



Bartender Butler Bot: B3

Final Design Report

Group 13

Corey Scott	co079592
Edward Nichols	ed891610
Yianni Babiolakis	yi657277
Rachael Caskey	ra231875

Electrical Engineering
Electrical Engineering
Electrical Engineering
Electrical Engineering

Table of Contents

1. Executive Summary	6
2. The Bartender Butler Bot	7
2.1. Motivation	7
2.2. Design Objectives	8
2.3. Requirements Specifications	9
2.3.1. Total Appliance Cost	10
2.3.2. Bartender Configuration Time.....	11
2.3.3. Ingredient Capacity.....	11
2.3.4. Order Time	11
2.3.5. Beverage Mixing Time	12
2.3.6. Drink Accuracy	12
2.3.7. Delivery Time.....	12
2.3.8. Delivery Range	13
2.3.9. Charge Lifespan	13
2.3.10. Obstacle Collision Rate	13
2.3.11. Bartending Bot Dimensions	13
2.4. House of Quality	14
2.4.1. Analysis	15
2.4.2. Notes on Exemplary Trade-offs	18
3. Background & Technical Research	19
3.1. Contemporary Projects & Products	19
3.1.1. SirMixABot.....	19
3.1.2. Automated Bartender	21
3.1.3. Bartendro.....	23
3.1.4. Elliot the Line Follower Robot	25
3.1.5. RaspRobot OpenCV Project.....	26
3.2. Relevant Components & Key Technologies	27
3.2.1. Arduino Uno/Mega.....	28
3.2.2. ESP8266	28
3.2.3. Raspberry Pi.....	28
3.2.4. Peristaltic Pumps.....	29
3.2.5. H-Bridge DC Motor Drivers.....	29

3.2.6.	Relay Modules	30
3.2.7.	Infrared Photodiode Sensors	31
3.2.8.	Hall Effect Sensors	32
3.3.	Market Analysis	Error! Bookmark not defined.
3.3.1.	Typical Consumer Archetype	Error! Bookmark not defined.
3.3.2.	Value Proposition.....	Error! Bookmark not defined.
3.3.3.	Commercial Roadmap	Error! Bookmark not defined.
4.	Standards & Regulations	35
4.1.	Health & Safety Standards.....	35
4.1.1.	Vending Machine for Food and Beverages.....	35
4.1.2.	Drinking Water System Components.....	35
4.1.3.	Fire Safety and Emergency Symbols.....	36
4.2.	Digital Standards	36
4.2.1.	Wireless Communication Standards.....	36
4.2.2.	OASIS standards- MQTT v5.0	38
4.3.	Key Design Constraints	39
4.3.1.	Economic and Time Constraints	39
4.3.2.	Environment, Social, Political Constraints.....	40
4.3.3.	Ethical, Health, Safety Constraints.....	40
4.3.4.	Manufacturability and Sustainability Constraints.....	41
5.	System Design & Phased Implementation.....	43
5.1.	Functional Description.....	44
5.2.	System Conceptual Block Diagram	45
5.3.	Strategic Components & Parts Selection.....	47
5.3.1.	Bartender.....	47
5.3.1.1.	Bartending Firmware Host.....	47
5.3.1.2.	Dispensing Fluids	50
5.3.1.3.	Proximity Sensing & Ranging.....	57
5.3.2.	Butler	61
5.3.2.1.	Autonomous Navigation	61
5.3.2.2.	Securing & Validating Cup.....	65
5.3.3.	Core Application.....	67
5.3.3.1.	Application Backend Environment.....	68
5.3.3.2.	User Interface.....	72

5.3.4.	Structural Components and Aesthetics	77
5.3.5.	Parts Selection Summary	77
5.4.	Phase I: Core Functionality	78
5.4.1.	Bartender: Simple Cocktails	Error! Bookmark not defined.
5.4.1.1.	Architecture	Error! Bookmark not defined.
5.4.1.2.	Functional Block Diagram	Error! Bookmark not defined.
5.4.1.3.	Integration Strategy & I/O Summary	Error! Bookmark not defined.
5.4.2.	Butler: Line-Following	Error! Bookmark not defined.
5.4.2.1.	Architecture	Error! Bookmark not defined.
5.4.2.2.	Functional Block Diagram	Error! Bookmark not defined.
5.4.2.3.	Integration Strategy & I/O Summary	Error! Bookmark not defined.
5.4.3.	Core Application: System Integration	Error! Bookmark not defined.
5.4.3.1.	Architecture	Error! Bookmark not defined.
5.4.3.2.	Functional Block Diagram	Error! Bookmark not defined.
5.4.3.3.	Integration Strategy & I/O Summary	Error! Bookmark not defined.
5.5.	Phase II: Advanced Features	92
5.5.1.	Bartender: Scaling Up.....	93
5.5.1.1.	Advanced Functional Description.....	93
5.5.1.2.	Advanced Architecture	93
5.5.2.	Butler: Path Recognition & Obstacle Avoidance	93
5.5.2.1.	Advanced Functional Description	Error! Bookmark not defined.
5.5.2.2.	Advanced Architecture	Error! Bookmark not defined.
5.5.3.	Core Application: GUI Aesthetics	Error! Bookmark not defined.
5.5.3.1.	Advanced Functional Description	Error! Bookmark not defined.
5.5.3.2.	Advanced Architecture	Error! Bookmark not defined.
5.6.	Phase III: Commercial Optimization	Error! Bookmark not defined.
5.6.1.	Design for Manufacturability	Error! Bookmark not defined.
5.6.2.	Market Niche Optimization.....	Error! Bookmark not defined.
6.	Prototype Construction & Coding.....	Error! Bookmark not defined.
6.1.	Integrated Schematics	Error! Bookmark not defined.
6.2.	PCB Manufacturing Documentation.....	Error! Bookmark not defined.

6.2.1. PCB Properties	Error! Bookmark not defined.
6.2.2. Vendor Information	Error! Bookmark not defined.
7. Prototype Validation Plan.....	95
7.1. Bartender Prototype Testing	Error! Bookmark not defined.
7.1.1. Photodiode Sensor Testing.....	Error! Bookmark not defined.
7.2. Butler Prototype Testing.....	104
7.2.1. Valid Drink Identification Testing.....	105
7.3. Core Application Testing	Error! Bookmark not defined.
7.3.1. Photodiode Sensor Testing.....	Error! Bookmark not defined.
8. Administrative Content.....	112
8.1. Tasks and Responsibilities	112
8.2. Budget and Financing.....	113
8.2.1. Estimated Budget.....	113
8.2.2. Current Expenditures	114
8.3. Project Milestones.....	116
8.3.1. Design Phase	117
8.3.2. Implementation Phase	118
9. Appendices.....	120
9.1. Appendix A - References	120

1. Executive Summary

As a formulation and scoping of a senior design project for the 2019-2020 academic year at UCF's College of Electrical Engineering and Computer Science, we propose and introduce the **Bartender Butler Bot (B3)**. This document renders an ABET mandated composition, to record the efforts dedicated to the research, design, development, and ultimate implementation of an automated beverage mixing and delivery home appliance.

This project serves to meet a clearly defined set of goals for an innovative solution to optimize an experience that is nearly universal in its popularity. The Bartender Butler Bot (B3) is a robot made up of a drink-mixing station (the Bartender) and a remote unit (the Butler) which can take a user's order, send it to the Bartender, and deliver the drink back to the user. The prototype developed over this course will demonstrate a strictly limited ability for usage an average home, as a proof of concept – to lay the groundwork for a more developed appliance that could be used to support hosting at private events.

Outside of the direct applications of the finished product, this project serves as both a learning experience of many skills within the field of electrical engineering for its designers and a demonstration of those which have already been developed over the course of their undergraduate studies. The finished design of the B3 will make use of multiple subsystems which communicate using multiple standards, as well as several sensors and forms of input that will be tested individually before integrating into a finished product, which will be adapted to meet the requirement specification laid out in this report. The report will not only be a summary of the design the finished product exhibits, but will also serve as a timeline and history of the project itself, including the initial plans for the B3, the problems encountered throughout their implementation, and the tools used and discovered in developing solutions to these problems.

2. The Bartender Butler Bot

This section provides a formal introduction of the Bartender Butler Bot (B3). It begins with a narrative of intrinsic motivation followed by an enumeration of natural qualitative design objectives that arise from the motivations explored. From the concepts formed, a Six Sigma “House of Quality” diagram is constructed and the key tradeoffs in the design process are identified. An analysis of the tradeoffs is conducted to then establish a prioritized and multi-faceted approach to the engineering design process.

