniel Krygsman, Royce Shioi, Isabella Almaraz

University of California, Santa Cruz

Baskin School of Engineering

June 16, 2024

Stephen Petersen

Abstract

This project investigates a solution to restaurants that are understaffed by creating an autonomous robot that delivers drinks to tables on demand. The Covid-19 pandemic brought difficulty to those in the restaurant industry due to lack of business. This forced restaurants to take measures such as having less staff which can lead to a number of problems especially during peak rush hours. Our solution was to develop a fully autonomous robot that can bring drinks to tables on demand as well as dispense water when necessary. This would alleviate some of the pressures that come with serving especially during rush hours. The goal is to not only ensure better customer service but to also improve the working environment for the servers.

Table of Contents

Introduction **4**
System Design Overview **5**
Embedded Software Architecture **6**
Software Introduction 6
Sensor Nodes 8
URDF 11
ROS2 Control 14
Mapping Localization and Navigation 16
Software Conclusion 20
Mechanical Design and Implementation **21**
Drive Train 22
Computer Aided Design of Drive Train 23
Motor Selection 25
Prototyping 28
Prototype Testing 29
Motivation Behind Shell Design 30
Shell Computer Aided Design 31
Prototyping Shell 34
Final Shell 36
Sensor Mounts 38
User Interface Design **40**
Water Dispenser 41
Table Modules 44
Power **57**
Power Requirements 59
Battery Selection 60
Voltage Regulation 61
Power Distribution Board Design 62
Testing and Validation 64
Lessons Learned **67**
Next Steps **68**
Conclusion **69**

Introduction

The rise of automation and robotics has grown to benefit several industries including hospitality. There is a growing interest in using robotic technology to enhance customer service, improve efficiency, and reduce operating costs. An application of this involves a beverage-serving robot that is capable of bringing drinks to customers on demand as well as dispensing water. This aims to not only improve customer service but to reduce stress and costs. The robot uses sensors to navigate to a table once the customer requests its service. This proves to be most effective during peak restaurant hours when there is decreased service efficiency and longer wait times. With the addition of the robot handling drinks, waitstaff can focus on enhancing customer service. The primary goal of this project is to develop a prototype of the beverage serving robot to improve service delivery times and reduce the workload on staff.

The ongoing labor shortage across the restaurant industry in America has made it challenging for many establishments to adequately staff their serving teams, especially during peak dining hours. As servers rush around fulfilling duties like taking orders, delivering food, bussing tables, and providing checks, drink service often suffers. Servers simply do not have time to properly attend to drink refills and requests while handling all their other tasks. Some restaurants have investigated utilizing robot servers to handle certain basic tasks like delivering food and bussing tables. Still, nothing is stopping them from doing more complex tasks like serving beverages directly to tables. This allows the human servers to focus on providing good customer service while the robots assist with some of the routine work. At the same time, the restaurant sees increased profitability through a rise in beverage orders facilitated by the robots.

This report provides a comprehensive overview of the design and implementation process undertaken to realize the final beverage service robot. It begins with a system design overview

that outlines how the various subsystems interconnect and interface with the overall system. This complete design overview follows the block diagram presented in Figure 1. The system design overview section follows, detailing the “Embedded Software Architecture” which covers critical components like sensor nodes, universal robot description format (URDF), Robot Operating System 2 (ROS2) control, and mapping, localization, and navigation capabilities. The “Mechanical Design and Implementation” section delves into the drivetrain mechanics, use of computer-aided design (CAD), motor selection, shell design motivations, prototyping, final implemented designs, and sensor mounts. The “User Interface Design" discusses the water dispenser and table modules. Electronics of the simplex radio transmission between the robot and table modules are discussed in the “UI Electrical System" section. The “Power” section analyzes requirements, battery selection, voltage regulation, power distribution board design, testing, and validation to provide each electrical component with power. Finally, the report concludes by discussing lessons learned and next steps for the system.

System Design Overview

In designing the beverage service robot, we focused on creating a system that seamlessly handles customer requests and dispenses beverages autonomously. The customer module, powered by a 9V battery, includes an Arduino Nano and an NRF24L01 transmitter, allowing it to wirelessly send button press signals to the robot. In the robot, another Arduino Nano and NRF24L01 receiver work alongside a Raspberry Pi 4, which processes these signals and manages the robot's operations. We implemented ROS2 on the Raspberry Pi for software management and SLAM for real-time mapping and navigation. Sensors like the RP LiDAR and a depth camera provide environmental data via USB to the Raspberry Pi.

For motor control, we used an H-Bridge and drive motors, with feedback from motor encoders and hall sensors ensuring precise movement. The water dispenser module is powered by a rechargeable battery and controlled via a charging control PCB and a water pump motor. To manage power distribution efficiently, we designed a power management unit that supplies 12V high and low current rails and a 5V rail to the various components. This integration of wireless communication, autonomous navigation, and power management enables the robot to deliver beverages upon customer request efficiently.

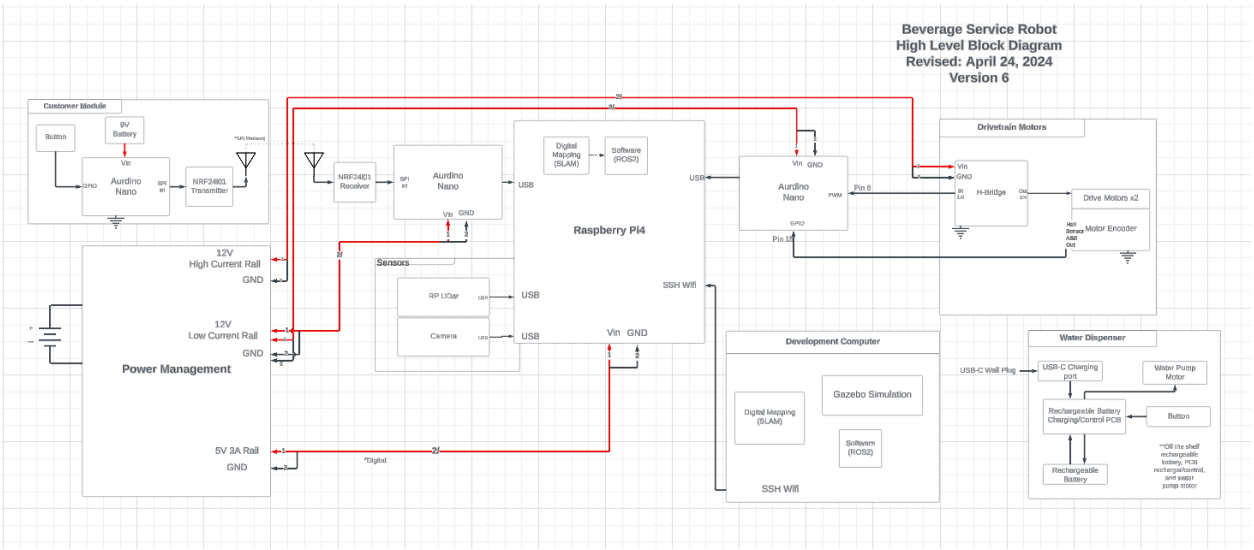


Figure 1. High Level Block Diagram

Embedded Software Architecture

Software Introduction

When researching means to achieve our project goals, specifically software systems, ROS or Robot Operating System is the clear answer. Contrary to its name, ROS is not an operating system, but a collection of open source packages, libraries, and standards which has found itself to be the current industry standard. The idea behind ROS is to create layers of

abstraction and separate a robot's parts into "nodes". A node could be anything from sensors and their specific driver, actuators and their interfaces, and anything in between. Nodes communicate via topics, services, and actions. The abstraction clears up a system by having any sub-node information invisible to other nodes and the system as a whole, this abstracts any useless information that might otherwise be distracting/confusing.

The beverage service robot and our project goals is the newest ROS2 distribution at the time of the start of our project, ROS2 Iron, running on two separate Linux machines running Ubuntu 22.04. The ROS system needed to achieve our goals

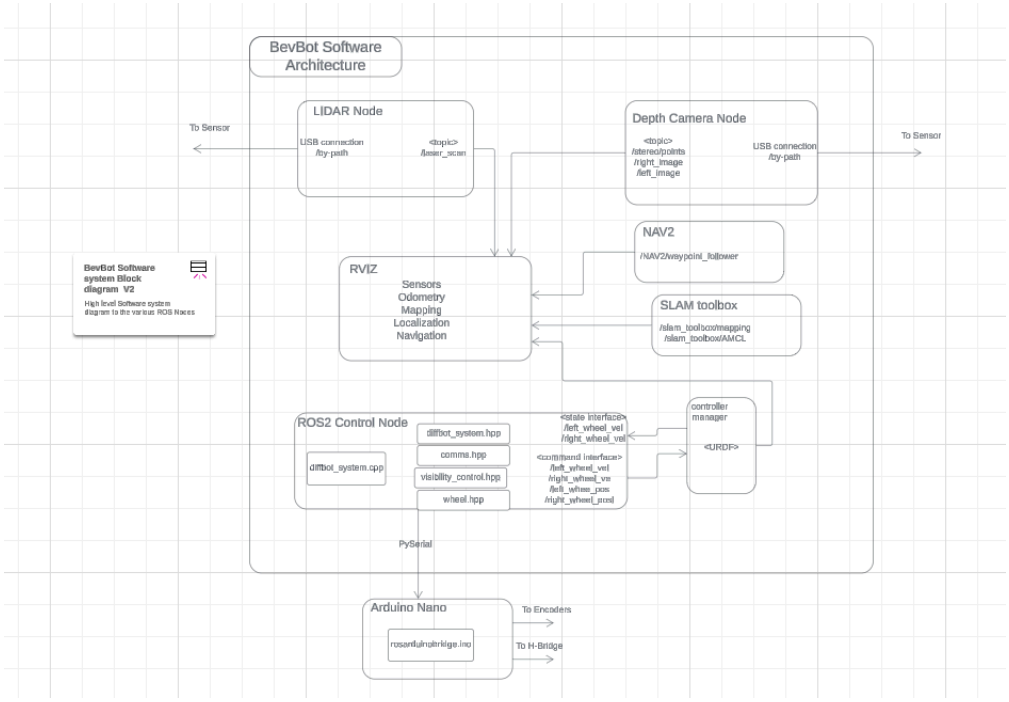


Figure 2. Software System Block Diagram

consists of a differential motor control node, a 360-degree lidar (light detection and ranging) node, a camera node, and our visualization and navigation nodes. Another large part of this ,which is not technically a ROS node, is our URDF or Unified Robot Description Format files.

One could imagine our system where the visualization and navigation nodes sit at the center with our lidar and camera nodes describing the world around it, and our motor control node being the means for it to move and navigate. The accuracy and usability of the ROS system experienced with this project have given us as a team a better understanding of why ROS is the leading industry standard in robotics.

Sensor Nodes

The accuracy of a robot is heavily dependent on how accurately it can observe the world around it. That being said, the budget is a large limiting factor in this project. The sensors we chose were Slamtec's RPLIDAR A1 and Luxonis' OAK-D-Lite stereo depth camera. The LIDAR consists of a rotating time of flight laser sensor which produces a 2d map of any obstructions in its plane of view. The OAK-D-Lite is an RGB camera and depth camera that can produce a 3d point cloud of the area it is in. The idea that emerged after testing and prototyping was to mount the lidar in front of the bot close to the floor to detect any obstructions on the floor such as bags or chairs, while the depth camera is higher up to detect people or tables. These sensors allow us an accurate view and versatility such as remote operation or teleoperation.

The Slamtec RPLIDAR is specifically designed as a plug-and-play sensor for the Raspberry Pi and has been consistently the sensor of choice for ROS robots. This 2d, horizontal plane scan of the room is ideal for room mapping robots because of its ability to simultaneously map all surrounding walls and obstacles at once because of its 8,000 samples per second. There is not much needed to configure the RPLIDAR other than

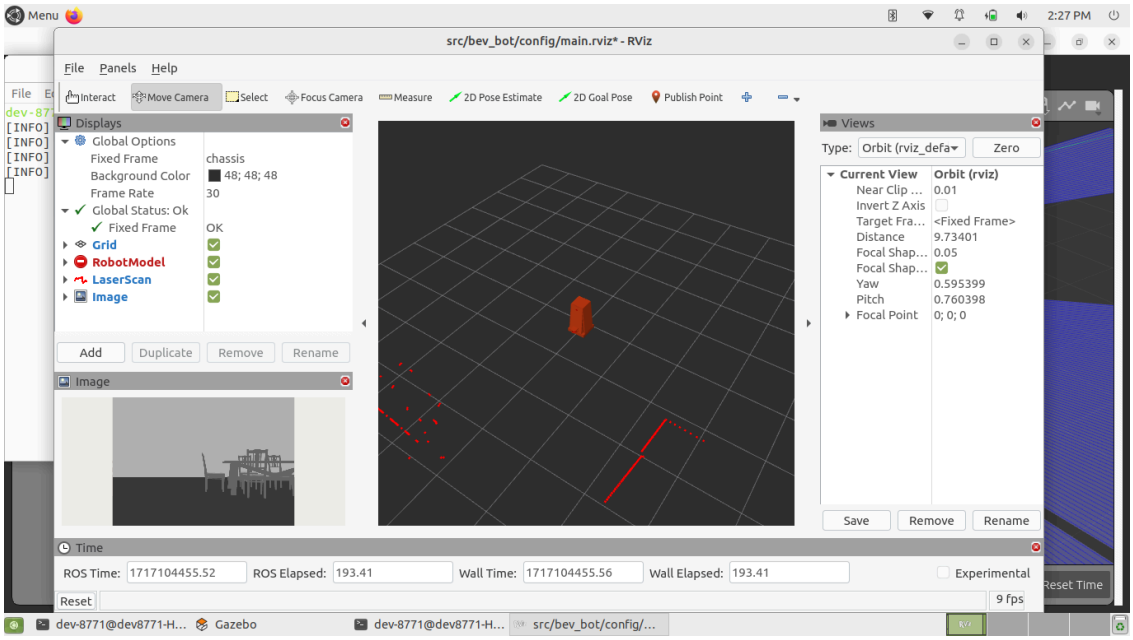


Figure 3. RPLIDAR Data Visualization

the USB port used, the URDF link it corresponds to, angle compensation, and scan type. The URDF link ties the data produced with the description of the robot for visualization and navigation. With these set the RPLIDAR node is set up and able to transmit its topic “laser_scan” to the rest of the ROS system.

The OAK-D-Lite depth camera is a powerful camera that we eventually were unable to use to its full potential because of the CPU limitations of the onboard computer, the Raspberry Pi. The OAK-D-Lite uses a stereo depth camera to create a point cloud of its view, this could be set up using their launch file to share color and depth images to the ROS system. Just like the LIDAR, the depth camera must be linked to the description of the robot via the URDF file. This camera works amazing and some of the resolution it produces with its depth feature integrates into this project well. It was only discovered in our prototyping phase that along with the rest of the sensors and software running the onboard computer, an 8 GB Raspberry Pi 4, did not possess

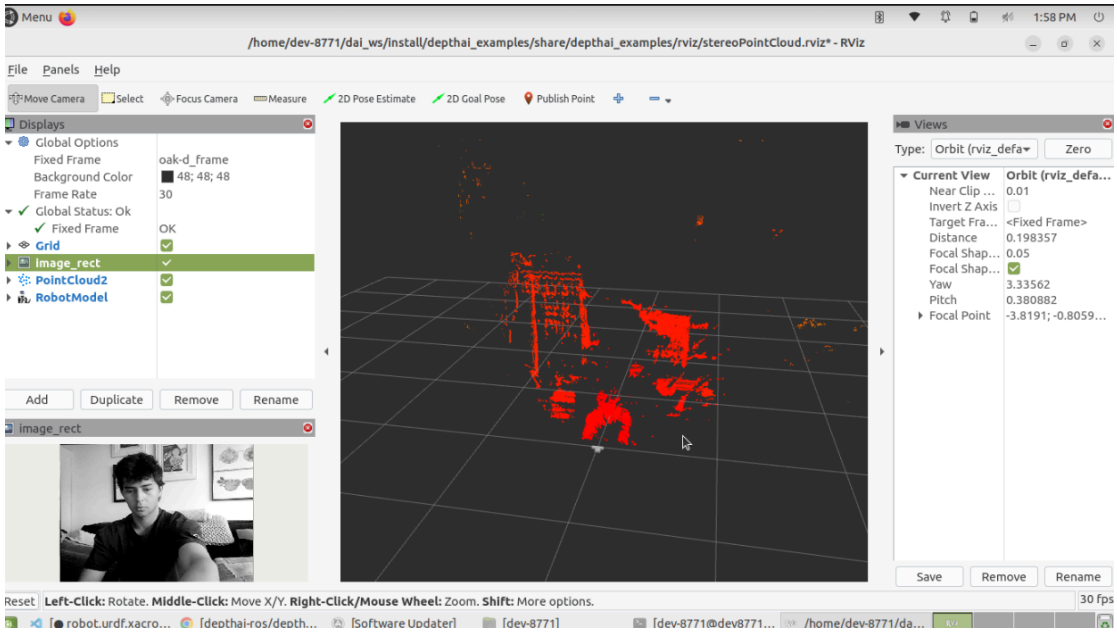


Figure 4. OAK-D Lite Data Visualization

adequate processing power. The camera was used and tested on the off-board development laptop, another Linux machine which was able to provide the required power. The data produced while running the depth camera on the development laptop produced topics such as point cloud stereo points, and RGB video images. While implementing it on the prototype and using the Raspberry Pi, the latency of the data deemed the depth setting unusable, and only the RGB data was able to be used. The ability to use the data produced is entirely obtainable and would have produced a more accurate view of the world for our robot, that being said with a more capable onboard computer the full use of this sensor would have been possible.

This wraps up the two pure senor nodes of the project, motor encoder feedback is used later on but this data is not published to the ros system in the form of topics. The data created from the motor encoders is used to publish its odometry or position and will be discussed along with the motor control node. The ROS system allows us to efficiently use sensors with their

unique drivers and launch files to create a more modular system, this has also proved to be helpful when approaching a specific sensor for debugging and quality control checking each part of the entire system. ROS allows for greater versatility for robotics defined by the needs of their respective system, although our project goals only required light obstacle avoidance, this technique can be employed for any application.

URDF

One of the largest learning curves that needed to be understood before progress could be made on this project is understanding ROS notation. Robots can be as simple as a single-jointed arm to as complex as a self-driving car. With many moving parts and sensors, a standardized notation is needed. URDF or Unified Robot Description Format is a collection of XML files written to describe every part of a robot. This entails the shape, weight, orientation, and countless factors of every still or moving part of a robot. An efficient way to think of the URDF model is to describe it as separate parts or “links” linking the robot together. These links could either describe a point or “frame” which is used by sensors and actuators, or describe a shape that represents the robot's actual shape such as a wheel or a body. The URDF is in a way the connection between all nodes of the robot, it tells the system where the robot is in relation to the location of its sensors, and how it moves based on the relation between moving and still parts. With nearly 1000 lines of code to describe our bot, notation must be maintained and understood.

The conceptually harder aspect of the URDF, the links describing a frame, is a good place to start. Although they describe a frame such as the location and orientation of “laser_frame” they are often the center of a body as well. The laser data is read at the center of a spinning cylinder which is the rotating time of flight sensor. This section will

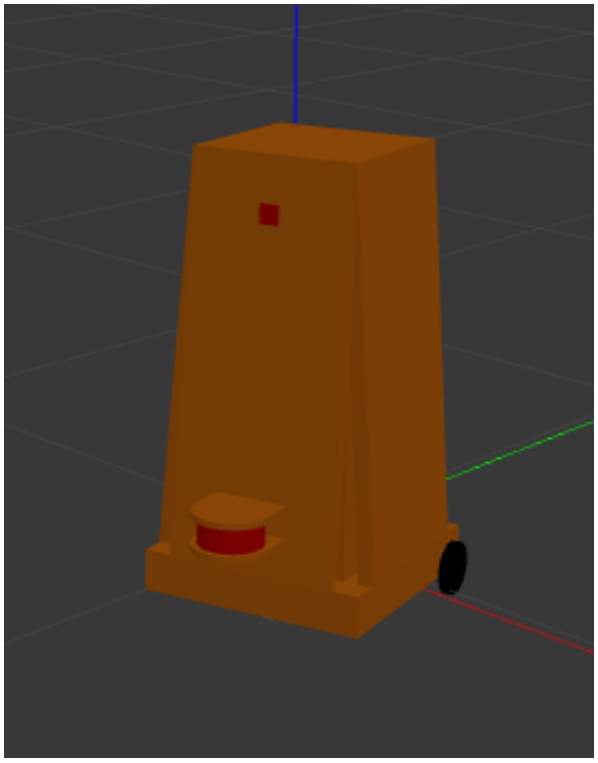


Figure 5. URDF Robot Description

only describe the frame of the sensor, and its shape or “body” will be described later with the rest of the shapes. The frames of a robot are heavily used later in the project when visualizing and navigating the robot. This data is extremely important because, without consistency in the description, the received data will not match. The frames mentioned are laser_frame, camera_link_optical, right_wheel, and left_wheel. The laser_frame and the camera_link_optical link their specific sensors to the rest of the bot, but the wheels are a little bit different. The reactivity needed for actuations is higher than for sensors where latency could be the difference between your bot driving into something and damaging itself or not. That being said, the frames for the bot’s wheels require extra setup such as setting up state (read) and command (write) interfaces. That

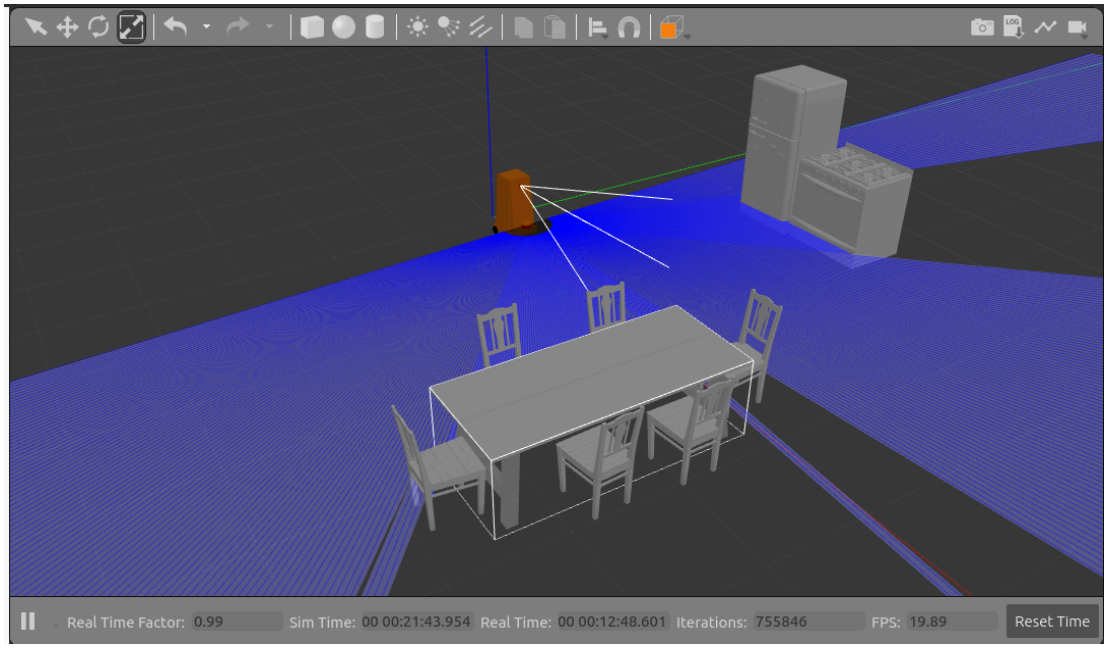


Figure 6. URDF with Camera and LIDAR Visualized

being said, these specific frames must be consistent throughout the ROS system and link the nodes to the rest of the system.

The other links connect the shapes of the robot. The body of our robot consists of 6 different shapes because of the design work of the mechanical lead. ROS only provides basic shapes, so a tapered rectangle had to be made out of a rectangular chassis, a taller rectangular chassis, and four flat side panels oriented to connect both chassis'. Bodies of drive wheels, caster wheel, camera body, lidar body, and lidar guards are also made. The ROS notation for URDF files allows for files to be broken up into xacro or XML macro files. The URDF is the connecting factor for the robot and because the file's accuracy limits the actual robot's accuracy damping values, inertial values, weight, and friction coefficients are added. High precision values such as moments of inertia of shapes that are prototyped are unrealistic but rough

estimates increase the accuracy of the system down the road during mapping and localization which proved to be beneficial.

The URDF file is a monster with fifteen different shapes each at different x y z locations, yaw pitch roll values, weights, colors, collisions, inertia, and countless other metrics. The project and our goals demand it. The largest takeaway is how the URDF changed with our project, initially, our bot chassis was a non-tapered rectangular, with the lidar on top, the development we experienced with the prototyping process forced us to approach the bot with a different design, and these iterations tested and honed my understanding of how robot description must be written. The deployment of the beverage service robot like most developments in industry is iterative, and because of this standards are required.

ROS2 Control

The Most challenging part of the beverage service robot regarding embedded systems software architecture was the motor control node. As briefly discussed the node that controls the bot's motors is different then other nodes reactivity is crucial when referring to motors. The motors for a bot of this size can easily break the lidar sensor mounted in front if it runs into a wall. In comes ROS2 Control. ROS2 Control is a framework for setting up a control node, this consists of controllers, their interfaces, and the actual sensors or actuators. The motor controller being used is an open-source controller provided by a Bay Area robotics club, Home Brew Robotics Club, and the ROS2 Control code was forked from the ROS2 Control GitHub repository but changed to work with the H.B. Robotics controller. A confusing ROS convention that provides clarity if cleared up is that the C++ code that is the heart of ROS2 Control is referred to as “hardware”, regardless that it is a software program with another program running on an Arduino between it and actual hardware. The ROS2 Control node runs high-level C++

functions and communicates with a ros-bridge motor controller on an Arduino performing low-level computations via classes. There are 3 classes for the entire system, wheel, config, and comms. This type of abstraction is consistent with the ROS2 software architecture.

The ros-arduino bridge used is specifically designed to interface with ros. The ros-arduino bridge communicated with the pi via pyserial, a Python module for communicating via the device's serial ports. Homebrew robotics separates the ros-arduino bridge into a motor driver, encoder driver, and differential controller. The encoder driver keeps track of encoder counts, the differential controller utilizes PID controllers for accurate real-time speed feedback, and the motor driver uses the transferred commands from the Raspberry Pi to set the PWM signals to the H-bridge. The only class communication between the Arduino controller and the ROS2 control controller is the config class, with an object, or instance of that class, cfg. Cfg contains all the information needed by the lower level ros-arduino bridge, the left wheel name, right wheel name, loop rate, string device, baud rate, timeout time, encoder counts per revolution, and PID controller values. The ROS2 control system can now focus on only high-level actions while low-level calculations are hidden.

The ROS2 Control “hardware” operates in lifecycles, with different states performing different actions. These life cycles include init, export state interfaces, export command interfaces, configure, activate, read, write, deactivate, and cleanup. The data communicated between the ROS2 Control hardware and the ros-arduino bridge is class config, object cfg, while the classes and objects within this node are class Wheel with two objects/instances, wheel_left and wheel_r, and class ArduinoComms object/instance comms. The code within the ROS control node consists of only high-level information such as setting up the 6 interfaces it uses to communicate with the other parts of the ROS system, setting configuration values, and reading

and writing information to the Arduino controller. This setup can be used for any type of controller, where data must be passed to and from another microcontroller. For our bot, this allows easy control where we can get read and write information continuously to the system in the form of motor commands and encoder feedback. As discussed earlier the entirety of the high-level ROS system communicates with the ROS2 control node In the form of state and command interfaces, the controller manager sets this up in an XML macro labeled Diffbot.ros2_control.xacro. Inside diffbot.ros2_control.xacro all parameters and configurations are also set, this includes the motor encoder resolution, the frames of the URDF which get controlled, and the port being used for the motor controller. Together the Arduino controller and the ROS2 Control node work to provide responsive control of the entire drivetrain.

The implementation of the ROS2 control node became the first time sensors could work in unison with actuators. The project design changed throughout the project, such as our initial idea involving a four-motor drivetrain and although this is possible, it required writing a ROS controller which was decided by the team as out of the scope of this project. The project became about utilizing the tools available to us, understanding them, and modifying them to satisfy our project goals. The ROS2 Control framework became the largest part of the project when discussing embedded systems software architecture, and because of this, we have been able to construct a robust, accurate drive train for navigation.

Mapping Localization and Navigation

With the sensors such as the 2D LIDAR and OAK-D Lite depth camera and the actuators such as the matching encoded motors that make up the drivetrain, the system needs to understand its surroundings and run navigation to move through those surroundings. The ROS software architecture works with a mapping and localization plugin called SLAM Toolbox. SLAM stands

for simultaneous localization and mapping, where localization is the act of reading these sensor inputs and localizing or matching itself within a given map. The mapping involves reading the bot's sensor inputs and saving the data to be used again. Another tool that will be discussed in this section is RVIZ which is the robot visualization software for ROS2. RVIZ acts as its own node and visualizes all aspects of the ROS system, this includes the robot description provided by running the ROS2 control node, the LIDAR output, the camera output, and the map provided by the SLAM toolbox node. With these final nodes, our project can utilize everything already done and satisfy the beverage service robots' project goals.

RVIZ acts as a central visualizer for the entire ROS system. The global frame could be set such as the map and all sensor inputs can be added. RVIZ also shows the URDF in the form of a robot model, this allows for all shapes and frames of the robot to be shown and described with correct orientation. As the robot is controlled the corresponding shapes which are moved via the ROS2 Control node show their translation in RVIZ, this includes both of the robot's wheels. The last and most important part of RVIZ is its use in mapping and navigation, since it is the main tool for interpreting a robot's view, it acts as a central place to incorporate the mapping, localization, and navigation. RVIZ allows SLAM Toolbox and NAV2 to be used and controlled through RVIZ in the form of plugins. RVIZ is an essential tool for correctly interpreting the bot’s sensor and actuator outputs and was necessary to the project.

SLAM Toolbox is a set of software programs for the ROS network that assist with 2D mapping and localization. SLAM Toolbox is a large tool with many different mapping and localization options, for this project, we specifically used online asynchronous mapping. The process for this includes bringing up all sensors on the bot and running a

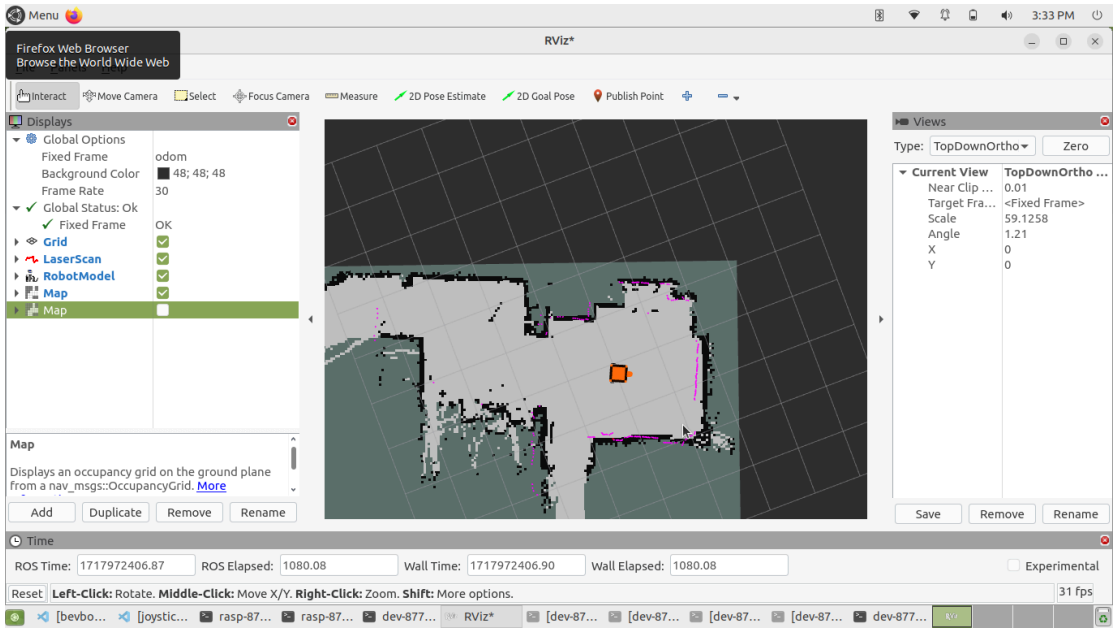


Figure 7. SLAM Map

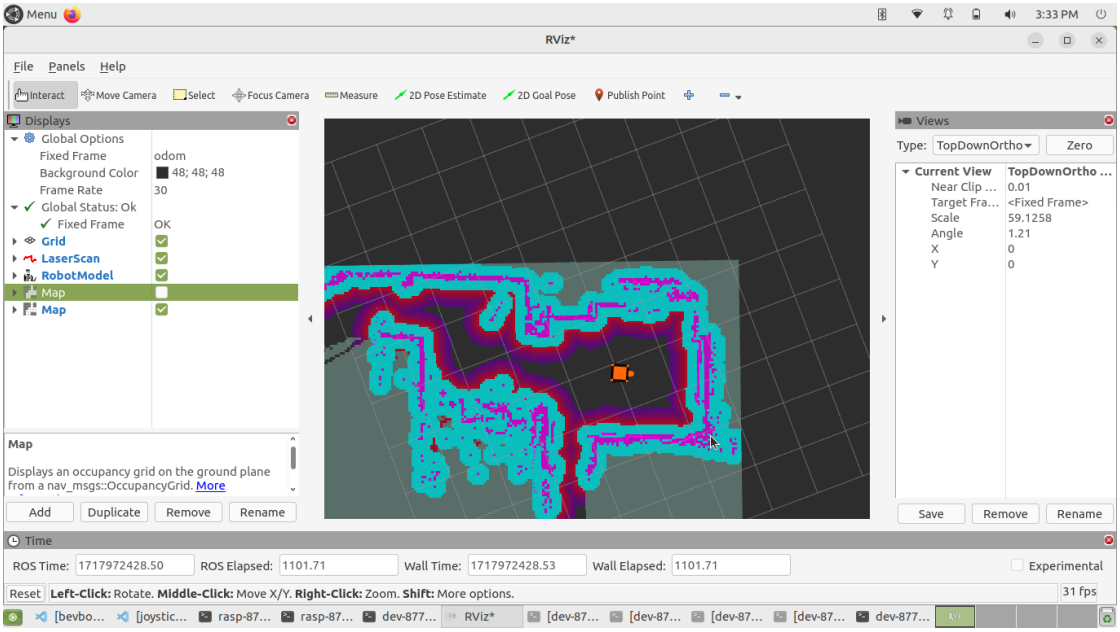


Figure 8. SLAM Cost Map

SLAM Toolbox node to create the map, with this map we can proceed to drive the bot around the room to maximize the data saved. All the data and how it changes while the bot moves around creates a 2D map of the area. This map gets saved as a posegraph and can be referenced later to be used. Although localization is included in the SLAM acronym, the actual localization process runs on a NAV2 node but uses this map. SLAM Toolbox is an essential tool for all high-level robotic actions such as navigation and localization provided by the NAV2 stack.