2.1. Motivation

Since the Industrial Revolution, the relationship between mankind and technology has been one of automation, of developing new devices which can simplify or replace altogether seemingly mundane or repetitive tasks to free ourselves and our time for bigger and better things. As time has passed, this concept has expanded from optimizing our places of work to our very homes, with so-called smart devices creating entirely new definitions of convenience in the modern world. One form of personal automation that has remained popular since early speculative science fiction has been the concept of a robotic butler or assistant, able to perform small tasks around the house for families. We are living on the cusp of such automated conveniences, with devices such as Amazon’s Alexa or Google Home having control over minor appliances such as light bulbs and locks. Inevitably, most aspects of human social interaction will be supported by robotic assistants, and it is hence natural that the following represent a sufficient motivation for a baccalaureate’s demonstration of exemplary engineering practice in that vein:

For millennia, and all over the world, mankind has gathered over the social bond of sharing drinks with one another. Entertaining others in one’s home, sharing warm stories and making bonds over cold drinks is so fundamentally ingrained in our culture that for many adults both young and old, it is a welcome and yearned-after end to a long workweek. Though part of the beauty in this unifying interaction is in its timelessness and simplicity, one can’t help but wonder why modern technology has not yet been used to facilitate the experience – especially for the dutiful host.

Unlike in a bar or restaurant, where the dedicated wait staff can quickly take refill orders, the lively stories and deep conversations that come to be while freely, openly, and directly engaging a group at home must be put on hold while the chivalrous host walks back to their bar to refill everyone’s drinks. They return to the possibility that the punchline has been delivered, the close game has been won, the heated debate lost – or worse, that the collective train of thought has been completely derailed. With this in mind, this project aims to bring the convenience of modern technology to serving and entertaining guests at a warm private event.

The overarching goal of this project is to design, build, and test a low-cost, proof-of-concept home appliance made up of two major parts: A “Bartender” bot to be able to remotely dispense mixed drinks as selected by the user, and a “Butler” bot to transport these drinks to the user without the need for them to leave the common area. By automating the process of refilling and transporting people’s drinks, this device can be used to facilitate the experience of hosting and entertaining guests with the luxury of a bartender and a waiter, all in the comfort of one’s own home.

2.2. Design Objectives

This section explicitly enumerates qualitative design objectives for the Bartending Butler Bot (B3). Design objectives are drawn from both the explicit motivation above and from the genuine technical aspirations intrinsically posited by the team. Provided first that the project, as a baccalaureate capstone, is limited in scope, natural overall objectives are deduced. Then, simple but explicit qualitative design objectives are generated for each major component of the B3, so as to yield discernable design boundaries to support and constrain design efforts.

Overall:

- Construct an electronic appliance which automates the process of mixing and delivering a beverage across a predefined path.
- Minimize the number of mechanical subsystems, such as to enable or otherwise simplify the manufacturing or construction of the overall assembly – especially as it relates to the achievement of mechanical functions.
- Maximize the number of off-the-shelf or open-source subsystems, such as to enable or otherwise simplify the design and development process – especially as it relates to rapidly implementing a functional prototype.
- Minimize the overall cost of the system, such as to practically heed the natural budget constraints of an undergraduate project and to support the potential for the commercialization of a more advanced prototype.

Bartending Bot:

- The bartending bot shall be able to dispense drinks made up of any of the individually configured ingredients combined across a range of proportions, as may be desired by the user – especially such as in preconfigured recipes.
- The bartending station’s available recipes shall be constrained to those whose beverage ingredients are both available and configured across the appliance’s core user interface.
- The bartending station shall be optimized for simple cleaning and quick configuration, such as to minimize the amount of time required by the host in maintaining it.
- The bartending station shall provide information on its usage and status so as to enable the compilation of an active and live inventory of its configured ingredients – visible to the host on the user interface upon request.

- The bartending bot shall have a form factor optimized or otherwise designed for placement on a countertop or tabletop environment, such as a contemporary kitchen appliance might.

Butler Bot:

- The butler shall be able to physically transport mixed beverages from the bartending station to an exact predetermined remote location.
- The butler shall be able to autonomously navigate a strictly defined path from the bartender to the delivery location without contacting any obstacles, damaging itself and losing its cargo.
- The Butler shall receive the order on an embedded user interface and communicate it to the Bartender.

Core Application & GUI:

- The graphical user interface (GUI) for configuring both the Bartender and the Butler shall be accessible at least across a dedicated touch input panel on the Bartender.
- The core appliance software shall automatically handle and suggest drink recipes from the available ingredients configured by the user.
- The core appliance software shall allow the user to configure recipe preferences, and ingredients available.
- The core appliance software shall mediate interactions and control data transfer between the two mechanical system control firmware on the Butler and Bartender, respectively.
- The core appliance software shall directly control the mechanical system control firmware on the Butler and Bartender, respectively.

2.3. Requirements Specifications

The following section explicitly states quantitative design objectives which arise as measurable quantities born from the qualitative objective specified previously – listed summarily in **Table 2–A**. These requirements shall be used as the benchmark to validate the successful achievement of the project’s operational design objectives. The specific value of each requirement is chosen depending upon the nature of the requirement itself; whereas, the reason for explicitly stating and observing these particular specifications as boundary conditions to the design process are that they are the minimum set necessary to achieve the functional design requirements demanded by aforementioned motivations for the project.

Description	Unit	Requirement
Total Appliance Cost*	USD	<\$1500
Max. Avg. Bartender Configuration Time	Minutes	<10
Min. Capacity for Beverage Ingredients*	Ingredients	>3
Order Time	Minutes	<1
Beverage Mixing Time*	Minutes	<2
Drink Accuracy	% Target Vol.	<5%
Delivery Time*	Minutes	<5
Delivery Range*	Meters	<10
Charge Lifespan	Minutes	>90
Obstacles Collision	Num./journey	<3
Dimensions of Bartender	cm ³	<50000

Table 2–A: Engineering Design Specifications (EDS)

2.3.1. Total Appliance Cost

This specification refers to the total financial expenditure required for a single final assembled prototype – not including expenses related to development, design, or the iterative process therein. The key motivation for the assertion of this requirement specification is the nature of the project as a self-motivated, self-funded, self-organized undergraduate exposition of technical knowledge. A team of average undergraduates has naturally limited financial resources – and so the final prototype must be within the scope afforded by the team’s combined financial resources.

Further, and equally as important, the potential for commercialization at a later stage of development would be supported best by aiming for a price-point nearabout what might be acceptable for the archetype of the target consumer – such as those who may purchase a luxury espresso machine. This is expressly an intrinsic objective to minimize the manufactured “cost of goods sold”.

To this end, a **maximum value of 1500 USD'19** was – by team consensus – a reasonable value for a ceiling limit on total expenditures for the first prototype. This value naturally constrains the scope of development and represents a hard barrier observed across all aspects of the design.

2.3.2. Bartender Configuration Time

This specification refers to the time it will take the user to configure the Bartender for typical usage – such as after a maintenance cycle or for initial setup. This only includes defining both which ingredients are available and how much of each is remaining for the Bartender to use.

To this end, the **maximum average configuration time of 10 minutes** is a compromise between realism and convenience of implementation. The value represents an achievable upper ceiling on the demands from the owner or maintainer of the home appliance – and that the value represents a concrete, but seemingly attainable design objective.

2.3.3. Ingredient Capacity

This specification refers to the minimum number of beverage ingredient options that can be configured on the B3, such that they are available to the Bartender for dispensing at any given moment. This alludes to the number of actively controlled pumps which must be controlled by the Bartender.

A **minimum of 3 simultaneous ingredients** for the Bartender was selected as an attainable benchmark to demonstrate a proof-of-concept ability to mix drinks of varied recipes through the precise individual control of multiple pumps. This number was, in part, chosen as many common cocktails contain 2-3 ingredients.

2.3.4. Order Time

This specification refers to the maximum average total amount of time it takes a user in a typical usage scenario to navigate the user interface's menus, peruse the options, and make a drink selection.

A **maximum order time of 1 minute** was selected based on the assumption that the user interface should be efficient and not detract from the user's experience, allowing them to order their beverage quickly enough such that the user observes no major interruption in conversation or presence. This reflects the belief of the design team that putting one's conversation on hold for more than a minute does not lend itself to a seamless interaction with an attendant – especially over the course of multiple rounds. This also relates directly to the abstract objective of minimizing the amount of time between ordering a beverage and receiving it.

2.3.5. Beverage Mixing Time

This specification refers to the maximum of the average amount of time required to mix a beverage – after receiving an order and initiating the fluid dispensing process at the Bartender.

A **maximum beverage mixing time of 2 minutes** was selected to support the minimization of the overall amount of time from ordering to receiving a drink – an abstract objective that is confounded variously. This value, along with Order Time and Delivery Time, had to each be selected with regards to one another in order to keep the overall time from starting an order to receiving a drink reasonably low during the entire process.

2.3.6. Drink Accuracy

This specification refers to the maximum range of tolerance in the specific volume of an ordered and delivered beverage – including the deviations in each ingredient’s specific pour, and the loss encountered on the journey, if any.

A **volumetric drink accuracy of $\pm 5\%$** of the target volume was selected in order to maintain both consistency in the beverages produced and a safeguard on alcoholic content, as over-pouring can become a health and safety risk. It was assumed that, whereas individual pours of ingredients in a commercial food & beverage environment are standardized to between 1.5-2oz, a maximum volumetric deviation of 5% aligns with a typical user’s threshold to discern variation in the composition or flavor of a beverage.