NAV2 provides the algorithms and means for which high-level navigation takes place. This includes behavior for autonomous navigation and a framework for autonomous actions in the form of NAV2 behavior trees. For our specific project, we
utilized the NAV2 Adaptive Monte Carlo Localization node, which takes in a saved map made using SLAM Toolbox and runs a particle filter to determine itself in the specific room based on the sensor data. When driving the robot in the room, the user can see the sensor inputs based on the map. As the bot drives freely, if the odometry and map

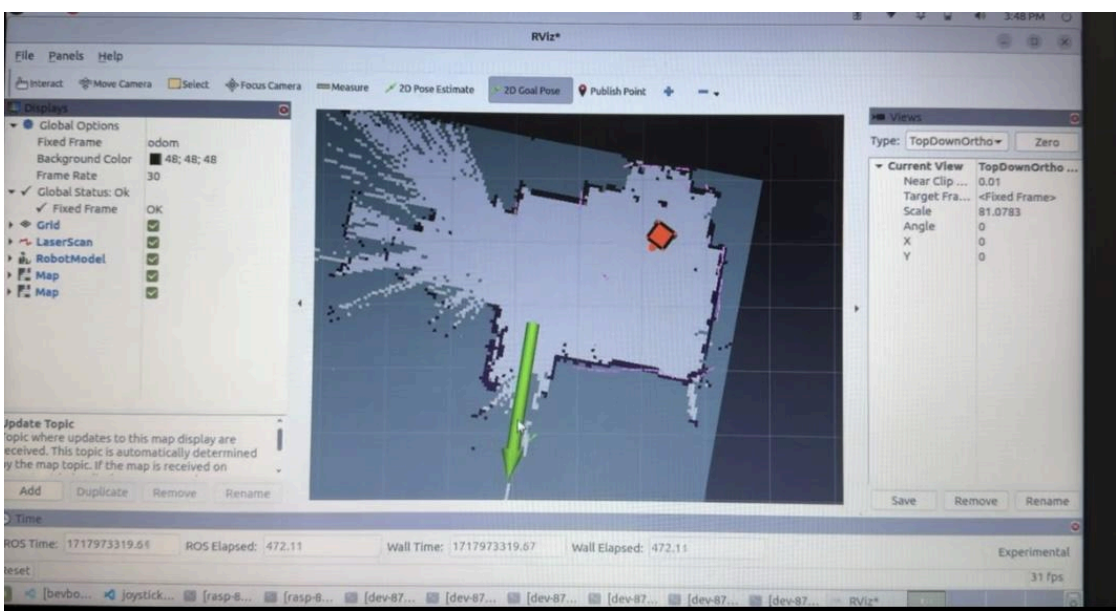


Figure 9. NAv2 Waypoint Following Setting Waypoint

frames are visible in RVIZ, one can see the odometry drift to keep the sensor inputs localized with the map. Next, we can run NAV2 navigation scripts that are provided and set waypoints on RVIZ. RVIZ also includes a plug-in for NAV2 where this is done. NAV2 runs this navigation using autonomous obstacle avoidance and sets its path based on an area's “cost”. This cost is calculated based on the robot's distance from an object, the closer the object, the higher the cost. The robot will navigate to the set waypoint in the path of least cost. A given map can be represented as a cost map and different costs of an area are represented with a color gradient. The bot is now using high-level programs to efficiently use its sensor data in its world.

The beverage service robot is an iterative project, iterating on software systems and programs already developed for our specific needs. The ROS software system allows a lot of freedom when it comes to navigation and the work done is only the tip of the iceberg. Navigation using NAV2 includes many other tools to fit different applications such as writing behavior trees. This is a large part of navigation which makes the bot fully autonomous, and although this was not achievable within the scope of this project, full autonomous actions are included in plans for this project. Navigation is the center of all mobile robots and stood as the main goal when discussing software, as the beverage service robot team, we were able to run high-level navigating software specific to our bot.

Software Conclusion

The complexity of our project and the restrictions placed by our project goals required a specific means of running our hardware. The ROS software system gave us the modular, iterative freedom to create the project our way. We had been able to interface nodes for specific motors, sensors, and navigation. Our robot was able to be designed in software to accurately reflect the shape, weight, and motion of our physical robot, and we were able to bring all of this together to

accurately map and traverse any area. This project involved understanding the entire process of industry-standard robotics development, including installing the hardware in the form of two Linux machines and all dependencies for the ROS environment. The largest takeaway from this project, when referring to software, is the importance of working consistently with standards. Robotics runs on standards set by the tools being used, ROS being such a large presence in robotics development, sets its standards based on the required systems. Understanding how to properly understand and describe these systems became essential for debugging and eventually utilizing them for this project. Software showed itself to be such a large aspect of this project and the work done provides insight for future works in industry.

Mechanical Design and Implementation

Mechanical design includes everything the robot needs to properly service a customer. Overseeing physical design, prototyping, and fabrication aspects of the robot, including material selection, drive-train configuration, sensor positioning, and industrial design aesthetics. This involves developing CAD models and drawings to ensure the assembly of all mechanical components. They validate that the final robot meets all requirements related to range of motion, structural integrity, tolerance, fluid pumping system performance, and safety factors. The end objective is to deliver a robust, field-ready robot platform that meets the required suspension specifications and reliable mechanics for dispensing drinks, with sensor mounting points and space pre-allocated for embedded systems integration. In our project, the mechanical subsystem comprises two components, the drive train and the shell.

Drive Train

The drivetrain consists of all the parts needed for the movement of the robot. As a team we brainstormed the best way for a robot to move around a restaurant without inhibiting the workflow of the servers. We first decided to try a holonomic drive train which would have had four motors driving omni wheels. The main goal of this drive was to have omni directional movement (a feature that allows motion in any direction without orientation maneuvers) for faster obstacle avoidance and to never get stuck in tight corners or around tables. We later found out that writing a controller in the ROS system would be too complicated for our project. We still wanted to have four motors to help move the main weight of the water (~10lbs), battery, and shell. This was when we went for our backup plan, a square base with four motors driving each wheel. This design was then again setback by the available ROS motor controllers. After many iterations and setbacks, we decided to build a differential drive system consisting of two motors, each driving a 3.5-inch wheel, and a single caster wheel in the front. We decided to go with this layout because the ROS system has prewritten controllers for differential motors. The single caster design helps with better turning and obstacle avoidance.

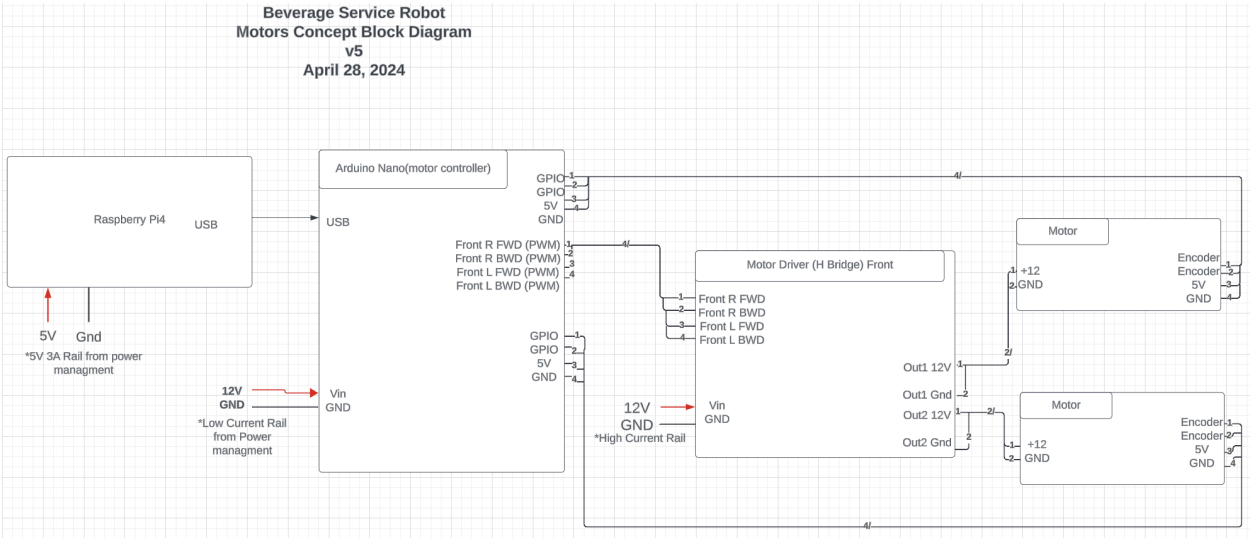


Figure 10. DriveTrain Block Diagram

Starting from the left of the drivetrain block diagram we see that the architecture of the drivetrain starts off with the Arduino Nano connecting to the Raspberry Pi4 via USB. Each component like the Pi, Nano, and H-bridge connects through the power rails provided by the Power subsystem. The motor drive we chose was an H-bridge to take in PWM (pulse width modulation) signals from the Nano and differentiate the 12-volt output and ground terminal to be flipped to turn the motor in the clockwise or counterclockwise directions. When the motor gets driven from the H-Bridge it returns encoded values back into the Nano which feeds the position of each wheel to the ROS controller system.

Computer Aided Design of Drive Train

During each iteration of the drive train a CAD (computer-aided design was created to test the feasibility and plan out the build. These designs are illustrated below in the SolidWorks computer program. This helped us plan the best size, shape, and configuration of the drive. This helped the team plan what materials we wanted the baseplate and motor mounts to be made out of. This was important to keep the base lightweight and strong enough to support the weight of all the internal components and the water that will be dispensed.

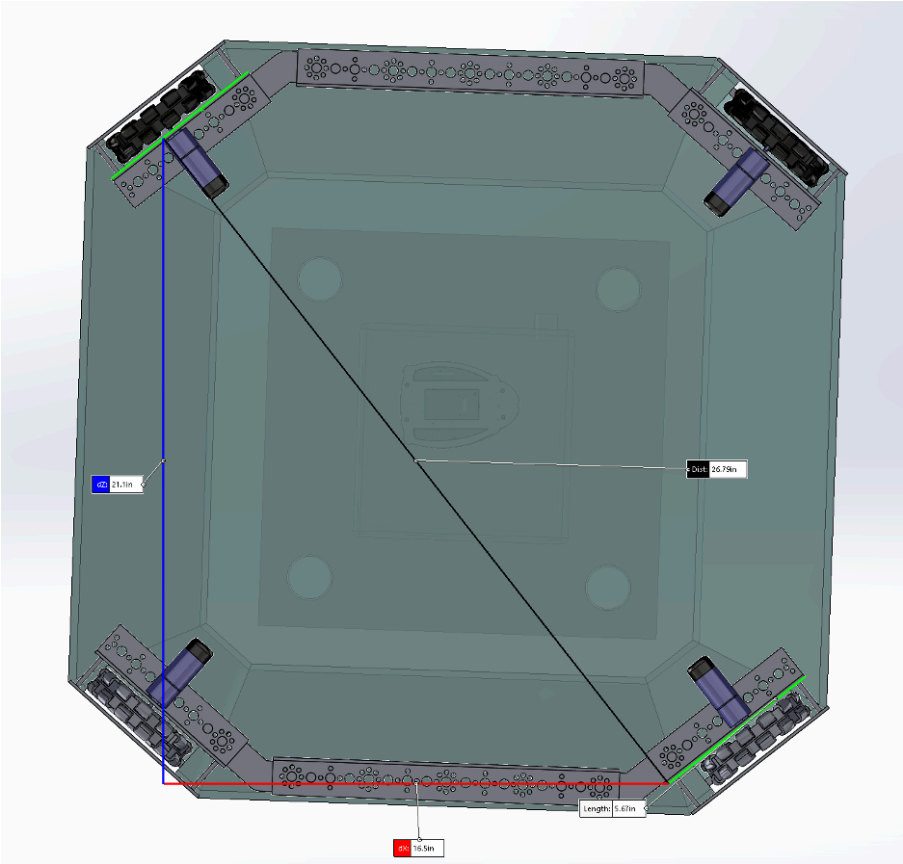


Figure 11. Holonomic Concept 1

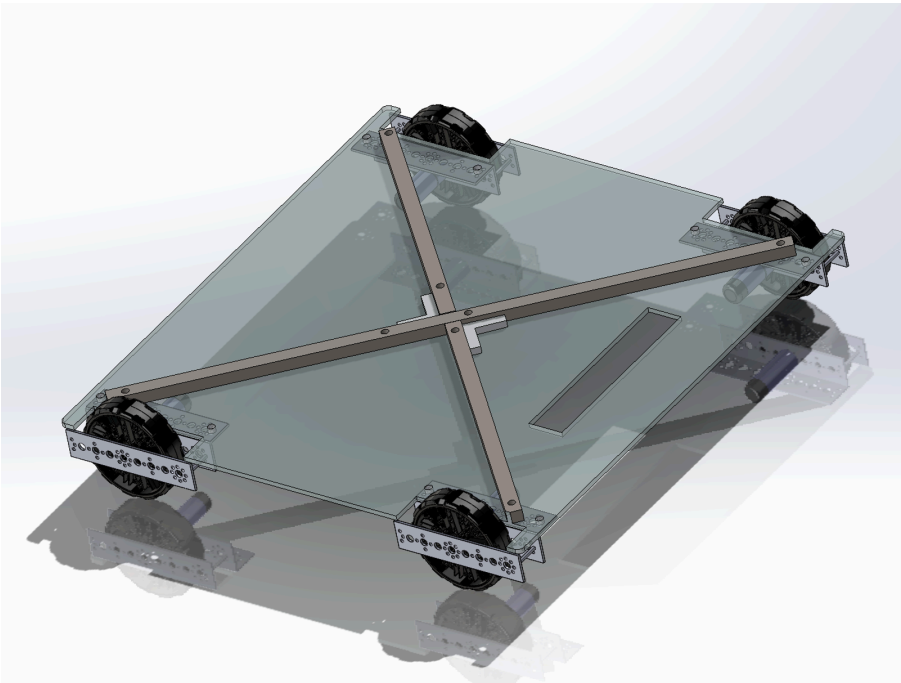


Figure 12. All Wheel Drive Back Up Concept 2

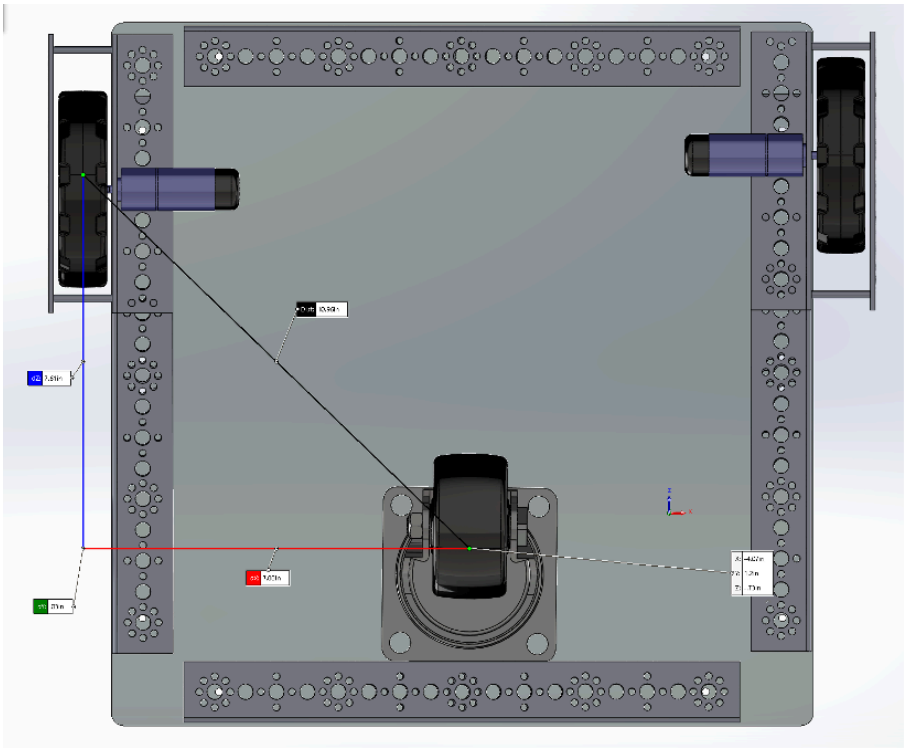


Figure 13. Final Differential Drive Concept 1 (bottom)

Motor Selection

The mechanical design process involved the critical selection of motor components that would meet the robot's specifications. The motors needed to operate smoothly to ensure safe and reliable transportation of water around the restaurant environment. The key requirements included sufficient torque to propel the robot at a constant speed, an internal encoder for relaying rotational position feedback to the Robot Operating System (ROS), a minimum movement speed, and lightweight construction. Particular emphasis was placed on torque considerations.

To determine the required torque, the angular rotation and torque formulas given in Equation 1 were used, based on the pre-planned estimated total weight of the robot and the number of motors utilized. The design initially started with four motors, allowing each motor to require less overall torque. However, this design was subsequently revised to a final configuration involving two motors, necessitating each motor to possess greater torque and rotational power to maintain the specified minimum speed. The different calculations are presented in Equations 1 and 2.

To Move 30lbs

Mass = 13.607 kg

F = 13.607 kg * 9.81 m/s^2 = 133.485 N towards gravity

Four wheels: 133.485 N / 4 = 33.37 N

Convert N to Nm: 33.37 N * 0.0508 = 1.6952 Nm required per wheel

Equations 1. Minimum Motor Torque Calculations (four wheels)

To Move 30lbs

Mass = 13.607 kg

F = 13.607 kg * 9.81 m/s^2 = 133.485 N towards gravity

Two wheels: 133.485 N / 2 = 66.74 N

Convert N to Nm: 66.74 N * 0.0508 = 3.39 Nm required per wheel at axel

Equations 2. Minimum Motor Torque Calculations (2 wheels)

Torque at wheel edge = Torque of axle * (Wheel Radius / Axel radius

Torque of axle = 68.4 kg force per cm => 6.71 Nm (datasheet)

Torque = 6.71 Nm * 0.044 meters / 0.006 meters = 0.915 Nm

Equations 3. Wheel Torque Calculations of Chosen Motor (2 wheels)

Equation 3 represents the torque applied at the edges of the 3.5-inch wheels, calculated by multiplying the applied torque at the axle, as provided by the motor's datasheet, by the distance from the outer edge of the wheel. Subsequently, to determine the force required to propel the robot, the estimated total weight of the final assembly, including the water reservoir, was utilized. This force value was then divided by four, as the initial design incorporated four motors. By employing this calculation method, a comprehensive evaluation of commercially available motors with internal encoders could be conducted to ascertain their suitability for the project's requirements.