2.3.7. Delivery Time

This specification refers to the maximum average amount of time required to physically transport a complete and dispensed beverage from the Bartender to the Butler’s initial location, where the user submitted the order, at the maximum range of delivery range of the Butler. This travel time is reasoned to be the maximum of either the fetch or return journey, and thus represents the target of development efforts – whereas, achieving the target for one intrinsically achieves the target for the other half of the journey. This objective also encapsulates a requirement for minimum avg. velocity along the predefined route.

A **maximum delivery time of 5 minutes** was selected – by consensus – as a reasonable pace for mess-free and collision-free navigation within the context of minimizing an overall “order submission to beverage receipt” time. This value was selected considering the expected max speed of the Butler, the added weight of its total serving load, and the selected maximum delivery range. Further, consideration was paid to service times in a commercial food and beverage environment.

2.3.8. Delivery Range

This specification refers to the maximum target range for a route across a level, nominally unobstructed surface in operation – such as while retrieving a beverage. To be clear, this is also a scope-limiting target on the total one-way translation distance the Butler may have to traverse.

A **minimum delivery range of 10 meters** was selected in order to support the fundamental utility of the device, as the express purpose of the Butler unit is to deliver a beverage from the remote Bartender to the user. Though navigation is a repeatable process, it has many possible confounding variables. The range is an attainable value conceivably near the lower boundaries of what might be expected for a useful route between the user's entertainment center and their choice of location for a Bartending station.

2.3.9. Charge Lifespan

This specification refers to the minimum amount of active time the B3 can continuously handle orders without a dedicated charging period – including taking an order, retrieving a beverage, and returning to standby.

A **minimum single-charge lifespan of 90 minutes** was selected to support the device's practicality in usage across the duration of a private social gathering. This requirement encapsulates an implicit objective to minimize power consumption with power management strategies or otherwise supply ample power reserves.

2.3.10. Obstacle Collision Rate

This specification refers to the maximum average number of unintended collisions with obstacles in or near the predefined route for one half of the route – especially including those that cause a minor spill or otherwise impede the ability of the Butler to subsequently navigate. This alludes to the sophistication of the autonomous navigation strategy that will be employed in the Butler and sets an implicit objective for maximizing it.

A **maximum collision rate for each one-way journey of 3** sets an attainable objective that limits the scope of sophistication required in the navigation strategy, but also yields a decent foundation for future development efforts.

2.3.11. Bartending Bot Dimensions

This specification refers to the maximum volume that the Bartending station may occupy, without specifying or restricting the form of its packaging. This volume includes the pumps, electronic hardware, structural framework, outer packaging, and all supporting systems otherwise required – but not the bottles beverage ingredients which may yet be varied in size and shape.

The **maximum overall volume of the Bartender is 50,000 cm³**, to ensure this portion of the device could reasonably be kept in the user's kitchen, dining room, or location of choice without taking up an amount of space beyond what might be expected of a common contemporary counter-top appliance, such as a microwave or espresso machine.

2.4. House of Quality

To enable an analysis of the various competing requirements, to narrow the scope for focus on aspects of the engineering design process, and to maximize development efficiency it is useful to integrate the design objectives and engineering requirement specifications into a matrix of key engineering considerations which may then be related against the motivating party's own qualitative or quantitative objectives. In other words, approaching the engineering requirements from the summary perspective of key engineering considerations yields a useful correlation matrix from which one can plot a design approach and later gauge a current design status.

Due to the fact that some of the "customer's" requirements are loosely bound qualitative objectives, the assessment is in some ways subjective. However, by constructing a framework for assessing the nature of the relationships in a systematic manner, and then consistently applying that framework for each consideration, the matrix can be constructed. In **Figure 2-B**, key engineering/technical considerations derived from the requirements specifications relate in mixed ways against the summary and key requirements from the hypothetical customer.

Naturally any analysis framework must suit the subject being analyzed. Hence, a discussion of the framework and the process by which it was constructed is served by first enumerating and defining the key (mostly qualitative) objectives – from an engineering customer's perspective, in terms of utility. In this context:

(Reasonable) Total Maximum Cost

Whereas, the entire appliance ought to be under some reasonable total price. Be that the utility of the device is in its appeal at social gatherings and its ability to serve drinks successfully, the intuition is that its total cost should not surpass this utility. In this case, as the project is self-motivated and self-funded, an asserted value can be taken directly from the engineering design specifications: \$1500

(Immediate) Availability of Beverage Recipes

Whereas, the appliance should ultimately provide the maximum range of beverage options at any given time as practically possible. The intuition being that a potential private owner of the product, for example, would maximize their utility by reducing the need to reconfigure the apparatus to choose a new set of beverage flavors or styles, but that it might be readily capable of immediately serving from a large range of pre-configured and accessible options. Or, similarly, one may wish to scale the appliance to serve from the ingredients available in a full bar.

Autonomous Drink Delivery

Whereas, the appliance is to maintain some capacity to deliver a beverage from the bartending station to the user at a predefined location autonomously; it is implicit that a private user of this appliance would yield the maximum utility from its ownership from the maximally sophisticated level of autonomous navigation possible, considering this feature is inherent to the overall application of the device.

Ease of Implementation

Whereas, the appliance is a self-funded and self-motivated engineering project under a formal and strict development timeline, it is prudent to include the objective to facilitate the implementation of this appliance and its features wherever and however possible.

Single-Charge Life Span

Whereas, the appliance yields its utility from serving beverages at private social gatherings, it is intuitive that a private owner of this appliance would yield the maximum utility from its ownership where the appliance is capable of operating for as long as possible during any given event.

Ease of Use

Whereas, the appliance is expected to often interface with lively and inebriated partygoers at social gatherings, potentially in the absence of the owner and configurator, it is intuitively imperative that the appliance also be designed such that is as simple to use as possible.

2.4.1. Analysis

To be clear, the nature of the relationships between these mostly qualitative objectives is driven from the practical nature of the technical considerations. The technical engineering considerations are bound by the engineering design specifications, the conceptual outline of the B3 appliance, and the engineering design process. Further, each technical consideration defined maintains a specific quantitative objective or, otherwise, a direction of objective optimality - an optimal direction to push the consideration, so to speak.

From an engineering perspective, in alignment with the incentives and constraints mentioned before, it is desirable to observe the following key engineering design considerations which have been compiled from the overarching design objectives and requirements specifications:

1. Maximize the number of pumps in the system – so as to enable the maximum availability of beverage ingredient options.
2. Minimize the number of sensors in the system – so as to facilitate implementation, such as by simplifying the sophistication of the firmware required to meaningfully and efficiently interpreting sensor information for operational purposes.
3. To achieve a target size for the Bartender – so as to ensure the appliance has both an aesthetic packaging and a minimally invasive form factor.
4. Minimize the sophistication, or design intensity, of the Bartender’s integrated PCB – so as to facilitate implementation by reducing the amount of time that must be invested in development.
5. Minimize the overall cost of a manufactured Bartender’s integrated PCB – so as to support remaining within the prototype’s budget constraint and otherwise constrain the sophistication of the PCB.
6. Minimize the amount of time required to render a functional firmware for the Bartender – so as to enable direct focus on the immediate scope of implementation and other design objectives.
7. To achieve a target size for the Butler – so as to render a stable and minimally obtrusive platform for transporting a full beverage.
8. Minimize the sophistication, or design intensity, of the Butler integrated PCB – such as to facilitate implementation by reducing the amount of time that must be invested in development.
9. Minimize the overall cost of a manufactured Butler’s integrated PCB – so as to support remaining within the prototype’s budget constraint and otherwise constrain the sophistication of the PCB.
10. Minimize the amount of time required to render a functional firmware for the Butler – so as to enable direct focus on the immediate scope of implementation and other design objectives.
11. Maximize the number of off-the-shelf components – so as to directly facilitate implementation by limiting the scope of open-ended design elements outside the scope of the team’s field of expertise.