The systematic analysis of torque requirements, in conjunction with the incorporation of encoder feedback for precise motion control, formed the basis for an informed motor selection process. This approach ensured that the chosen motors would provide sufficient torque and rotational control to facilitate safe and reliable transportation of the water dispenser throughout the restaurant environment, thereby enhancing the overall user experience. In Equation 2, it is shown that the exact same calculations were done for the differential drive with two motors

moving the weight. The specified torque from the motors was given by the goBILDA website. The motor selected had a D-Shaft that providing 117 RPM matching our required speed with a torque of 68.4 kg force per cm. This was then translated to Nm (Newton Meters), which is standard for torque calculations. It is seen that the 6.71 Nm is more than the 3.39 Nm required which is helpful in keeping the motors clear of stalling underweight.

Prototyping

While the motors were being shipped the next step was to build the base plate and motor mounts. Since the base needed to be strong and lightweight to ensure longevity, TETRIX metal C-channels were used for the frame. The connecting base plate was laser cut to match the exact CAD model. Metal bars were installed to add strength to the cross beams and hold the caster wheels. A Metal C-channel was cut to create an L-bracket to mount the motors. In Figure 14 the TETRIC C-channels can be seen mounted together with custom-made corner brackets and metal crossbars.

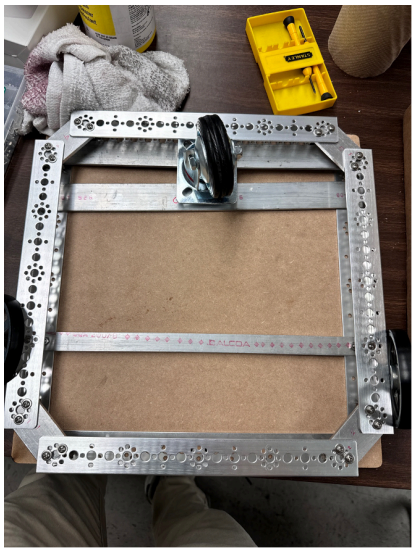


Figure 14. Frame Prototype 1

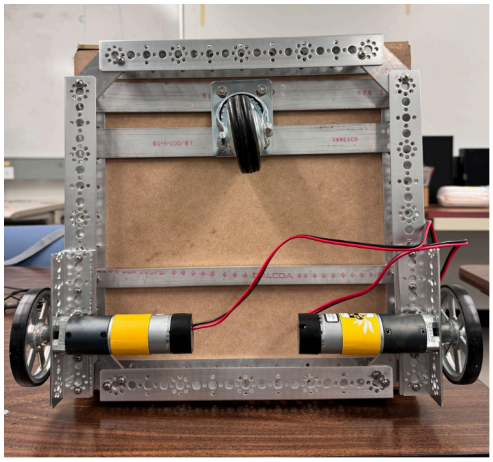


Figure 15. Bottom of Base Plate

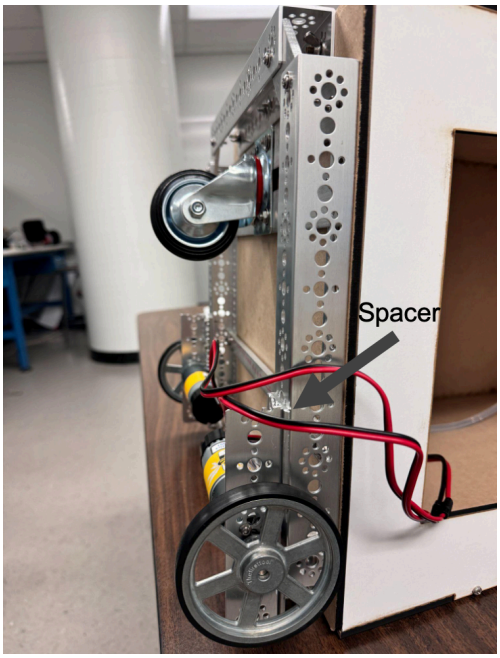


Figure 16. Side of Base Plate

Prototype Testing

After the initial prototype of the drivetrain, we needed to run a stress test to make sure the robot could support the intended weight of the internals. To do this we placed weight on the robot to simulate the water and battery weight totaling 20 lbs or ~9.1 kg. After weight was

added to the drivetrain on the floor we checked for stress points and bowing under the pressure. There were no major points of flexion in the baseplate or frame, so the prototype was verified. We then tested the movement of the wheels and turning using simple code to run the H-bridge. We initially ran into an issue with one wheel being under-rotated but figured out it was an encoder plug-in issue. The motors gave us the required rpm of around 100 and did not strain underweight. From this prototype, it was decided to continue changing small things to get to our final build. One main thing we needed to fix from the first few prototypes is the balancing of the caster wheel height to the driven wheel height. In Figure 16, a spacer was added to level out the height of the driven wheel to match the height of the caster wheel. This is critical for the sensor being added to be perfectly level with the floor on which it is traversing.

Motivation Behind Shell Design

The shell enclosure plays a critical role in protecting the internal electronics, battery, and water reservoir of the robot, while providing a sleek and streamlined design for seamless maneuverability in a restaurant environment. Careful consideration was given to the shell's shape, size, and weight during the design and implementation phases to ensure optimal functionality and user experience. Through extensive team discussions and assessment of the robot's operational requirements, a truncated square pyramid shape was selected for the shell design. This choice was driven by the need for a slim profile to minimize spatial footprint around tables and walkways, thereby facilitating unobstructed navigation. Additionally, the truncated top portion of the design provided an ideal flat surface for integrating the water dispenser user interface.

Material selection was a crucial aspect of the shell design process, as the chosen material needed to exhibit strength, durability, and resistance to potential impacts, moisture, and soiling. Initially, carbon fiber was considered due to its exceptional durability. However, to validate the design concept and optimize costs, medium-density fiberboard (MDF) prototypes were fabricated and tested, leveraging the material's affordability and strength. Upon successful verification of the MDF prototype and confirmation of its suitability for the intended application, acrylic was selected as the final material for the shell. This choice provided several advantages, including precise laser cutting capabilities for achieving exact angles and measurements aligned with the robot's CAD model. Furthermore, acrylic's lightweight and semi-flexible properties mitigated the risk of cracking or damage in the event of accidental impacts, ensuring a robust and long-lasting user interface enclosure.

Shell Computer Aided Design

The computer-aided design (CAD) process for the robotic shell underwent several iterations to optimize its form, functionality, and manufacturability. The initial design iteration employed a rectangular prism shape, primarily driven by the simplicity of its representation in the Unified Robot Description Format (URDF) software. However, subsequent evaluations revealed that the rectangular prism shape was excessively bulky, leading to higher material costs and increased vulnerability to impact damage, particularly at the corners. To address these concerns, finite element analysis (FEA) simulations were conducted using SolidWorks software to assess the structural integrity and durability of the shell under various loading conditions. Though the minimal water dispenser weight at the top of the shell was not posing to be a problem. The FEA results indicated that a truncated square pyramid shape offered superior performance and robustness.

This design featured a top plate that could be securely fastened over a smaller area than the base, enabling the side panels to lean against each other for enhanced structural support. The CAD models illustrating the original shell architecture and the final optimized design are presented in Figures 17 and 18, respectively. The truncated square pyramid shape not only improved the overall structural integrity but also facilitated a more efficient use of materials, thereby reducing manufacturing costs. Additionally, the tapered design provided a more streamlined aesthetic, enhancing the user interface and overall visual appeal of the robotic system.

Figure 17. Rectangular Prism Concept 1

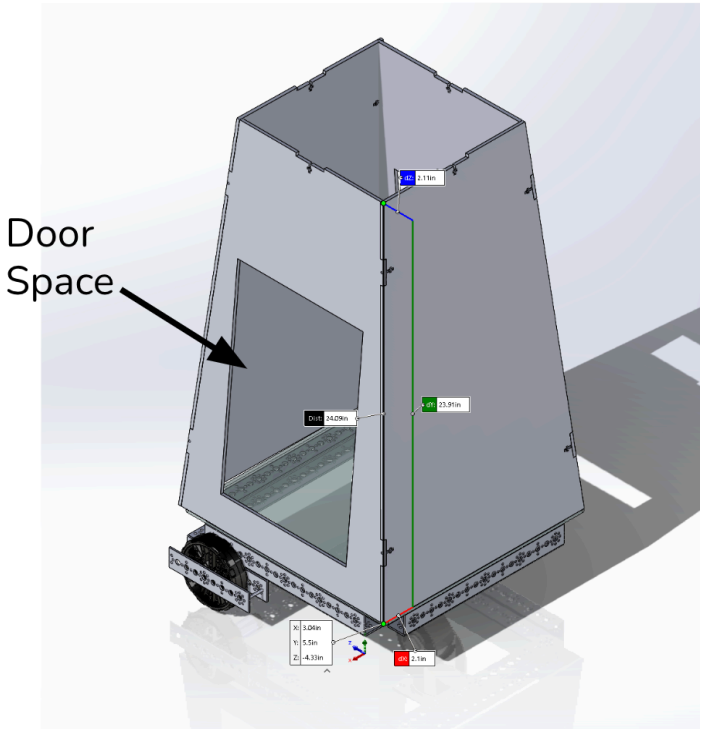


Figure 18. Truncated Square Pyramid Final Concept

A comparative analysis of Figures 17 and 18 reveals the significant differences in the overall shell shape, directly influenced by the constraints imposed by the drivetrain design. The door space indicated in Figure 18 was initially a placeholder, later replaced by an actual door. This door facilitated efficient access to the internal electronics, eliminating the need for complete shell removal during maintenance or troubleshooting operations.

The original concept involved leveraging the LIDAR placement to detect and detach tables obstructing the robot's path. However, this approach was subsequently revised, and the LIDAR was repositioned at the bottom of the shell, directly above the drivetrain. This strategic placement enabled the LIDAR to effectively detect and navigate around any obstacles in the robot's trajectory, enhancing its overall maneuverability and collision avoidance capabilities. Furthermore, as evident in Figure 19, each connecting panel of the shell was designed with

interlocking tabs and slots. This design choice served two critical purposes: first, it ensured a secure and precise panel-to-panel connection, and second, it facilitated the integration of screws to reinforce the assembly and provide additional structural integrity. The implementation of tabs and slots leveraged the precision of laser cutting techniques and CAD modeling, resulting in a robust and seamless shell assembly. This design approach not only optimized the structural integrity of the shell but also streamlined the manufacturing and assembly processes, contributing to an efficient and cohesive user interface enclosure.

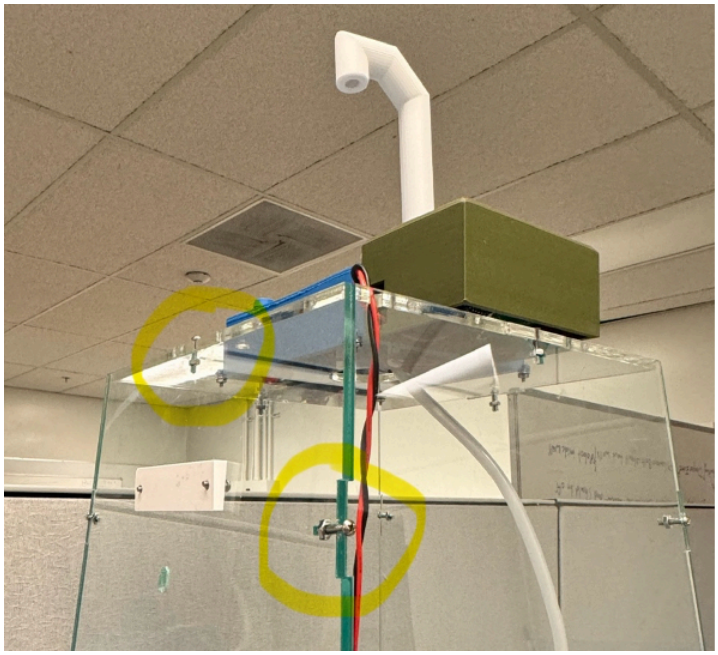


Figure 19. Acrylic Panel Connections

Prototyping Shell

Upon finalizing the shape and attaching it to the drivetrain in the computer-aided design (CAD) software, and receiving approval from the software lead for the truncated shape, the next step involved prototyping the shell. The initial prototyping process began by cutting each panel

of the shell from a medium-density fiberboard (MDF) using a laser cutter. MDF was chosen for its low cost and malleability, making it suitable for rapid prototyping iterations.

Since the laser-cut panels were precisely replicated from the CAD model, minimal modifications were required for the first prototype, apart from introducing a hole for the door and screw holes for mounting, which was later implemented in the second prototype iteration. The prototyped shell design was seamlessly integrated with the drivetrain and was easily attached using metal brackets, as illustrated in Figure 20.

This prototyping approach allowed for an efficient evaluation of the shell design's compatibility with the drivetrain system, facilitating further refinements and optimizations before proceeding to the final manufacturing stage.

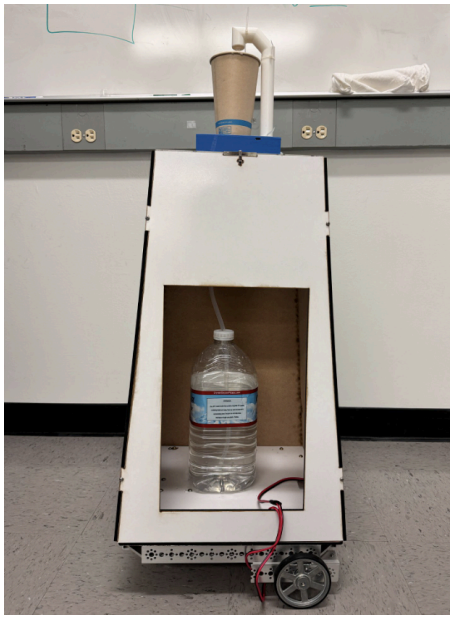


Figure 20. Prototype of Shell v.2 (with door hole)

The second prototype iteration was successfully integrated with the overall robot design. Upon completing the stress tests on the physical robot prototype, a crucial analysis was conducted to evaluate the robot's resistance to tipping when subjected to external forces. This calculation of the moment of inertia was essential for ensuring safe operation, as accidental collisions or impacts are common in restaurant environments.

The analysis revealed that the robot could withstand an applied moment of up to 450 $lb \cdot in^2$ at a height of 11 inches from the ground without tipping over. This determination was critical to quantify the maximum allowable force that could be exerted on the robot while maintaining its stability. Calculations of the moment of inertia were performed for each weighted section of the frame using established formulas.

The validity of this analysis was supported by the low center of gravity achieved through the strategic placement of the battery and water reservoir at a height of only 4 inches from the ground. Furthermore, the weight distribution of the motor and frame components contributed to maintaining a low center of mass, thereby enhancing the robot's overall stability. This rigorous stability evaluation ensured that the user interface design accommodated safe interactions with the robot, minimizing the risk of accidental tipping and potential hazards in the dynamic restaurant environment.

Final Shell

Upon completing the moment of inertia calculations and physically validating the design through shaking and pushing stress tests, the focus shifted toward fabricating the final shell enclosure. The shell was constructed using 4 mm thick blue-tinted acrylic sheets for the side panels, providing an aesthetically pleasing and durable exterior. The top cover of the shell was

laser cut from a 6.4 mm thick acrylic sheet. The increased thickness of the top panel served a crucial structural purpose, acting as an anchor to securely hold all four side panels together. This design approach ensured a robust and integrated shell assembly. The final shell design, seamlessly integrated with the baseplate, is captured in Figure 21. The meticulous fabrication process and the strategic material selection resulted in a visually appealing and functionally sound user interface enclosure, capable of withstanding the rigors of operation in a restaurant environment while maintaining a cohesive and inviting aesthetic.

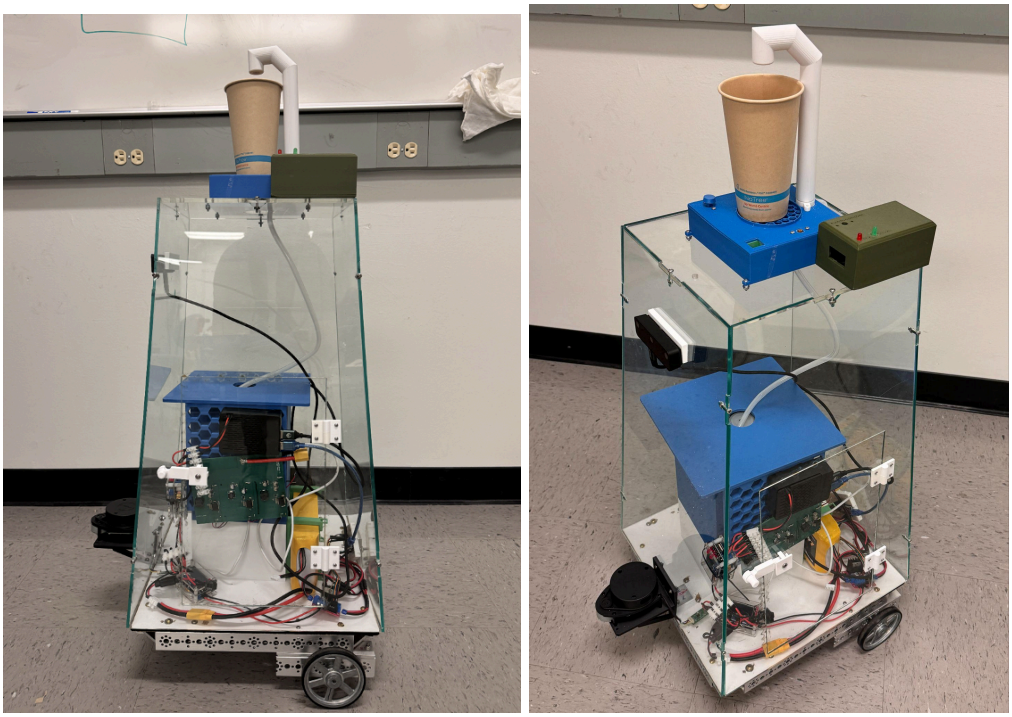


Figure 21. Final Acrylic Shell Design

Sensor Mounts

For simple integration of the chosen sensors, each sensor mount was custom 3D printed and implemented onto the final robot. The important sensor mounts include the Lidar holder and camera mount. It was necessary to 3D print these parts because the shell of the robot has a 85.3 degree angle with the base plate. This angle is crucial to replicate on the camera and Lidar mounts because they are required to be perfectly parallel to the floor to provide the most accurate data to the system.