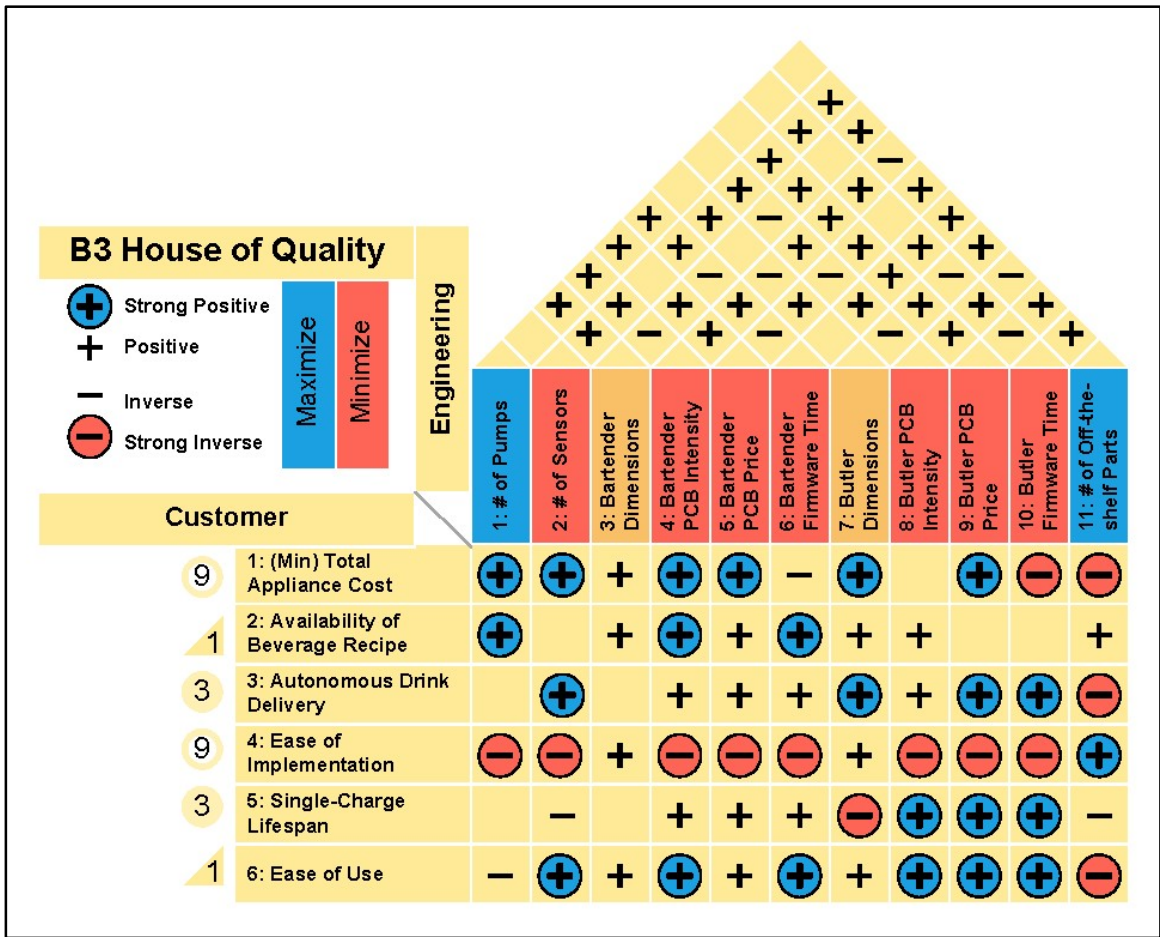


Figure 2-A: House of Quality (QFD)

In **Figure 2-A**, the relationships are approached as either positively or inversely related. Values are assigned subjectively, but along the aforementioned conceptual framework. Crucially, a weight is assigned to each customer objective according to the number of supporting engineering considerations – wherein a subsequently calculated “focus” score then represents the degree to which the customer objective is not positively supported (or enabled) by the competing engineering design considerations.

For example, the customer’s required capacity for the device to navigate autonomously from the bartending station to the user is positively increased or technically enabled by also increasing the number of sensors, and the amount of time spent on the Butler’s autonomous navigation firmware. As another example, the total cost of the appliance is strongly positively correlated with the number of pumps, but only inversely correlated with the number of off-the-shelf parts; total cost increases as the number of pumps increased, but the total cost of a fully constructed system may only slightly decrease if more of its components are off-the-shelf.

2.4.2. Notes on Exemplary Trade-offs

This section lightly explores noteworthy observations made from the House of Quality given in **Figure 2-A**.

Note 1: It is evident that the customer perspective requirement for a capacity to achieve autonomous delivery is in direct contention with the parallel requirement to simplify the implementation. This may yet be feature of how the requirements were specified.

Note 2: The customer's objective for ease of implementation scores a high 9 on the "focus" score. It can be deduced: To achieve the customer's objective, hard and overwhelming focus must be placed on the technical design considerations which support its achievement – namely on maximizing the number of off-the-shelf parts or open-source software.

3. Background & Technical Research

This section explores the contemporary environment to discover the most practical approach to the various mechanical, electrical, and system engineering problems the B3 must be designed to overcome – as if on the path to commercialization, but with explicit focus on the achievement of core functionality in the prioritized manner deduced during the analysis of design tradeoffs in the previous section.

This begins with an analysis of projects and products that fulfill at least portions of the design objectives, followed by a detailed review of the most critical components in those designs; these serve as a contextual background for an initial design architecture. To further frame and direct the initial design architecture and set an initial prototype on the path to more advanced development, a brief but dedicated review of the potential commercial space is made – including a roadmap for optimizing the end-design to find and capture a market space.

3.1. Contemporary Projects & Products

The following section is based on research relative to those contemporary engineered systems which are similar in major key aspects of function and offer insightful clues as to where to initiate a foundation for the B3, motivated and introduced in the previous section; that research plays a key role in building upon existing solutions to engineering problems, so as to further develop technology within the intended area is notwithstanding.

3.1.1. SirMixABot

The SirMixABot began as a personal project with a goal of automating the bartending process. The developers aimed to facilitate the beverage selection and making process in a private space and setting. Once the original project was completed and in use, the positive feedback from those who tested it and the interest it garnered led to the project undergoing a noticeable development and expansion. The enthusiasm for the device noticed by the developers led them to expand what was originally a personal passion project into a fully commercialized product, sold online for shipment to users' homes.

The overall design objective for this project is similar to that of the SirMixABot in that both aim to construct an electronic device that automates the process of selecting and mixing a beverage. However, the B3's intended design deviates from SirMixABot's in the inclusion of a drink delivery feature provided by the Butler. Nevertheless, the drink dispensing features of the SirMixABot are a very useful exemplar in designing the Bartender and uses many components that can be similarly purposed for this project.

Key Features: Wi-Fi Enabled

Key Components:

- Atmel ATmega2560
- Infrared Obstacle Avoidance Sensor Module
- A3144 Hall Effect Sensor
- 1602 LCD display and Keypad
- SparkFun Wi-Fi Shield
- AdaFruit v2 Motor Shield

Noteworthy: Open-Source and thorough documentation

The chosen microcontroller in this product was the Arduino Mega 2560, built on Atmel's ATmega2560 chip, which is a common choice due to its versatility. This Arduino possesses the capacity to handle a broad scope of tasks. For this product, these tasks include the following: receiving signals from both Hall Effect sensors and IR sensors, controlling motor motion, updating LCD screen, and initiating LED lights throughout the mixing process. Additionally, it can be used to administer a Wi-Fi microchip, expanding the product to have Wi-Fi capabilities. The creators have used the AdaFruit v2 Motor Shield in combination with a SparkFun Wi-Fi Shield to achieve this feature. A SparkFun Wi-Fi Shield is a development board integrated with an AT-command, and it breaks out and provides command access to all of the ESP8266's I/O. This was said to be done to decrease the development time, regardless of the cost increase.

A common limitation that must be observed for a contemporary microcontroller is its memory capacity. A microcontroller such as this has 256k of total memory. In light of such a limitation, other methods of memory can be implemented. In the case of the "SirMixABot", a drink server was created to program and store various drink combinations and generate a menu to be ordered from. A notable aspect found in this research is that the company made their code open source for others to use as building blocks and to encourage improvement and creativity.

With respect to marketing, SirMixABot produced two models of the "SirMixABot". The first model is constructed to integrate six bottles and is named the "Sixer". Their second model is named the "Mega", which integrates ten bottles into its configuration. Both models are available to buy, either as a kit or already assembled. This additionally provides context to the expected number of bottles available in a marketable form of this project, which can be explored further in the marketing of a post-prototyping B3 unit.

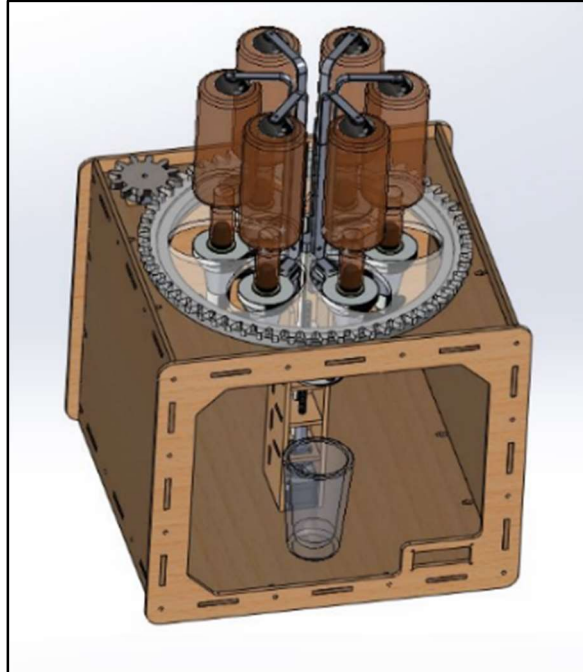


Figure 3-A: SirMixABot
Image courtesy of: [SirMixABot](#)

3.1.2. Automated Bartender

This incomplete project was found on a platform called Hackaday where people can collaborate, create, and receive feedback from within the community as they progress through their publicly documented projects. The primary motivation for this project was intrinsic technical aspirations; the creators were entering the *Hackaday Automation Challenge*. The main idea for the project was to automate domestic bartending while increasing mixing speed, decreasing messes and spills, and overall enriching the mixology experience.

This project idea shares similar objectives with our intended project, such that the goal is to automate the beverage mixing process with a bartender bot and in turn limit or even eliminate factors such as unwanted spills, slow mixing speed, and inaccurate pours. Again, the groundwork for this project is insightful and provides a vision for an ambitious implementation of a sophisticated bartending system.

Key Features: Mixing multiple beverages at a time, rotating platform

Key Components:

- Atmel ATmega328P
- Raspberry Pi
- 8 Channel Relay Module - Frentaly® 8 Channel DC 5V Relay Module
- Self-Priming Pump

Noteworthy: Incomplete project, too sophisticated for development timeline

The Arduino Uno, built on Atmel's ATmega328P, was the chosen microcontroller in this project. It was designed to manage the actuators, pumps, position switches, and the drive motor. The amount of outputs that needed to be controlled exceeded that available on the Arduino Uno. Therefore, a shift register was added to accommodate the additional outputs and bypass this limitation.

Another key feature is the creator's choice of dispensing system. This project utilized a self-priming pump with vinyl tubing. This seemed to be a noteworthy aspect due to the fact that it was not the common choice of pump throughout the research of existing products. Pros for the self-priming pump would be that this pump only needs to be initially primed before the first use and can handle a very wide variety of fluids, and solids if necessary. Taking these pros into account potentially led these creators to choose the self-priming pump for this project. A flow meter was also added to achieve the desired accuracy of dispensed liquid.

Another notable difference is the physical framework of the system itself. The platform housing the beverage glasses was designed to rotate throughout the mixing process. This feature added pleasing aesthetics to the design as well as another layer to its functionality. This added functionality resulted in a unique feature not commonly found in other researched projects: the added rotation allowed the bot to execute more than a single beverage at a time. Thus, the bot now not only would automate the mixing of beverages but could do so for multiple beverages simultaneously. This platform was constructed with the use of a drive motor in combination with a pulse-width-modulated motor controller which is used to control the speed of the platform's rotation. Another aspect to note is the use of position switches to ensure the correct location of the platform holding the beverage glasses relative to the base before the beverage mixers are dispensed.

This project is also open source to allow others to follow along their journey of the automated bartender bot's creation. This was done by the creators to give others the steps to create one themselves, use it as a steppingstone to further a creation of their own, or even for others to simply satisfy their curiosity of the project and possibly learn new things along the way. The creators not only documented the information for the bot with files on their project site, but they also video documented their steps throughout the design and building process and showed the success at each step before moving on. Unfortunately, however, the only missing video is the final video of the finished product.

This team has an interesting take on the automated bartender bot. Although, our project relates in some respects, this project possesses fairly defined differences. The impressive feature of mixing multiple beverages at once is definitely one to take notice of and a positive motivation for our intended project and possible future expansions. [2]

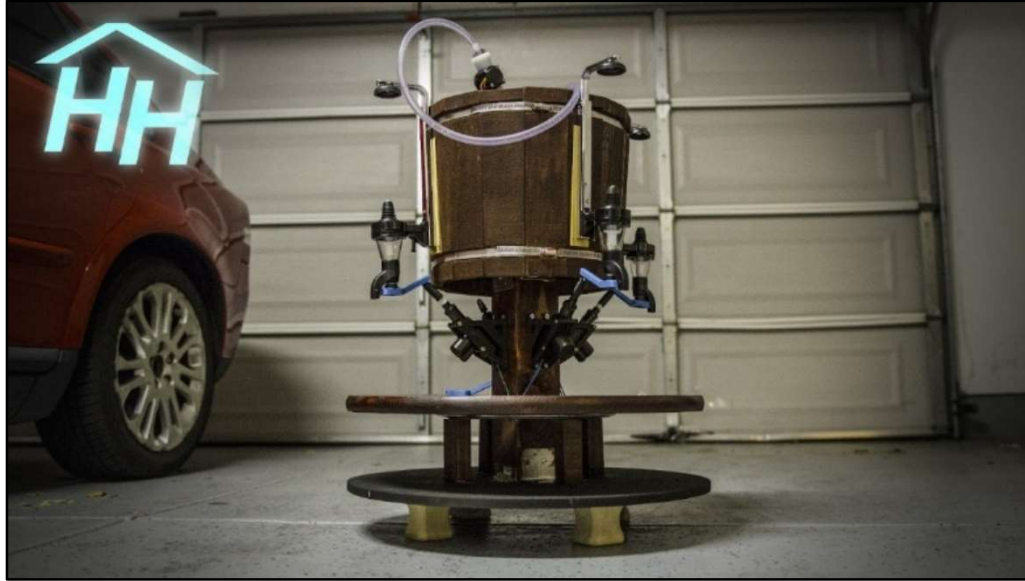


Figure 3-B: Automated Bartender

Image courtesy of: [Hacker House](#)

3.1.3. Bartendro

The “Bartendro” was created by a company called *Party Robotics*. Back in 2008, they started with the idea of a personal robot that automated the cocktail making process. The idea took off. They began with only a few of the pump systems which they called the “Bartendro Dispensers”, and they kept expanding with this design until ultimately reaching a product capable of accommodating as many as fifteen bottles. They now sell the systems of various bottle capacities with the pitch of using the systems not only in a personal space such as your home, but also using it for small or even large parties or events.

In reference to design objectives, the Bartendro possesses a similar idea to our intended project. As a general objective, both are intended to facilitate the process of mixing cocktails. The scale of its output is a crucial and outstanding feature that differentiates it from its peers – at up to 15 bottles in one appliance, it has the broadest range of possible recipes. However, at nearly 3500 USD’19, it is a hefty investment in entertainment.

Key Feature: Wi-Fi Enabled

Key Components:

- Arduino ATmega168
- Raspberry Pi 1 B+
- Hall Effect sensors

Noteworthy: Open-source, Peristaltic pump system

The team from Party Robotics began designing a modular, open source bartending bot that they called the “Bartendro” back in 2008 where they, as any good engineer would, built upon what was previously available to them. Throughout the development, they posted their findings as they debugged, redesigned, and improved upon them. They did so in an open-source platform so that posterity could advance upon their results. Utilizing a microcontroller board with integrated sensors, an ATmega186 microchip, and peristaltic pumps, they aimed to design an easily assembled pump system. After much trial and error in this process, the “dispenser” system was successfully created. This led the team to achieve the single bottle dispensing system, which they deemed the “ShotBot”. They expanded to multiple versions of the product by integrating additional dispensers, in turn broadening the selection of possible beverage combinations. The resulting products are Bartendro 3, Bartendro 7, and Bartendro 15, which have a 3-bottle dispensing system, a 7-bottle dispensing system, and an impressive 15-bottle dispensing system, respectively. While they do sell these versions of the “Bartendro”, their main product is their dispensing system as a kit. As mentioned before, being open source, they encourage the use of their designs to inspire and push the ingenuity of others, which is a noteworthy aspect of any project in the engineering world.

The Bartender Butler Bot intrinsically shares some similarities with the “Bartendro”. It forms the core conceptual approach of a successful and scalable beverage dispensing system. In the sense of achieving the B3’s minimum functionality, this project yields a wealth of insight and direction. [3]



Figure 3-C: Bartendro 15
Image courtesy of: [Party Robotics](#)

3.1.4. Elliot the Line Follower Robot

Autonomous navigation is a highly sought-after technical achievement. In recent years the plethora of cheap sensors and powerful microcontrollers in the robotics market has made even rudimentary versions of automated navigation accessible to entrepreneurs, hobbyists, and scope-limited engineers. The Elliot, a fully documented Arduino project hub contribution, is a guided approach to autonomous navigation: a path following robot, supported by an array of path-monitoring sensors and a PID controller. The project is explicitly motivated by the desire to minimize the cost and technical difficulty of implementation – i.e. by proving it is accessible to hobbyists, tinkerers, and DIY enthusiasts. Though the project serves no functional purpose, it lays the groundwork for an approach to a qualified, if strictly limited, autonomous navigation system that would be purposefully suited for the B3's design objectives.

Key Features: PID controller, off-the-shelf robotic chassis

Key Components:

- Arduino Uno – Atmel ATmega328p
- IR Sensors
- L298N motor controller module

Noteworthy: Simple electrical tape is sufficient for physically defining a path

The Elliot is a crude combination of off-the-shelf components. A digital system composed of an Arduino Uno, some IR sensors, and a motor controller is laid out on a prototyping breadboard. The Uno receives digital inputs from the IR sensors to ascertain its location relative to the center of the physically demarcated path. The deviation from the path is estimated and pushed through an algorithmic proportional-integral-derivative open-loop controller that yields a calculated and precisely tuned correction. The correction actuates the DC motors through the motor controller, generating a stochastic forward motion that intends to minimize the robot's deviation from the center of the path along its route. This entire electronic system is mounted upon a lightweight prefabricated plastic robotic chassis, is powered by a simple nickel metal hydride battery and is ultimately functional for less than 100 USD'19.