The sensor mounts were first tested to find the optimal position to match the URDF file in CAD. This allowed for pinpoint accuracy when mounting to the front panel. The images of the Lidar mount are provided in Figure 22.

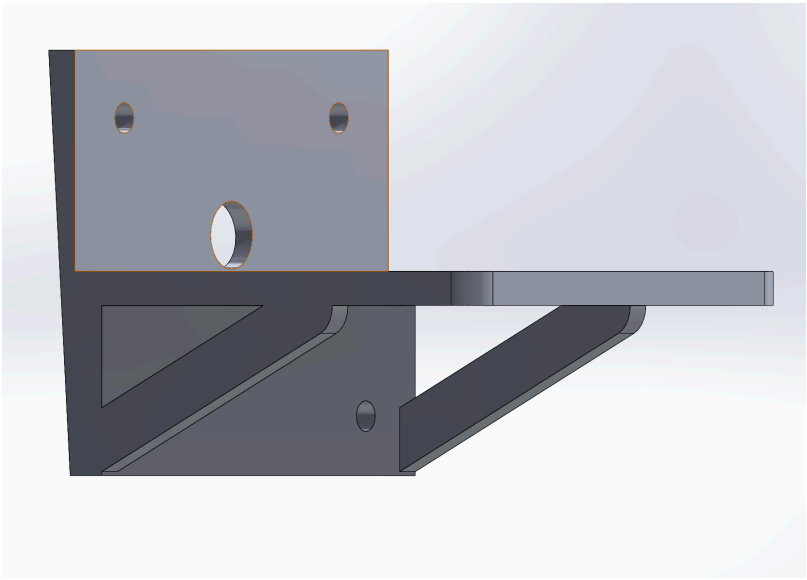


Figure 22. Lidar Mount to Front Acrylic Panel

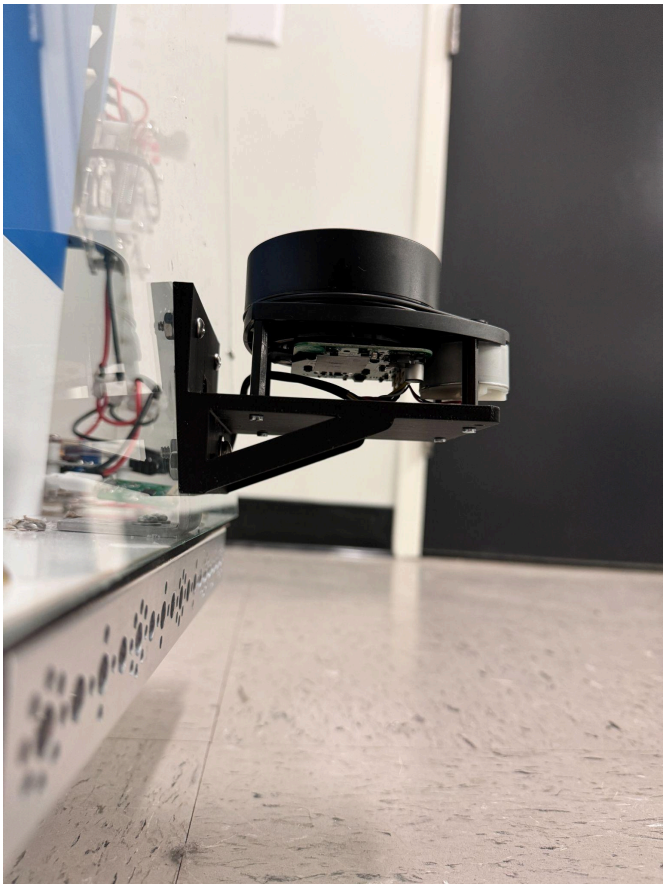


Figure 23. Implementation of Lidar Mount

By prioritizing the accurate positioning of sensor mounts through iterative testing and verification against the CAD model, the user interface design process ensured a cohesive integration of essential sensors with the robot's mechanical and software components. This attention to detail laid the foundation for reliable sensor performance, enabling the robot to effectively navigate its surroundings and deliver a seamless user experience in dynamic restaurant environments.

As depicted in Figure 21, the design incorporated several 3D printed components beyond the primary structural elements. One notable example is the water reservoir case and green battery mounts, which served a dual purpose: protecting the baseplate from potential water

exposure and providing a dedicated mounting location for the electronic components. These components were designed and optimized to meet specific functional requirements, while seamlessly integrating into the overall system architecture. These 3D printed parts demonstrate the connection between mechanical design, electronic integration, and user interface considerations, emphasizing the multidisciplinary nature of the engineering team process.

In conclusion, the mechanical design process for this restaurant robot involved extensive iterations, prototyping, and analysis to ensure optimal functionality, durability, and user experience. Through meticulous CAD modeling and systematic prototyping, the critical components—including the drivetrain, shell enclosure, sensor mounts, and material selections—were methodically thought out. The truncated square pyramid shell design, strategically integrated with the differential drive system, provides structural integrity while maintaining a sleek, streamlined aesthetic. Comprehensive stability evaluations validated the robot's resistance to tipping forces, safeguarding user interactions within dynamic restaurant environments. The integration of custom 3D-printed sensor mounts facilitated precise data collection and obstacle avoidance capabilities. This multifaceted mechanical design approach, supported by thorough torque calculations and iterative physical testing, culminated in a robust, field-ready robotic platform capable of reliably dispensing beverages while navigating obstacles and accommodating user interface requirements in restaurant settings.

User Interface Design

The user interface (UI) design encompassed two primary components: a water dispenser and a table radio transmitter module. The UI design process involved conducting user research

and interviews to gather insights into the needs of customers and restaurant staff. These insights informed the creation of prototypes and the final UI design in close collaboration with the software lead. The overarching goal was to develop a seamless and aesthetically appealing UI that enhances the overall customer experience, reduces the workload for waitstaff, and promotes user satisfaction. The UI design team was accountable for developing an intuitive and user-friendly interface that required minimal training for customers to operate effectively. Through iterative brainstorming sessions, the team explored the most efficient approach to integrate a water dispenser and a robot calling module that customers could utilize with simple instructions.

Water Dispenser

The team integrated a pre-built water pump assembly comprising a pump, battery, and charging port into a custom 3D-printed design for the water dispenser component. The requirements for a user-friendly water dispenser included a spill tray to catch any water spilled by customers and prevent it from reaching the electronics, an easily accessible button, a faucet for dispensing water, and a cup placement area at an ergonomic table height. Additive manufacturing techniques allowed for the customization of the button, screw holes, and cup placement areas through 3D printing. This approach facilitated seamless integration with the off-the-shelf water pump and enabled precise positioning of the fabricated faucet. The final CAD design of the water dispenser, shown in Figure 25, incorporates the cup placement area, water faucet, and button. To comply with sanitary regulations, the faucet was printed in two pieces to accommodate a water transfer tube. Once the tube is inserted, the faucet can be assembled, creating a seamless and hygienic water dispenser, as depicted in Figure 24.

Figure 24. Final Water Faucet CAD (iso view)

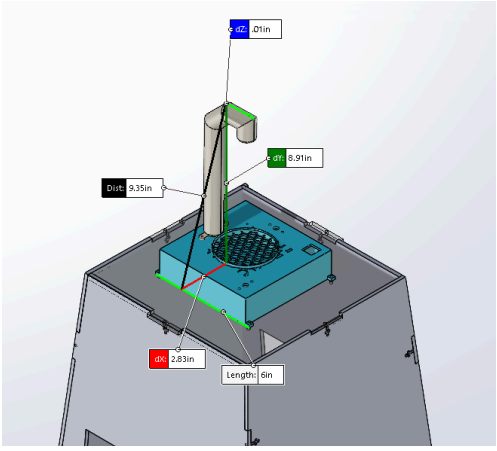


Figure 25. Final Water Dispenser UI CAD (iso view)

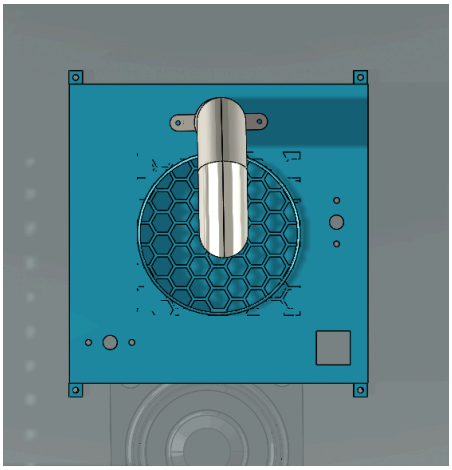


Figure 26. Final Water Dispenser UI CAD (top view)

After the 3D print was finished the dispenser was attached to the top plate of the truncated shell. The final dispenser is portrayed in Figure 27, with each part of the final print attached and able to dispense water to customers. This system has a single button required to be pressed to dispense water into a customer's cup when the robot arrives at their table.

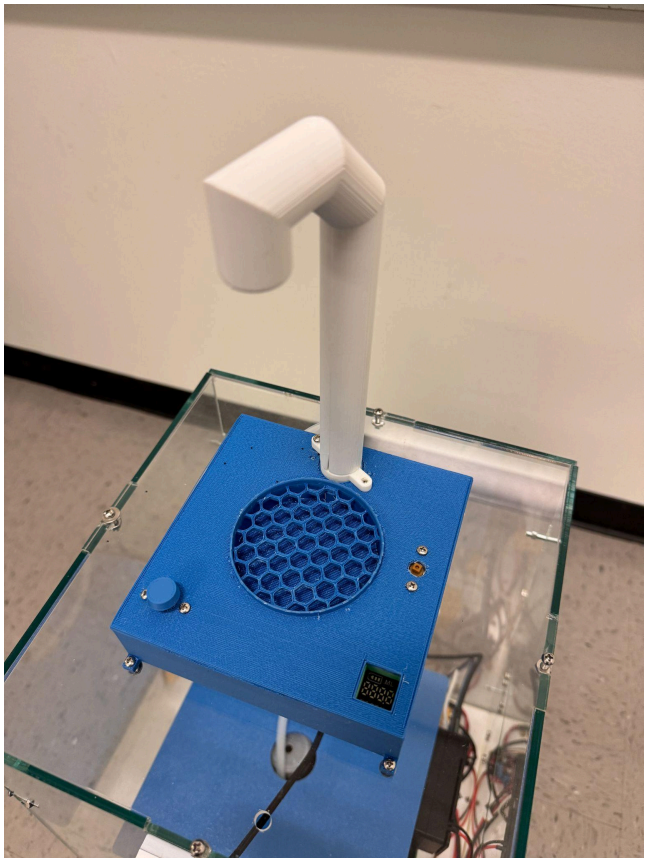


Figure 27. Final Water Dispenser

Table Modules

The user interface (UI) system is an integral part of the robot that is responsible for calling the robot to the desired table upon pushing a button. The UI system sits on the table at the restaurant and has a large push button that the customer presses to request the robot's service and an LED that lights up to notify the customer that the request has been received. This is done using a transmitter that sends a signal to a receiver on the robot. The transmitter and receiver are both powered by an Arduino nano microcontroller that is programmed to manage the power supply and communication protocols for the system. This design consisted of two parts, one for the robot to integrate into the Raspberry Pi and another for the customer to interact with.

To facilitate scalability and easy replication of the user interface across multiple tables, the table modules were designed for fabrication through additive manufacturing techniques. The computer-aided design (CAD) model of each table module is depicted in Figure 28, while the final implemented version is showcased in Figure 29. Each table module featured a minimalist design consisting of a single button and a light-emitting diode (LED) indicator.

While the current implementation focused on a streamlined user experience, future iterations could incorporate additional LED indicators to provide customers with real-time status updates on the robot's movement. Potential enhancements could include separate LED indicators for "busy," "onroute," and "finished" states, enabling customers to better understand the progress of their request. The decision to employ 3D printing for the table module fabrication process allowed for rapid and consistent replication, ensuring a uniform user experience across all tables. Additionally shown in Figure 30, the there is a receiving module that hold a printed circuit board with a Aurdino Nano to correctly relay to the robot that service to a corresponding table is being requested.

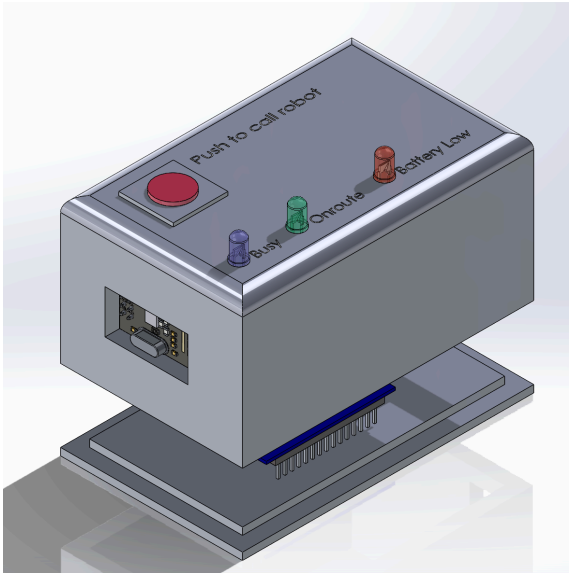


Figure 28. UI Table Module

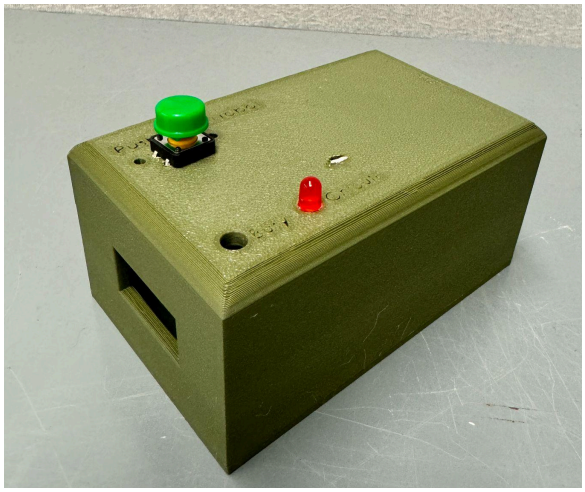


Figure 29. UI Table Module

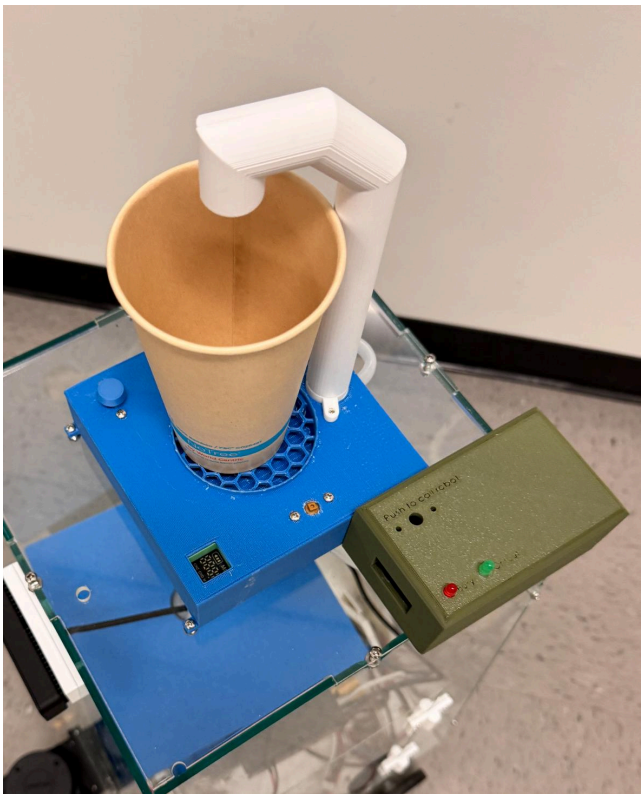


Figure 30. UI Robot Module

In summary, the user interface design process seamlessly integrated customer insights, usability principles, and scalable manufacturing techniques to deliver an intuitive and visually appealing solution. The water dispenser component, featuring a custom 3D-printed enclosure and

a hygienic faucet design, provided an ergonomic and user-friendly experience for customers to obtain beverages. Concurrently, the table module design prioritized simplicity and replicability through additive manufacturing, enabling efficient system expansion across multiple restaurant tables. For the receiving side of the robotic radio transmission the aesthetic of the 3D printed water dispenser was combined with a usable case to store LEDs to present the corresponding table module being activated.

UI Electrical System

The UI system for this project is designed for wireless communication between the customers and the robot itself. To establish wireless connection between the table module and the robot, both modules need to have both a microcontroller and transceiver. The table will have a nano and a transmitter to be able to send a signal to the robot. The receiver, that will be on the robot, will receive the signals from the table module when the button is pressed by the customer. It plugs into the onboard computer, which is a Raspberry Pi, and this tells the robot where it needs to go.