The PID controller's parameters can be optimized in the algorithm to smooth the stochastic output to render the finest motion across the path possible. A trove of documentation is available on the project page, sponsored directly by Arduino. Open-source sample code, descriptive imagery, and a simple BoM is provided. A more sophisticated implementation of this approach, such as one involving obstacle detection or avoidance, could be constructed from the groundwork available and more advanced functions from other projects could be systematically integrated.

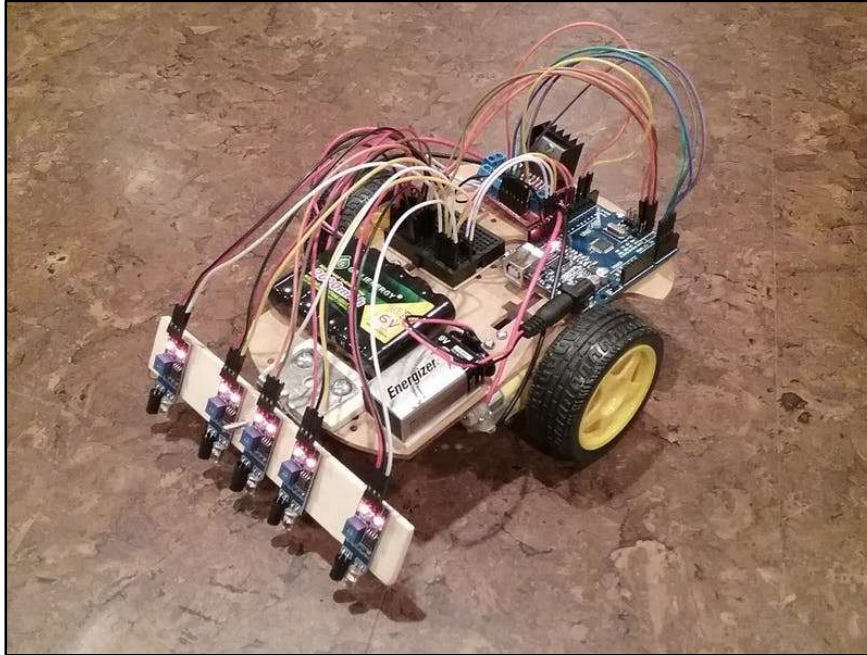


Figure 3-D: Elliot the Line Follower Robot
Image courtesy of: [Arduino](#) & SMM2

3.1.5. RaspRobot OpenCV Project

There are various technical obstacles to the implementation of autonomous navigation. In particular, it is challenging to use digital equipment and sensors to generate a sufficiently crisp, detailed, and intelligible portrait of the analog world for computer algorithms to efficiently and accurately interpret. As a result, efforts turn to imitating biological methods – such as with visual processing in terms of shapes, colors, edges, and brightness. The “RR.O.P.” is a Brazilian engineering student’s successful attempt to demonstrate the implementation of OpenCV for computerized motion on the most inexpensive platform possible. The project began as university assignment but completed as a machine vision public sensation. The RR.O.P. is only meant to find and track a bright green object, but it does so efficiently, at low cost, and with straight-forward well-documented manner. It lays the groundwork for object identification and tracking, key challenges to overcome in designing and programming robotic systems to dynamically react to the spatial environment.

Key Features: Computer-vision, object-tracking

Key Components:

- Raspberry Pi B+
- Logitech C270 HD Camera
- USB 802.11 Wi-Fi EDUP Module EP-MS1537
- Python OpenCV

Noteworthy: Translated documentation is unwieldy but available

Mounted on a spare off-the-shelf robotic chassis driven by H-bridge motor drivers, a front-facing camera, Raspberry Pi, and Wi-Fi module are integrated to form the hardware backbone of a sophisticated algorithmic approach to mechanical motion. Images from the camera are fed into a well-developed community-driven computer vision API and transformed into vector data on hue, saturation, and brightness. The processed image information is then qualified by built-in OpenCV functions to identify a clearly and strictly specified object, a bright neon-green ball, in the field of view. The position of the object relative to the defined “center” of the field of view is then analyzed to calculate a corrective motion for the robotic platform so as to bring the object into a prespecified location within the field of view. The design iterates: motion is first linear – assessing proximity (or depth), and then planar – assessing the location of the object in relation to itself in 3D space.

This project yields insights on the effort required to achieve the B3’s most advanced conceivable design. The algorithm employed in this project nears the self-assessed boundaries of what might be possible to achieve in the given development timeline but offers a tantalizing and motivating vision of successively more sophisticated versions of the B3. For example, this algorithm could be used to find and locate the specific user in the room that made the order, or it could be used to facilitate obstacle detection and avoidance. A take on this approach, in fact, is being used by Tesla, an advanced contemporary automaker, to develop a level 5 autonomous automobile and clearly renders an unknown wealth of potential.

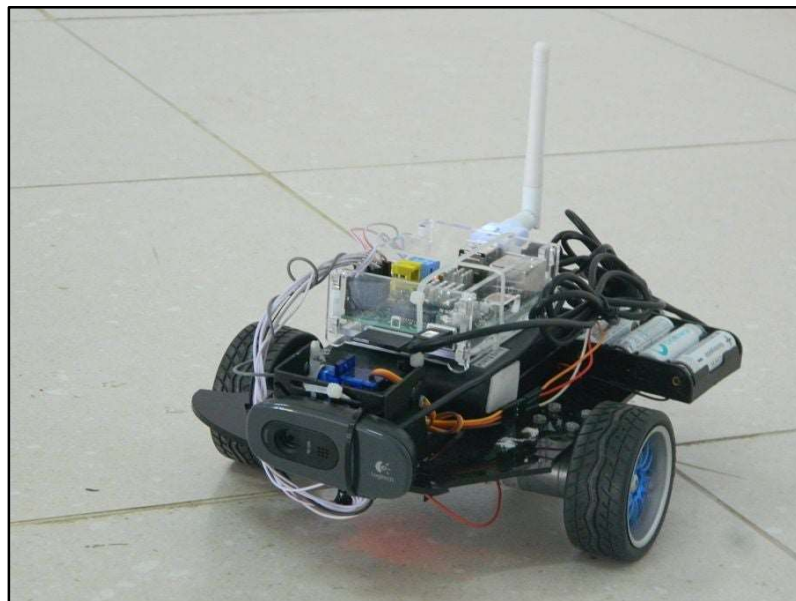


Figure 3-E: RR.O.P

Image courtesy of: [Oliveira Saymon](#) & Instructables

3.2. Relevant Components & Key Technologies

This section selects from the list of specific key, useful, or noteworthy components identified from the similarly scoped projects explored in the previous section and

offers further details on their functionality, underlying technologies, and their relevance to the development efforts documented in this report.

3.2.1. Arduino Uno/Mega

A microcontroller such as an Arduino Uno is a crucial component in many projects. An easily flashable development board with a multitude of applications greatly simplifies the implementation of embedded systems. Firmware can be written more dynamically than an analog system can be designed; the integrated circuit and flash memory technology proving once-again its flexibility and practicality. Due to its ease of implementation and wide compatibility, the Arduino platform itself is commonplace. Due to the fact that Arduino is an open-source development platform, the community surrounding it is vast. All items about the Arduino development approach further enabling the range of creative development with a focus on practicality, ease of implementation, and deep support foundations.

Key Technologies: NMOS integrated circuits, open-source development boards, integrated development environments, flash memory, SRAM, EEPROM

3.2.2. ESP8266

This tiny Wi-Fi-enabled SoC is versatile and well-developed. It has a high number of GPIO pins with a multitude of functionalities and is often embedded on a large variety of integrated boards. Crucially, implementing this chipset in a design allows for the establishment of Wi-Fi communications. It was used in “Bartendro” and the “SirMixABot”. For the B3, a conceivable application of this capability could, for example, allow a drink to be ordered by an app on a user’s phone; or provide the communications framework for IoT-based data exchange.

Key Technologies: 802.11b/g/n wireless RF communications, CMOS integrated circuits, QFP packaging, 2-layer PCB

3.2.3. Raspberry Pi

The Raspberry Pi is a full-featured complete SoC that offers a platform to run external components such as an LCD screen or display. It is powerful enough to sustain a complete operating system with high-level languages and environments like MATLAB, R, and Python, whilst making a native suite of GPIO pins available for embedded digital system applications. This component is used where sophisticated algorithms are required, such as with OpenCV or a dynamic user interface. This component integrates a multitude of technologies and yields a powerful package for a broad range of applications.

Key Technologies: System-on-a-Chip, combined-MOS integrated circuits, DDPACK packaging, multi-layer printed circuit boards, operating systems, audio codecs, video codecs, DDR volatile random-access-memory, microSD, Bluetooth

3.2.4. Peristaltic Pumps

Driving and dispensing fluids naturally requires the use of pumps. Peristaltic pumps are DC driven pumps that drive fluids without ever coming into direct contact with it. From **Figure 3-F**, a central rotor (1) is spun by the DC current, driving “rollers” or “lobes” on spokes (2). The “rollers” compress the flexible tubing (3) inside the pump against the circular enclosure (4) as they rotate about the rotor – generating a positive fluid flow. As a result, they were employed in both the SirMixABot and Bartendro.

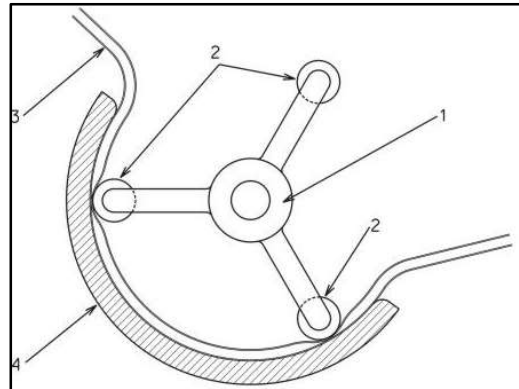


Figure 3-F: Peristaltic pump diagram
Image courtesy of: [Wikimedia Commons](#)

Key Technologies: anterograde peristalsis pump action, direct current induction

3.2.5. H-Bridge DC Motor Drivers

It is practical, in many applications beyond the scope of this project and nearest peers, to control the direction and speed of DC motors. An H-Bridge circuit is a four terminal switching circuit that allows for control over the polarity of the voltage drop across the load. Control over the polarity of the voltage across a DC motor also controls the direction the motor rotates, thus its direction. PWM control over the DC input to the motor enables the control of its speed.

From **Figure 3-G(A)**: when S1 and S4 are OFF, and S2 and S3 are ON, the voltage across M is equivalent to the input voltage. As in **Figure 3-G(B)**: when S1 and S4 are ON, and S2 and S3 are OFF, the voltage across M is equivalent to the opposite of the input voltage. Practical H-Bridge motor drivers are generally MOSFET-based implementations of H-Bridge circuits in an IC package. These packaged IC devices enable both direct polarity-switching and PWM control of the DC motors on the load, and many also support multiple output channels. These were used, at least implicitly, in *all* of the products listed above.

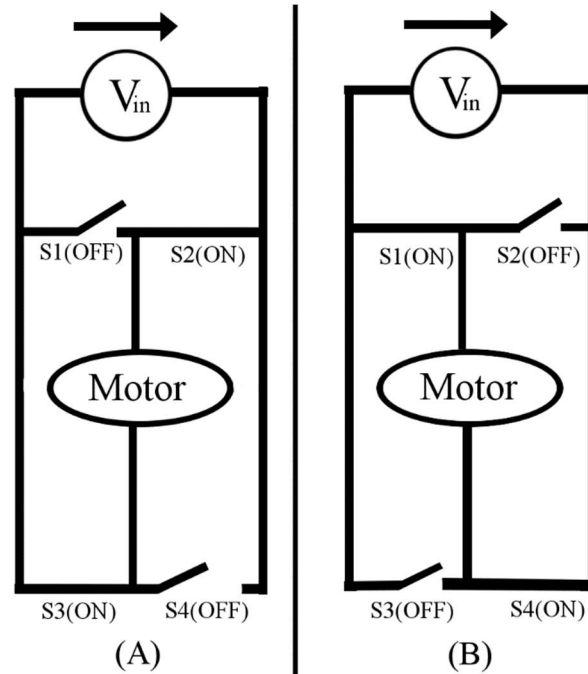


Figure 3-G: H-Bridge circuit in the case of A) Positive V_{in} over the motor and B) negative V_{in} over the motor

Key Technologies: metal oxide semiconductor field effect transistors, pulse-width-modulation, combined-MOS integrated circuits

3.2.6. Relay Modules

The standard state of operation for various systems will require the use of relay module. A basic relay module is used to determine whether connected circuit should be powered or unpowered by default and is used to control various appliances and equipment with high current. Typically, there is an interface that can be controlled directly by a microcontroller, such as an Arduino. Due to their small size and the fact that a microcontroller is often used to control the logic input, it is common to see interface boards that consist of multiple relay modules so an entire system of equipment can be controlled through a single microcontroller.

Figure 3-H shows a simple diagram of a relay modules two basic states, which are normally closed operation (NC) and normally open operation (NO). A relay module consists of 3 inputs, a logic input, and a switch. One of the three inputs is simply the common input, while the other two determine whether the connected system is normally powered terminal, NC, or unpowered terminal, NO. The internal switch is connected to the NC terminal by default, hence the name. When the logic terminal receives an input, the internal switch changes from the NC input to the NO input via a magnetic attraction of the metallic switch. So long as the logic terminal receives a positive voltage, the switch will remain connected to the NO terminal. When the logic voltage becomes zero, an internal spring will return it to

default state. Regardless of whether a system is connected to NC or NO, triggering the switch will change its on/off state.

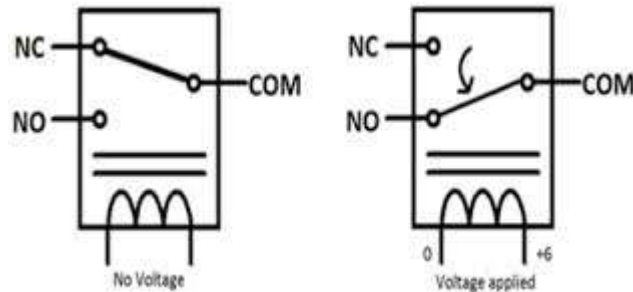


Figure 3-H: Relay module schematic
Image courtesy of: [Circuit Basics](#)

Key Technologies: Reed switch, electromagnet, MOSFET logic

3.2.7. Infrared Photodiode Sensors

Infrared obstacle detection sensors are simple photodiode circuits with an output dependent upon the magnitude of infrared light detected. Common implementations of this sensing unit have both a digital and an analog range of output with a built-in IR light-emitting diode. These were invariably employed explicitly on the Elliot and other similar projects.

From **Figure 3-I**, the IR LED emits infrared light, and if a reflective surface is before it the photodiode will yield a significant output voltage. As seen in Part (A), if a reflective surface is present, the IR light will reflect and be received by the Photo Diode, whereas (as in Part (B)) if a non-reflective surface is present, the IR light will be absorbed, and little to no IR light will be received by the Photo Diode. Therefore, by placing a particularly reflective or particularly non-reflective object as a guide, a path for the line follower can be produced. In many instances, simple black electric tape is used for this purpose. The application decides whether the output received from this Photo Diode needs to be a digital value or analog value. Although a digital implementation may be simpler, an analog implementation yields more detailed set of data on the environment.

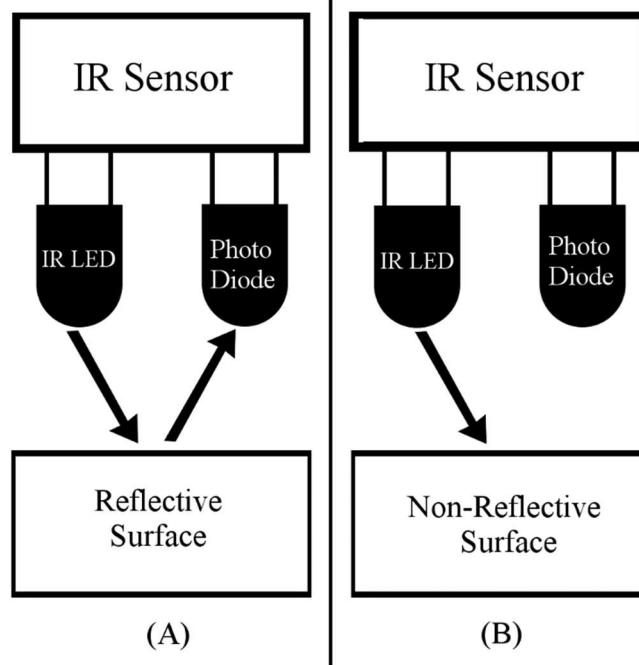


Figure 3-I: Infrared photodiode sensor for: A) a reflective surface, and B) a non-reflective surface

3.2.8. Hall Effect Sensors

Hall effect sensors can be implemented to achieve countless goals within a design. Some of the common implementations are for motion sensing, sensing availability of power supply, sensing rate of flow, voltage regulation, and sensing proximity or pressure. Of the projects researched and introduced, the SirMixABot and the Bartendro, both used this technology at some capacity within their designs.

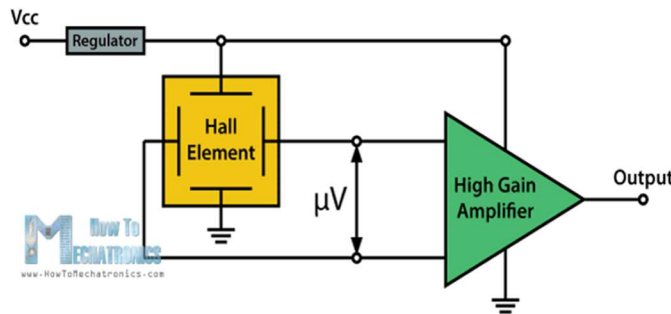
The Hall Effect sensor is essentially a transducer that yields an output as a response to a magnetic field. Hall Effect sensors can be broken down into categories based on the operation and on the output. From there the category of operation, can be further divided into either a Bipolar Hall Effect sensor or a Unipolar Hall Effect sensor. To state simply, the bipolar sensor operates by utilizing both the positive and the negative magnetic field of a magnet to activate and release the sensor whereas, the unipolar sensors operate by only the use of the positive magnetic field to activate and release the sensor.