The motivation behind choosing the Arduino nano was specifically because of its compatibility with the nRF24L01 and the number of GPIO pins while still being relatively small in size. The Arduino and transceivers communicate with each other via SPI (Serial Peripheral Interface) Protocol, a form of wireless communication. SPI protocol is a type of synchronous serial communication used for short-distance communication. It operates using a master-slave architecture and can have multiple slaves with a single master. The master generates a clock signal on either a rising or falling edge that synchronizes the data transmission to the slaves. The master is able to send and receive signals simultaneously with the slave in a full-duplex manner.

The nRF24L01 transceivers are powered using the 3.3V source on the Arduino Nano. The Arduino Nano on the receiver is powered by the Raspberry Pi and those on the transmitters are plugged into the wall via USB-C.

It was determined that the UI system should be implemented on a PCB (printed circuit board) in order to fit on both the robot and table modules. Before this could be done a series of tests were conducted to ensure the design of the modules system would work. This began by developing a block diagram of the overall system. The first test was to determine if the transmitter could send a signal and be received by the receiver. This was done by plugging two transceivers into a single Arduino Nano to establish a connection between the two. The block diagram of this is shown in Figure 31.

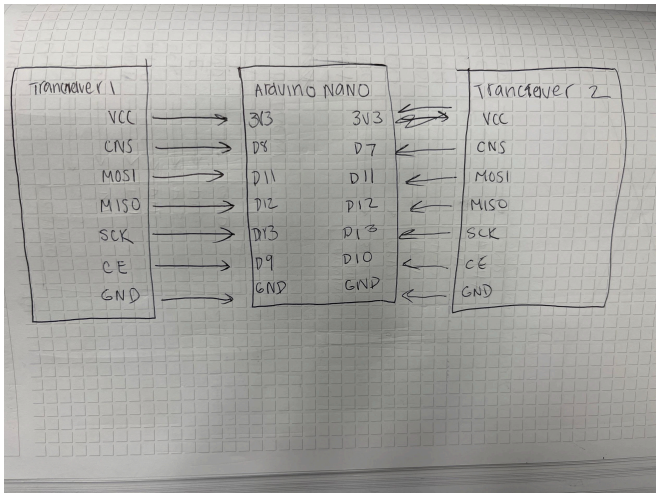


Figure 31: Block Diagram for First Transceiver Test (Single Nano)

Once it was established that the transmitter and receiver could send and receive signals to each other, a second test was conducted to establish a connection between them with two Arduino nanos. The circuit for this experiment is shown in Figure 32.

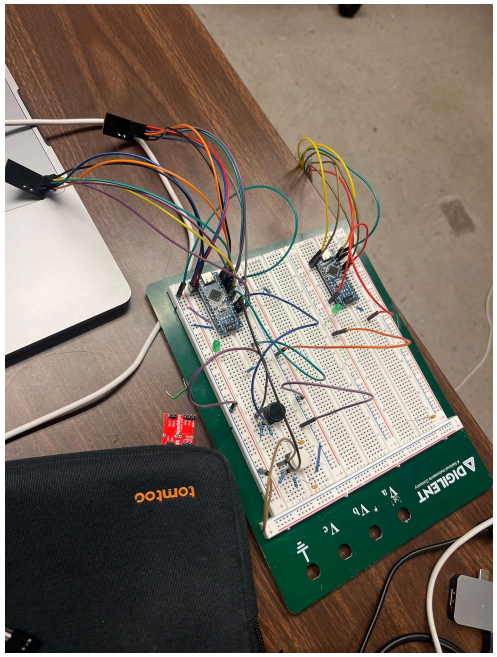


Figure 32: UI Circuit Test with Two Arduino Nanos

This test was conducted to establish connection between the transmitter and receiver on two separate nanos. This test was done twice. Once was done on the same breadboard to establish a connection and the second was done on two separate breadboards to determine if a connection could be made from a distance. One person stood on one side of the lab and another on the other side. Each circuit had a button wired to the Arduino nano. A code was written to send a message in the serial monitor if the signal was received when the button was pressed from the opposite transceiver. The code for this test is shown below in Figure 33.

TITLE UI code PROJECT

Continued from Page

Arduino Nano One transmitter to one receiver

include lib SPI.h, nRF24L01.h, RF24.h

Create RF24 object with radio
Create address "0000n"
radio.begin();
set address radio. open writing Pipe
set as trasmitter:
radio. stop listening()

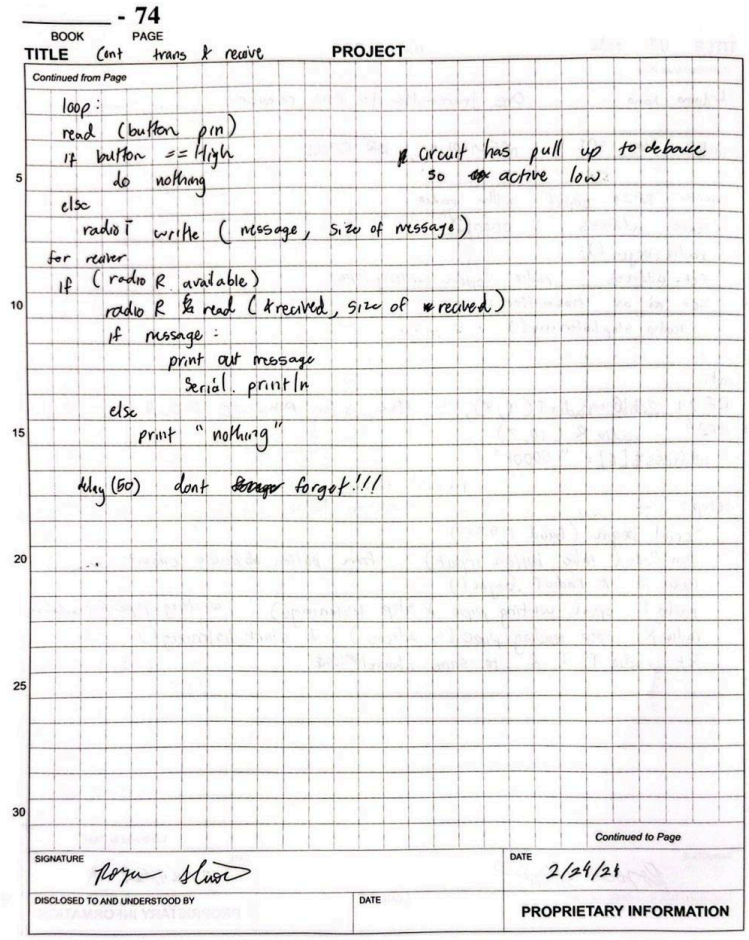
init:
RF 24 radioT(9, 8); this is for Pins CE & CSN
RF24 radio R (10, 7)
address[6] = "00001"

Setup:
Serial begin (baud 115200)
pin Mode (take button input) from button debouce circuit
Radio R & radioT .begin()
radio T open writing pipe & stop listening() writing pipe for address
radio R open reading pipe (1, adress) & start listening()
set radio T & R to same channel#100

Continued to Page

SIGNATURE Roger Hunter DATE 2/24/24

DISCLOSED TO AND UNDERSTOOD BY DATE PROPRIETARY INFORMATION



_____ - 74
BOOK PAGE
TITLE Cont trans & receive PROJECT
Continued from Page

```
loop:
read (button pin)
if button == High                    circuit has pull up to debouce
    do nothing                       so active low
else
  radio T write (message, size of message)
for reaver
  if (radio R avalable)
      radio R read (received, size of received)
      if message:
            print out message
              Serial. println
      else
          print "nothing"

  dlay (50)  dont forget!!!
```

Continued to Page
SIGNATURE Roger Shuse DATE 2/24/24
DISCLOSED TO AND UNDERSTOOD BY DATE PROPRIETARY INFORMATION

Figure 33: Code for Second UI Test

After these tests were conducted, it was determined that this circuit would work for the robot UI system. The next step was to then implement both circuits onto a PCB (printed circuit board) and then run additional tests. A final circuit schematic was drawn for both the transmitter and the receiver modules before developing the PCB (Figures 34 and 35).

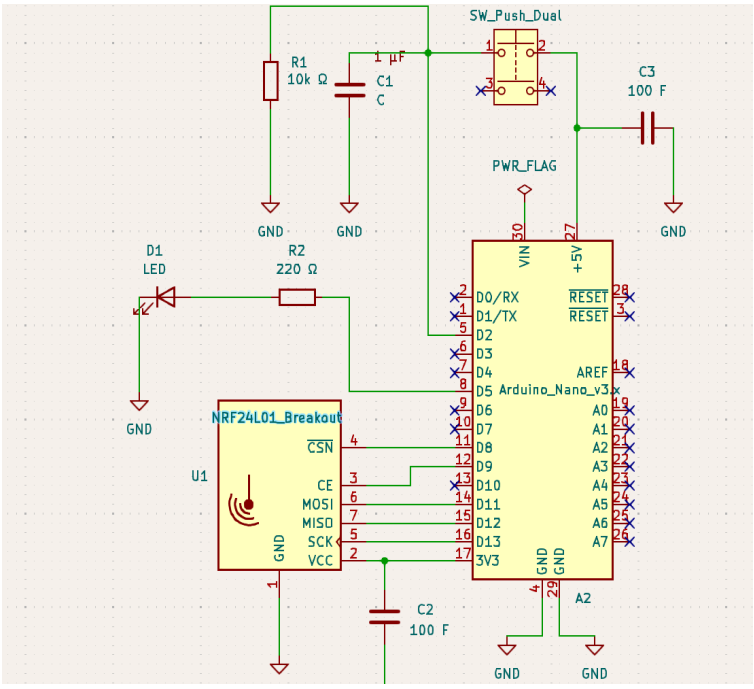


Figure 34: Final Circuit for Transmitter

The circuit in Figure 34 is the Transmitter circuit. The main part of the circuit is the push button for the customer to call the robot over. The push button has a pull down resistor that keeps the logic level at a 0 when it is not pushed. When the button is pushed, the signal will change to a 1 and send a signal to the nano. A 1 microfarad capacitor was placed in parallel with the resistor to debounce the signal. If this is not in place the signal could switch between a 1 and 0 and send improper signals to the nano. Each voltage source is connected to a capacitor to decouple the signal. This is to ensure it is getting enough power. The second resistor is in place before the LED to limit current going through it. LEDs allow current to flow in one direction and light up when there is a steady flow. The current limiting resistor is in place to ensure the LED does not blow from too much current. The nRF24L01 transmitter is powered using the 3.3V source from the arduino nano and this is also connected to a decoupling capacitor. The circuit for the receiver is straightforward. The nRF24L01 connects to the Arduino directly. The Arduino has pins that specifically connect to the nRF24L01 to establish the SPI protocol. This circuit also has an LED

with a current limiting resistor. This LED lights up when a signal is received from the table module.

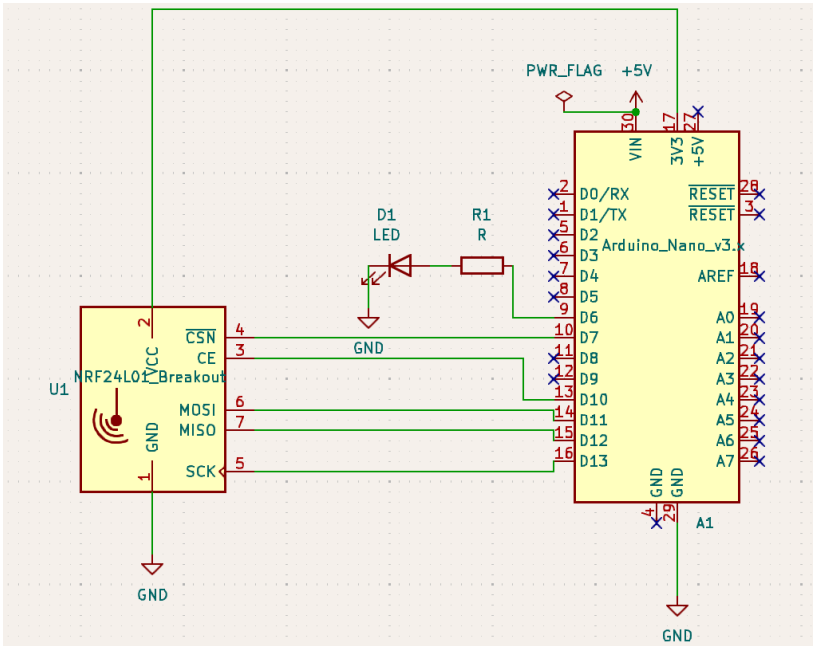


Figure 35: Final Circuit for the Receiver

Once the schematics were finalized, both circuits were implemented on a PCB. The two circuits are in Figures 36 and 37. For the transmitter circuit in Figure 36, a two layer PCB design was done. The top layer is copper and is shown in the red traces. The top layer routes all the signals. The bottom layer is also copper, shown in blue, and is used as a ground layer. This provides a low resistance path for the current to return to. In the transmitter PCB there were two traces made on the ground layer to avoid crossover with the top layers. The two traces on the bottom layer were taken in careful consideration to ensure no current paths were broken. The PCB for the receiver is much more simple and just has signals routed from the receiver to the nano. These traces were made small so the receiver could be compact to fit on the robot. The sizes of the PCB are 4 in by 2.5 for the transmitter and 3.5433 in x 1.6339 in for the receiver.

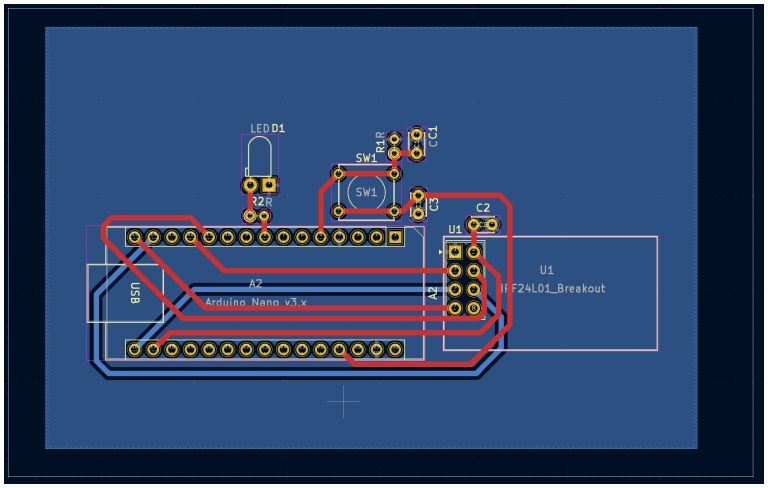


Figure 36: PCB for Transmitter

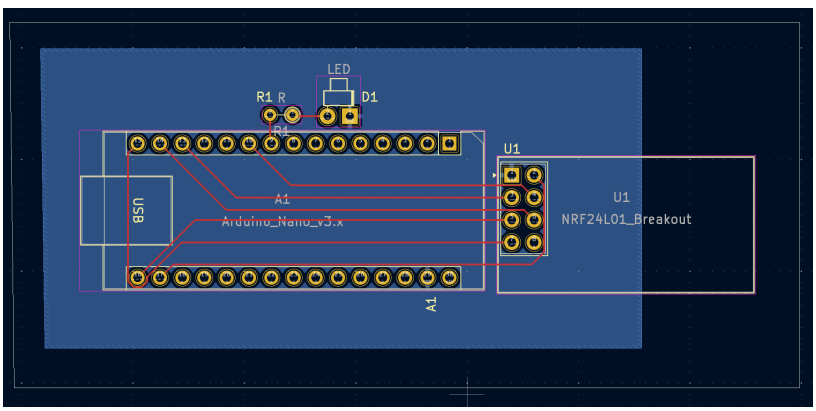


Figure 37: PCB for Receiver

Upon completion of the PCB designs, the schematics were sent to the board house for milling. The completed PCBs are in Figures 38 and 39. These are the PCBs without any components soldered on. After the components were soldered on, the final tests were conducted. The PCBs were placed into the module boxes and the button was pressed to light up the LED on the robot module. Each table module corresponds to a different LED on the robot module. The test is shown in Figure 40. The code for this test and for the UI system is in Figure 41. The code uses addressing to send signals between the modules when the buttons are pressed.

The UI system worked successfully and was able to integrate well with the rest of the robot. The components used were compatible with each other and worked well to transmit and receive signals across an open space. In the future, additions could be made to the UI system to add LEDs that also tell the customer that the robot is busy or on its way to their table. For a first subsystem prototype, the UI system worked well and is easy to use by the customer.

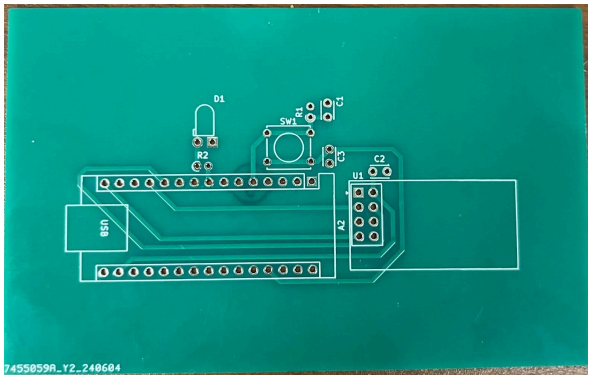


Figure 38: Transmitter PCB

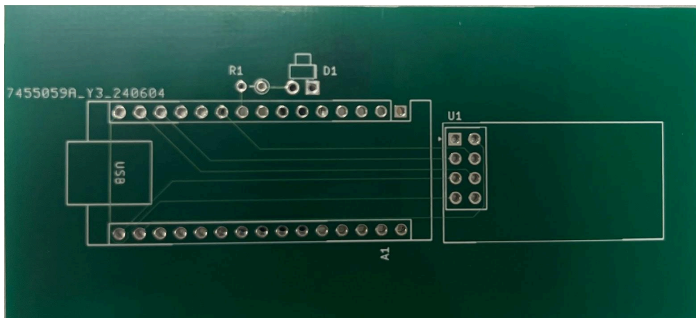


Figure 39: Receiver PCB

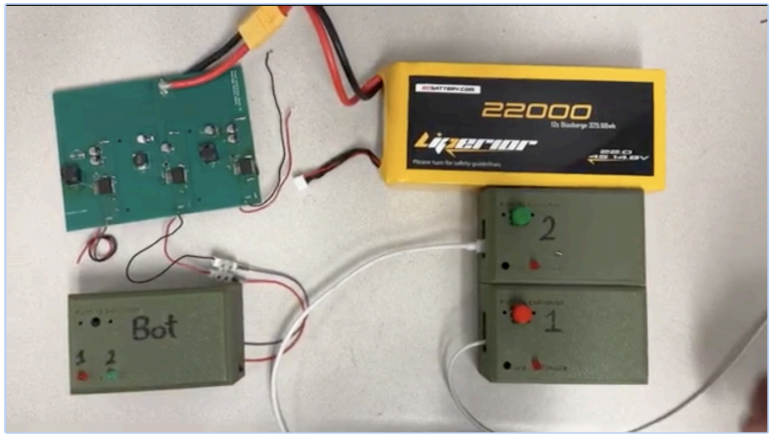


Figure 40: Final UI System Test

```
//Include Libraries
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <string.h>

const int buttonPin = 2; //pin number
int buttonState = 0;
bool mess, receieved;
const int LED = 5;

//RF24 code
RF24 radioT(9, 8);  // CE, CSN

//address through which two modules communicate.
//const byte address[5] = {0xE7, 0xE7, 0xE7, 0xE7, 0xE2}; // Address for communication, match this with transmitter
const byte address[5] = {0xE7, 0xE7, 0xE7, 0xE7, 0xE1};

void setup()
{
  Serial.begin(115200);
  //button code//
  //initialize the LED pin as an output:
  //pinMode(ledPin, OUTPUT);

  //initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
  //LED
  pinMode(LED, OUTPUT);

  //nRF code
  radioT.begin();
  //set the address
  radioT.openWritingPipe(address);
  //Set module as transmitter
  radioT.stopListening();
```

```
  //set the same channel if needed
  radioT.setChannel(100);
}

void loop()
{
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed. If it is, the buttonState is HIGH:
  if (digitalRead(buttonPin) == LOW) {
    digitalWrite(LED, LOW);
    //Send message to receiver
    //mess = notpress;
    //radioT.write(&mess, sizeof(mess));
  } else {
    const char mess[] = "Button2";
    radioT.write(&mess, sizeof(mess));
    digitalWrite(LED, HIGH);
  }
  delay(100);
}
```

Figure 41: Code for Final UI System

Power

The design and implementation of an efficient power distribution subsystem is crucial for the reliable operation of electronic systems, especially in our project which involves multiple components with varying power requirements. This sub-system focuses on developing a power distribution system for an embedded application that includes a Raspberry Pi, Arduino Nano. multiple sensors and several motors. The subsystem uses LM2678 buck converters to provide stable 5V and 12V power rails. This details the design, implementation, and testing of the power distribution subsystem, highlighting the critical design decisions and outcomes based on the high level block diagram shown in Figure 42.

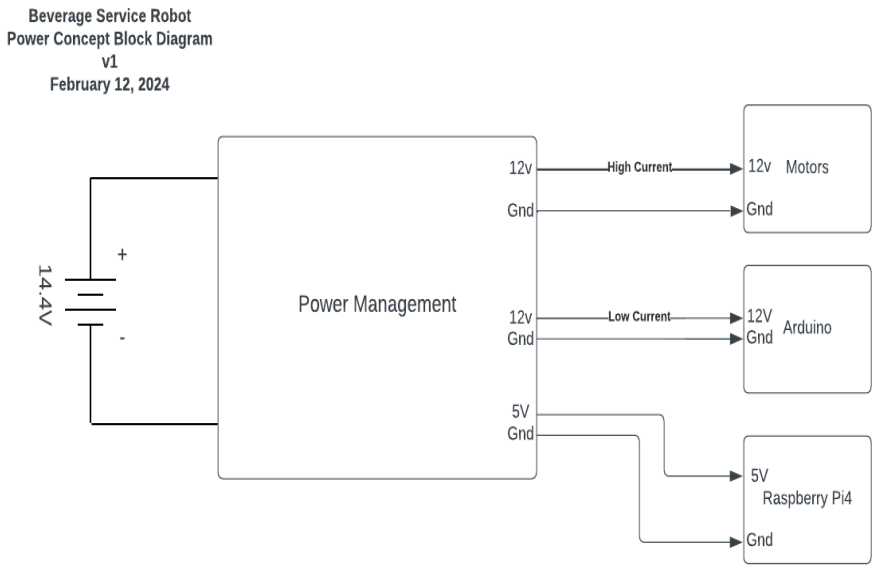


Figure 42: High Level Block Diagram of the Power System

The power distribution subsystem must meet voltage and current specifications to ensure the stable operation of the Raspberry Pi and motors. The 5V rail must supply up to 3A continuously, as well as the 12V rail handling up to 3A, maintaining voltage stability within ±5%. A 14.8V Lithium-Polymer battery with a capacity of at least 22,000 mAh is chosen as the power source, equipped with an integrated Battery Management System (BMS) to prevent overcharge, over-discharge, and overcurrent conditions. Efficiency is paramount, with each LM2678 buck converter operating at a minimum of 90% efficiency to minimize power loss and heat generation. Adequate thermal management is essential to keep all components within safe operating temperatures. To ensure safety and reliability, the system incorporates overcurrent, overvoltage, and short circuit protection mechanisms. The power distribution board (PDB) must feature a detailed schematic and optimized PCB layout, minimizing electromagnetic interference (EMI) and voltage drops while handling high current paths effectively.

Power Requirements

To make sure the performance and reliability of our power distribution board, a review of our power budget is vital (shown in Appendix 1). This power budget outlines all the power requirements needed for each component in our system. By showing these power requirements and ensuring that our power distribution board can meet these demands under all operating conditions, we can prevent power shortages and ensure the stable operation of the entire system. The power budget chart is an analysis of how much power a system or application needs to operate. It can also be defined as the maximum amount of power that's available for use. The power budget chart is divided into several sections, each playing an important role in the system's requirements. These sections include; voltage and current regulations, spectral purity/noise, current draw and operational states alongside total consumption and energy requirements.

In our project, voltage and current regulation are important for maintaining the stable operation of sensitive electronic components like motors and microcontrollers. Voltage regulation ensures a consistent voltage level despite fluctuations in load current or input voltage, which is vital for the smooth and efficient operation of motors. Without stable voltage, our motors could experience speed and torque variations, leading to low performance and mechanical stress. Our Raspberry Pi, which serves as the brains of the other devices, requires precise voltage levels to process instructions and manage peripherals. Voltage fluctuations can result in weird behavior, wrong data, or even permanent damage to the microcontroller. In our project, managing spectral purity and noise is also important for the accurate operation of sensors like LiDAR and depth cameras. These sensors rely on clean signals to show the environment

effectively. Any noise or interference can lead to perception errors, affecting our robot's ability to navigate and interact safely with the surrounding areas such as customers and tables.

Knowing the total consumption is also important for designing power supplies and ensuring the system operates effectively within its energy budget. We must account for current draw and operational states to determine total consumption. Different operational states, such as standby, active, and high-performance modes, have varying current draws. Accurately calculating the current draw for each state ensures that power supplies can handle the maximum current demand and that energy requirements are met without experiencing power shortages or failures.

Creating a power budget is essential for the reliable and efficient operation of our robot equipped with LiDAR and depth cameras. This power budget prevents power shortages and system failures while optimizing power management.

Battery Selection

With the power budget in place, the next step in the power system was selecting the appropriate battery. The battery choice is important because it directly impacts our robot’s performance, operational duration, and safety. When choosing the battery we wanted, factors such as capacity, voltage, energy density, and discharge rates have to be evaluated to meet our robot’s power requirements.

Our choice was a 14.8-volt 4S 12C LiPo (Lithium Polymer) battery with a capacity of 22,000mAh. This battery has a high energy density, making it ideal for powering components like the Raspberry Pi, LiDAR and depth camera. Additionally, it is equipped with a built-in Battery Management System (BMS) for cell safety, protects against overcharge, over-discharge, and short circuits, and extends the overall lifespan of the battery (see Appendix 3).

The 14.8-volt rating aligns well with the voltage needs outlined in the power budget, ensuring that high-demand components receive consistent and adequate power, such as the motors. The 22,000mAh capacity is for the high current demands shown in the power budget. This allows the battery to supply sufficient current during peak operational states.

The discharge rate (see Appendix 2), often referred to as the C-rating, indicates how quickly a battery can discharge its stored energy. The 12C discharge rate of the 22,000mAh 14.8V LiPo battery makes sure that our robot can handle high current demands efficiently and safely.

Voltage Regulation

With the power budget and battery selection made, the next thing to consider was voltage regulation. One method for this is through the use of LM2678 buck converters. The LM2678 is a high-efficiency step-down (buck) switching regulator that is for voltage regulation. These converters are capable of delivering up to 5A of output current and can step down input voltages to a lower, stable output voltage with high efficiency. This makes them ideal for converting the 14.8V from the LiPo battery to the various lower voltage levels required by different components within our robot, such as sensors, microcontrollers, and communication modules.

Previously, the voltage regulation method was using LM7812 and LM7805 linear regulators. These regulators are simple to use and provide stable output voltages of 12V and 5V respectively, but my thermal calculations showed that they dissipate a lot of power as heat, which is not ideal for this project (see Appendix 4). In our robot, this heat dissipation could lead to overheating issues and require additional cooling solutions.

However, the LM2678 buck converter operates with efficiencies up to 92%, minimizing energy loss during the voltage conversion process. This efficiency is important for improving

battery life, ensuring that more of the battery's stored energy is used to power our robot rather than being lost as heat. Additionally, this converter supports a wide input voltage range, making it compatible with the 14.8V input from the selected LiPo battery. Moreover, with the ability to deliver up to 5A of output current, the LM2678 supports high-power components and multiple devices simultaneously. The LM2678 also includes built-in protection features such as thermal shutdown, current limiting, and over-voltage protection. These protections save the converter and connected components from potential damage due to overheating, overcurrent, or voltage spikes, ensuring reliable and safe operation. Adding LM2678 buck converters into the power system of our robot ensures that all components receive stable and efficient power.

Power Distribution Board Design

Using KiCad, an open-source PCB design software (see Appendix 5), we developed a custom printed circuit board (PCB) to regulate the power supply for our robot. This PCB uses three LM2678 buck converters to provide three stable output voltages: two 12V outputs and one 5V output. The design involved schematic creation, component placement, and careful routing to achieve clean output voltage and current while reducing electromagnetic interference (EMI) and ensuring electromagnetic compatibility (EMC) (see Appendix 6).

The design began with the schematic shown in Figure 43, where the LM2678 buck converters were placed and configured to deliver the voltages needed. Each converter was set up according to its datasheet recommendations, alongside appropriate inductors, capacitors, and diodes.

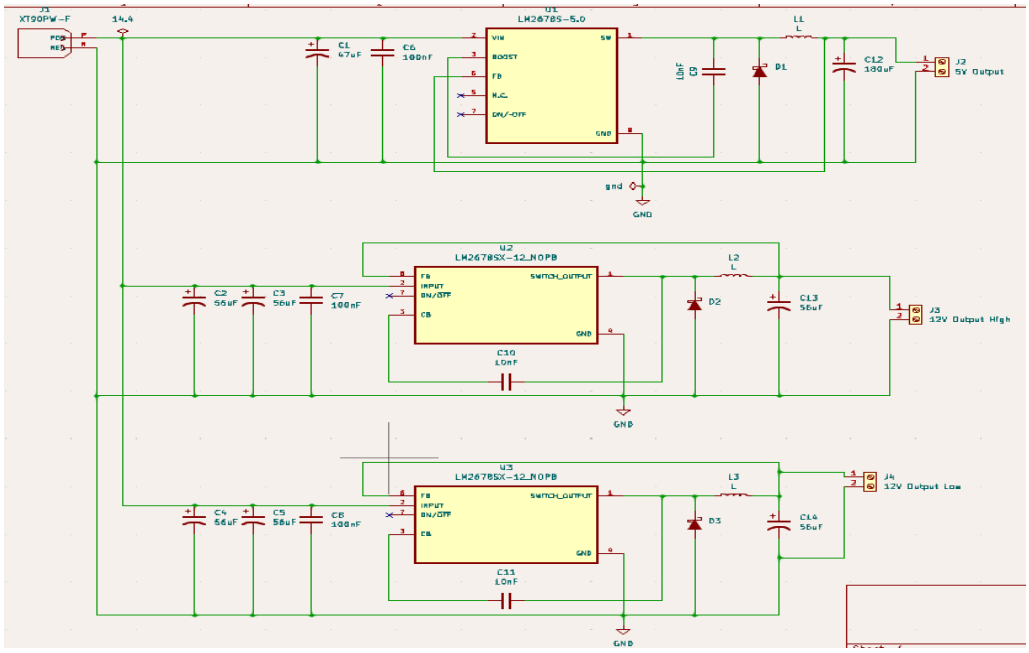


Figure 43: Schematic of Power Distribution Board

In the PCB layout editor, KiCad, components were carefully positioned to minimize high-current path lengths and reduce noise. The LM2678 converters were placed close to each other to minimize loop areas. Power planes were used for the ground (GND) and input voltage (VIN) layers, providing low-impedance paths for high-frequency currents and enhancing voltage stability. Separate power planes for the input and output voltages helped to minimize interference between different voltage domains. Power planes were routed so that the planes were wide and short to reduce resistance and inductance. A solid ground plane was used under the entire circuit (Figure 44) to provide a low-impedance return path for all signals. The use of dedicated power planes for VIN and GND significantly reduced EMI by providing low-impedance paths for high-frequency switching currents, containing noise within the power plane and preventing it from radiating and affecting other components.The resulting PCB design (Figure 45) delivered clean and stable output voltages of 12V and 5V.

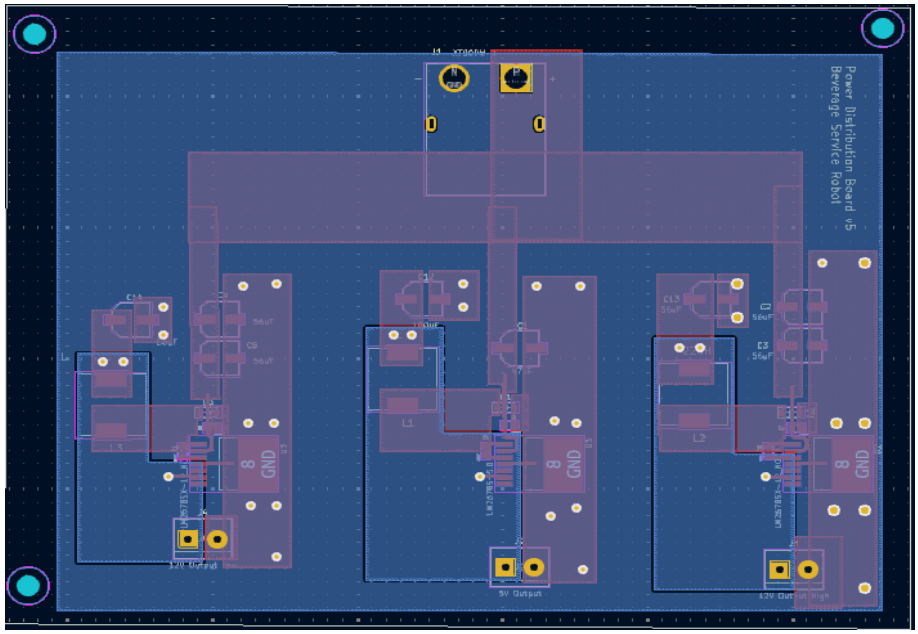


Figure 44: Solid Ground Plane of the PCB

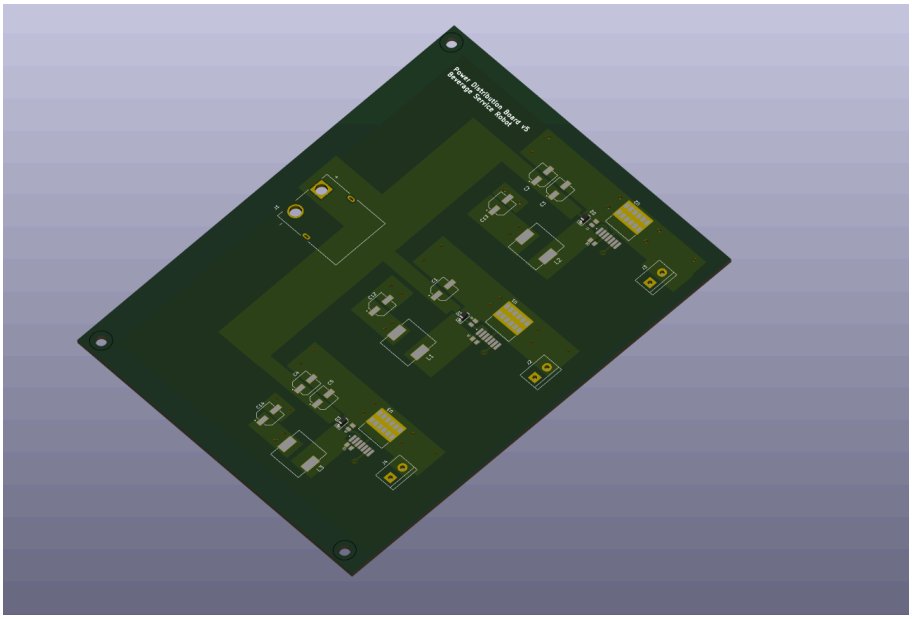


Figure 45: Final PCB Design

Testing and Validation

After designing the custom PCB with three LM2678 buck converters using KiCad, the next step was testing and validating the board to make sure it met the performance and safety

standards. This process involved a series of tests to verify the functionality, efficiency, and reliability of the power regulation system.

The testing phase began with a careful inspection of the assembled PCB (Figure 46) to check for any visible defects such as solder bridges or misplaced components.

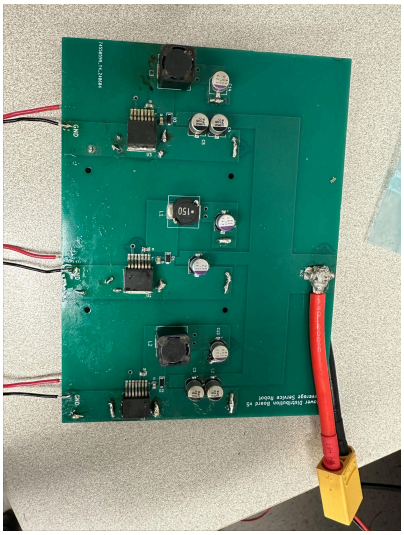


Figure 46: Assembled PCB of Power Distribution Board

After verifying the visual integrity, the board was powered up gradually using a bench power supply. Monitoring the input current and voltage during this initial power-up ensured that there were no immediate shorts or excessive current draw. The output voltages were then measured at the output terminals of each LM2678 converter to confirm they were delivering the expected 12V and 5V outputs (Figures 47 and 48).

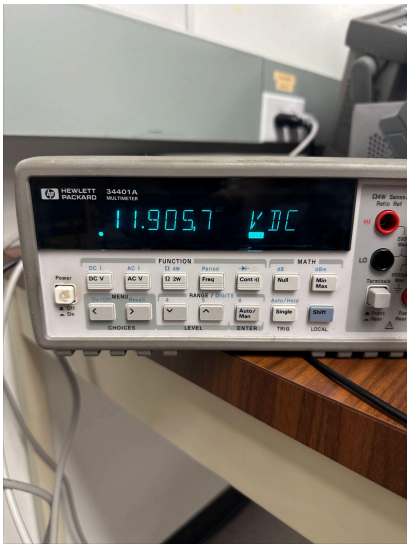


Figure 47: 12V Output of Power Distribution Board

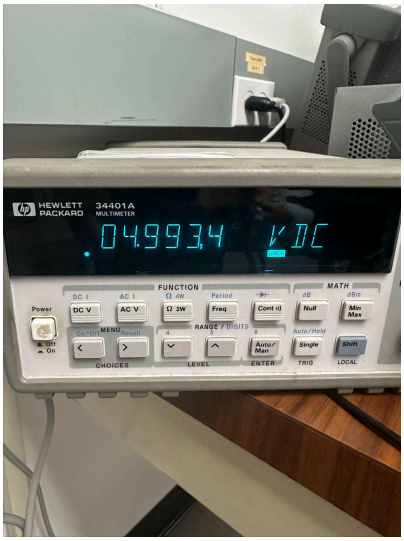


Figure 48: 5V Output of Power Distribution Board

Once basic functionality was shown, the next step involved load testing each output (Figure 49). Loads were connected to the 12V and 5V outputs, and the current draw was gradually increased to the maximum rated current of the LM2678 converters.

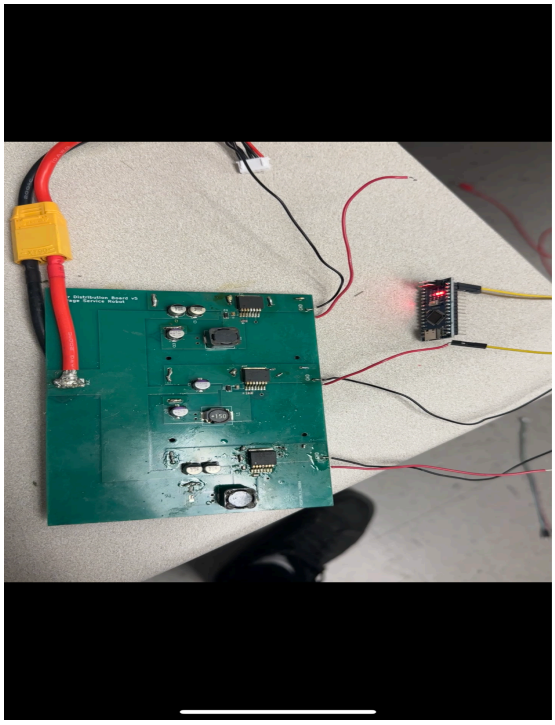


Figure 49: Load Testing PDB with an Arduino Nano

In summary, the testing and validation process confirmed that the custom PCB with LM2678 buck converters met all design requirements. Through careful initial power-up and load testing the board was validated to deliver clean, stable power efficiently and reliably. This thorough testing ensures that the autonomous robot's power system is robust, safe, and ready for deployment.

Lessons Learned

During the course of our project, several key lessons emerged that will help us in the future. One of the most significant realizations was the importance of team dynamics. We discovered that our team's initial collaboration efforts were not going to be enough to make the project work. The original lack of communication and coordination often led to misunderstandings and redundant efforts. Moving forward, it was clear that having a collaborative environment and communication within the team was crucial to improving

efficiency and achieving better outcomes. Thus, we were able to come to a set state and ultimately, end the project on a good note.

On the technical side, we learned that the subsystems of the project were far more time-consuming than initially expected. Each subsystem required a deep level of understanding and insane attention to detail, which we underestimated. This forced us to adapt our approach and allocate more time to these areas. In future projects, it will be important to conduct a more thorough analysis of the technical requirements and ensure that sufficient time is allocated to address situations like this.

Furthermore, we experienced the challenges of tackling new learning curves under tight time constraints. Learning new softwares and methodologies while also sticking to a strict timeline proved to be a significant challenge. This situation showed the importance of planning for adequate learning periods, as well as being realistic about the time required to become proficient in these areas.

In conclusion, our project provided insights into the importance of effective team collaboration, realistic technical planning, and managing learning curves under pressure. By using these lessons, future projects can be approached with a more informed and strategic mindset, ultimately leading to smoother execution and more successful outcomes.

Next Steps

Looking ahead, our next steps will focus on fully integrating the UI modules with our robot, enabling full automation of the system. In the current state of the robot, it is fully autonomous moving after a map is created and can be moved to set waypoints discussed in the software architecture section. The fully autonomous movement with setting waypoints on a decision tree when a UI table module is pressed has yet to be implemented. It would be best for

the robot to receive a transmission from different tables and autonomously navigate to it. A decision tree would allow us to previously set locations of each table on the map which corresponds to each table module. This integration is a milestone that will enhance the robot's functionality and user experience. Additionally, we plan to improve our sensors and add more sensors specifically for obstacle avoidance to prevent the robot from falling down stairs or getting stuck undercover hangs. These will boost the robot's navigation and ensure safer operation in dynamic environments.

Moreover, we know the need for the robot's casing to be more suitable for the environments it operates in. Therefore, we will redesign the casing to be more robust and adaptable to various conditions, enhancing the robot's durability, performance and overall its appeal to the eye.

Another step is the enhancement of our power distribution board. We will incorporate battery monitoring systems and develop methods for charging without disconnecting the battery. These will ensure more efficient power management and reduce downtime, thereby increasing the overall reliability and effectiveness of the robot.

Conclusion

In this project, we have successfully designed and implemented an autonomous beverage serving robot. This robot is able to work alongside servers in a restaurant setting to allow for more efficient service delivery during peak hours. The project has contributed to the service industry by introducing an improved way of serving drinks that can not only minimize contamination but aid workers. This benefits both the customer and the restaurant. The testing process ensured that our solution would meet the required standards and perform under a variety of conditions. This project not only showcases a practical application of concepts and theory

learned throughout our academic journey but also provides an innovative solution to problems in the restaurant industry.

Appendix 1

Power Budget

Beverage Server Robot on a 12hour Cycle Working Concept v1.1 3/8/24

Item	Value	Quantity	Nominal Voltage (V)	Needed Input V Regulation	Output V Variation	Spectral Ripple/Noise	State Based Current (mA) High	Low	Off	Time in State (hr/100% = 12hrs) High	Low	Off	Total Consumption (mA-hr)	Max Power (mW) (for design reference)	Total Energy (hr) (mW-hr)	Design Notes
Battery (source)	LiPo (XTX)	1	14.8	N/A	10.5 low 16.8 max	N/A										provides 22,000 mA-H
Microcontroller	Raspberry Pi 4	1	5.1	4.75 min 5.25 max	5v +-1% 3.3v +-1%	±125mV	1520	600	30	10	2	0	4529	7600	30000	
Arduino Uno		1	5	4.2 min 12 max	5V±4.100 3.3V±1%	0.00625 RMS	73.19	26.07	11.65	9	3	0	739.32	1000	5000	
Motor	AM4313	2	12	10.8 min 13.2 max	N/A	N/A	500	325	200	9	3	0	5475	1400	33600	
Motor Controller	DRX2002	1	12	4.0 min 40 max	0 low 12 max	N/A	2000	35	0	9	3	0	4500	12000	54000	
LiDar	DFR0315	1	5	3.6 min 5 max	N/A	N/A	300	60	0	10	0	2	3000	N/A	N/A	
Depth Camera	D435 Lite	1	5	5 Regulated	N/A		500	0	0	5	0	2	4500	N/A	N/A	These two components will drain power equally to the arduino
5V Buck Converter	LM2575-5	1	5	5 min 10 max	±.02V	±.2mVrms	6	0.5	0	10	2	0	61	4200	21000	
12V Buck Converter	LM2576-12	2	12	12 min 19 max	±.02V	±.76 mV	6	0.5	0	10	2	0	61	5000	60000	
3.3V Power Rail	Use Output from Arduino		3.3	N/A	±4.4%	30mV (RMS)	Dropout 1.1-1.5 V									not clear with Arduino current. Max output current from Arduino is 200mA from pin voltage 3.3 max. 50mA, however typically provides 50mA+. So please use 200mA+ w/ LDS (typically around 50mA+ as well
5V Power Rail	Use Output from Arduino		5	N/A	±4.4%	30mV (RMS)										
Transceiver	NRF24L01	1	3.3	1.9V min 3.6V max	N/A		11.3	0.022	0.00009	2	10	0	22.02	60	198	
										Totals:			22809.14	31760	207048	

Battery Capacity (mA-hr)	
14270.333333	14.0 volt battery

https://docs.google.com/spreadsheets/d/1SeY7WZJ9NSSD3kqYm-m037uhsIx1kaEV5lcNG-x0x0c/edit?usp=sharing

Appendix 2

Why is discharge rate so important?

1. Handles Peak Currents

Robots often encounter situations where they need to draw a significant amount of power in a short period. For example, rapid acceleration, heavy data processing, or operating multiple sensors simultaneously can require a surge in power. A high discharge rate ensures that the battery can provide this burst of energy without voltage drops or power shortages, maintaining our robot's performance and stability.

2. Efficient Power Delivery

High discharge rates allow for efficient power delivery to high-drain components such as motors, LiDAR systems, and depth cameras. These components may have substantial power requirements during intensive operations. The ability to meet these demands ensures that our robot can function optimally without compromising on speed, responsiveness, or accuracy.

3. Maintaining Voltage Stability

A battery with a high discharge rate maintains a more stable voltage under load. Voltage drops can occur if the battery cannot supply the required current, leading to erratic behavior or performance issues in sensitive electronics. By providing a stable voltage, the high discharge rate ensures consistent operation of all robotic systems, including critical navigation and processing functions.

4. Improved Safety and Longevity

Operating within the safe limits of the battery's discharge rate prevents overheating and potential damage to the battery cells. The built-in Battery Management System (BMS) also helps

manage this by monitoring and regulating discharge rates, further enhancing safety and extending the battery's lifespan.

5. Enhanced Performance in Dynamic Environments

In customer-facing environments, autonomous robots must react swiftly and reliably to dynamic conditions. Whether navigating around obstacles or responding to user interactions, our robot requires immediate and robust power delivery. A high discharge rate ensures that our robot can adapt quickly and perform effectively, providing a seamless user experience.

Appendix 3

Battery Monitoring System (BMS)

The primary functions of a BMS include:

1. Balancing the voltage and state of charge (SOC) of individual cells within a battery pack. This helps maximize the overall capacity and lifespan of the battery.

2. Monitoring the voltage of each cell and disconnecting the charging source when any cell reaches its maximum safe voltage, preventing overcharging and potential damage.

3. Monitoring the voltage of each cell and disconnecting the load when any cell reaches its minimum safe voltage, preventing over-discharge and potential damage.

4. Monitoring the temperature of the battery pack and individual cells, and taking measures to prevent overheating, which can lead to thermal runaway and safety hazards.

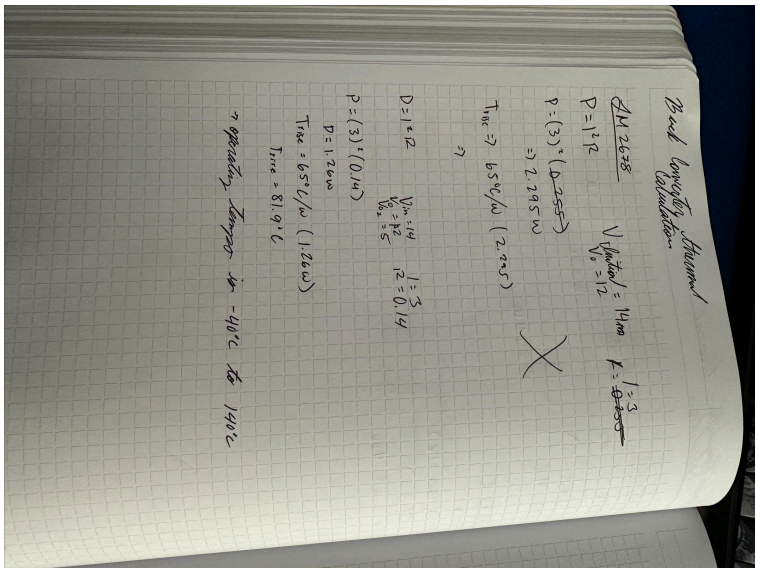
5. Detecting and preventing short circuits within the battery pack, which can lead to excessive current flow and damage to the battery or connected devices..

Appendix 4

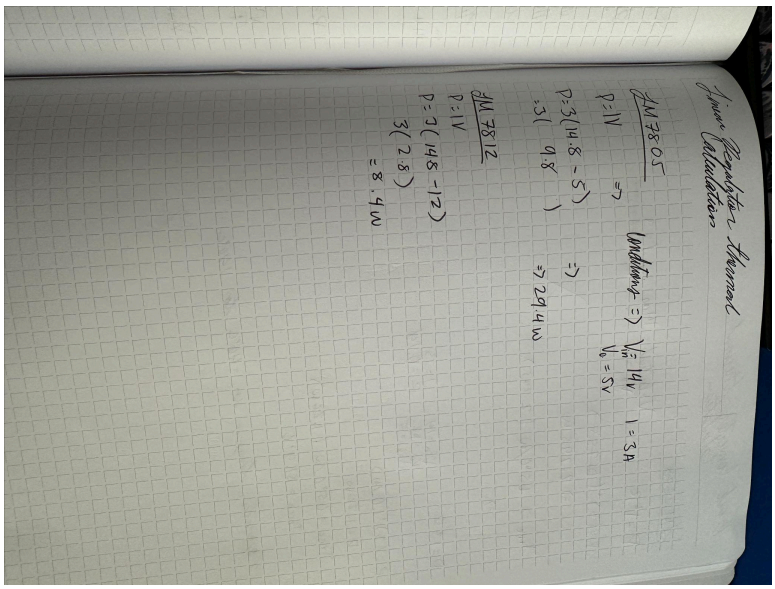
Thermal Calculations

Below are the thermal calculations for all three power converters; LM2678 buck converter, LM7805 and lastly the LM7812.

;



Appendix 4.1: Thermal Calculations for Buck Converter (LM2678)



Appendix 4.2: Thermal Calculations for Linear Regulator (LM7805 and LM7812)

Appendix 5

KiCad

KiCad is an open-source software suite for electronic design automation (EDA), primarily used for designing printed circuit boards (PCBs).

The key features of KiCad include:

1. KiCad allows users to create schematic diagrams of their electronic circuits, including symbols for components and connections between them. The schematic editor is intuitive and supports hierarchical design, allowing for complex circuits to be easily managed.

2. KiCad's PCB layout tool enables users to design the physical layout of the circuit board. Users can place components, route traces, define board outlines, and add text and graphics to create a complete PCB design.

3. KiCad includes a 3D viewer that allows users to visualize their PCB designs in three dimensions. This feature is useful for verifying component clearances, checking for mechanical conflicts, and creating realistic renderings of the final product.

4. KiCad includes a DRC tool that checks the PCB design against a set of user-defined rules to ensure that the design meets manufacturing requirements.

Appendix 6

EMI and EMC

Electromagnetic Interference (EMI) and Electromagnetic Compatibility (EMC) are crucial aspects of electronic design, especially for devices like autonomous robots that operate in customer-facing environments. EMI refers to the disturbance that affects the performance of an electronic circuit due to electromagnetic radiation or conduction. EMC, on the other hand, is the ability of a device to function properly in its intended electromagnetic environment without causing interference.

The key concepts related to EMI and EMC in electronic design include:

1. Electromagnetic signals emitted from a device that can interfere with other nearby electronic devices. Radiated emissions are typically controlled through proper PCB layout, grounding techniques, and the use of shielding materials.
2. Proper shielding and grounding techniques are essential for minimizing EMI and ensuring EMC. Shielding materials such as conductive enclosures and grounding techniques such as star grounding can help reduce interference and improve overall system performance.
3. The placement of components on a PCB can impact EMI and EMC. Placing sensitive components away from high-speed or high-current traces, and ensuring proper routing and separation of signal lines can help minimize interference.