For the focus of output, one could have either an analog output or a digital output, as seen in **Figure 3-J**. For the case of analog, as seen in **Part (A)**, the circuit is comprised of a voltage regulator, a hall element, and an amplifier. This yields an output directly proportional to the magnetic field. Contrastively, seen in **Part (B)**, the Hall Effect sensors with digital output differs by the addition of a Schmitt Trigger. This addition allows the Hysteresis effect to take place. Thus, the resulting circuit

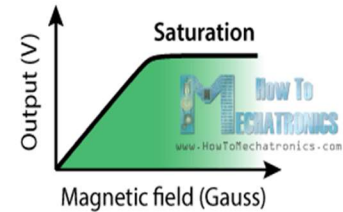
now only yields one of two outputs, either High or Low – similar to what is commonly known as a switch.

As previously mentioned, various projects implement sensors such as these to provide a wide range of information within designs. This allows for a multitude of possibilities to further advance our possible design features.

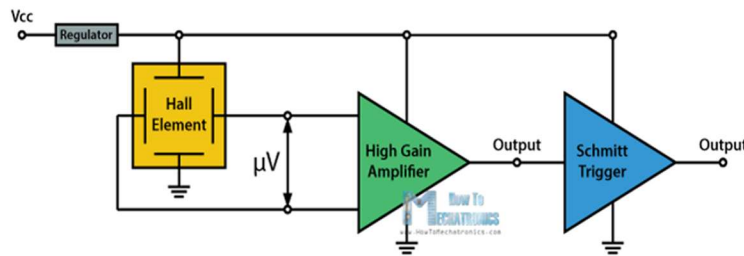
Part A) Analog Sensor Circuit



Analog Output



Part B) Digital Sensor Circuit



Digital Output

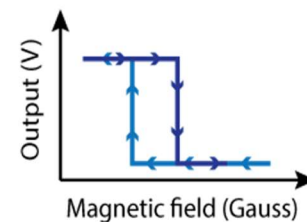


Figure 3-J: Hall effect sensor for: Part A) Analog Sensor Circuit, and B) Digital Sensor Circuit

Source: HowToMetatronics.com:

Key Technologies: hall elements, standard bipolar transistors, voltage regulation, combined-MOS integrated circuits

3.2.9. Straight-Bar Strain-Gauge Load Cell

A straight-bar, strain-gauge load cell can be used to cheaply and accurately measure the weight of a force (usually a weight) applied to an area. This process works using a metal bar, the strain gauge, which when presented with force below a certain threshold deforms in a reliable manner. This deformity causes a consistent change in the overall resistance of the bar, usually made of an aluminum alloy. This change in resistance can be used to reverse-engineer the value of the weight being applied to the bar, functionally creating a type of digital load cell.

In order to properly implement a straight-bar strain gauge load cell, the component is usually mounted on only one side of either end, as seen in **Figure 3-H** below. Generally, one of these mounting points is a secure surface, while the other is a free load platform upon which the weight in question can be applied. This setup helps to isolate the effect the weight has while minimizing any error present from other minimal effects on the strain the component experiences, allowing the metric that is developed for measuring a specific weight from the resistance to act as a much more accurate load cell.

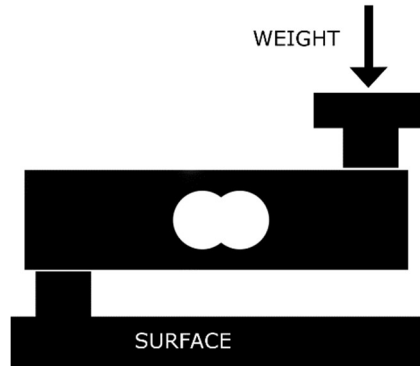


Figure 3-H: Standard Straight-Bar Strain-Gauge Mounting Setup

4. Standards & Regulations

The following section details the various externally imposed standards and regulations which must be met as part of the design and implementation of the B3. The section is split into broad categorical sub-sections that contain a rigorous assessment of specific standards and regulations that are relevant to a range of objectives, the overall design, or a particular functionality of the B3. This list is as complete as the authors of this report believe it can be, and new items are added as they are discovered. For each standard or regulation clearly identified, a summary description of it is made, its motivations are explored and its implications on the design are analyzed.

4.1. Health & Safety Standards

Due to the goal and nature of the B3, there are various health and safety standards that must be reviewed and incorporated into its design. These are crucial to the design and implementation of the B3 because these standards limit the types of components that can be used. The relevant health and safety standards related to this project are listed below.

4.1.1. Vending Machine for Food and Beverages

The standard that covers the dispensation of food or drink via vending machines is covered in NSF/ANSI 25. The purpose of this standard is to establish the minimum food protection and sanitation guidelines for materials, design, construction, and performance testing criteria. The required specifications are designed to prevent beverages from contamination to ensure that the materials used to construct vending machines are resistant to wear, vermin, and the effects of heat, sanitizers, and other various substances that can come into contact with it. Given the nature of the prototype, to reduce costs, the outer paneling will be constructed out of wood. If the B3 were to be commercialized in the future the outer paneling will likely need to be substituted with something more suitable.

4.1.2. Drinking Water System Components

The NSF/ANSI 61 is the standard that covers drinking water system components. This standard certifies what materials and chemicals that are used in drinking water systems are safe and will not produce negative health effects in its users. This is relevant due to the usage of fluid storage and fluid tubing in our design. This standard mandates that the containers and tubing do not contain chemicals that could contaminate fluids that come into contact with them that could result in negative health effects when ingested. Due to this, they will need to be of food grade; improper selection of components could be costly and create issues, so this should be avoided at all costs.

4.1.3. Fire Safety and Emergency Symbols

The NFPA is a fire safety standard that provides symbols used to effectively communicate fire safety, emergency, and associated hazards information. The purpose of this standard is to provide a series of easily comprehensible symbols and signs such that emergency response personnel, employees, and any average person can understand the meaning of the symbol regardless of reading comprehension or language spoken. They also greatly aid in comprehension to reduce reaction time of emergency personnel during a crisis. For the B3, the Butler contains batteries, as well as a charging station for the Butler. Electrical hazard symbols will be needed to warn of electric shock if the system is not handled with proper care.

4.2. Digital Standards

Given that the various subsystems of the B3 will be communicating wirelessly, there are various digital standards that must be taken into consideration when selecting components for the design of the B3. Microcontrollers with Wi-Fi capabilities have various Wi-Fi specific standards that must be followed, and as such only microcontrollers that follow these standards will be used. Furthermore, the messaging data that is being transmitted over this Wi-Fi must also follow the messaging standards outlined by one of various messaging standards. For the B3, the IEEE 802.11 series of standards will be used for the Wi-Fi and the MQTT v5.0 standards will be taken into consideration for the messaging standards.

4.2.1. Wireless Communication Standards

The Institute of Electrical and Electronics Engineers (IEEE) developed a set of standards known as IEEE 802.11 for wireless communication. In order to minimize costs, and due to the fact that the B3 will be communicating relatively small amounts of information, older options of Wi-Fi will be used for this project. Many cheaper Wi-Fi devices communicate on the 2.4-gigahertz(GHz) frequency band. IEEE 802.11b, IEEE 802.11g, and IEEE 802.11n all set the specifications for Wi-Fi communication on the 2.4GHz band [4]. In July 1999, IEEE 802.11b was created as an unregulated radio signaling frequency. It has a theoretical speed of 11 megabits per second (Mbps), and it uses Complementary Code Keying (CCK) to modulate the signal [4]. However, realistic conditions result in as low a bandwidth as 5Mbps due to interference from other 2.4GHz devices despite this modulation. The reason that the data rate drops is to allow for error correction in order to maintain the integrity of the signal being transmitted. The slower the data rates allow for much more data correction [4]. The most practical data rate of 802.11b is 5.9 Mbps using Transmission Control Protocol (TCP)[4].

Institute of Electrical and Electronics Engineers 802.11g was created to improve upon IEEE 802.11b and came out in June of 2003. The primary attraction was the significantly higher data rate of 54 Mbps. This was the theoretical value based on the new process of Orthogonal Frequency Division Multiplex (OFDM); however,

once again the theoretical value proved to be not practical. To allow for error correction, the more common data rate was 24Mbps[5]. IEEE 802.11g maintains backwards compatibility with 802.11b, but the presence of any 802.11b in a network would greatly reduce the speed of the net [5]. Direct Sequence Spread Spectrum (DSSS) helps to match the modulation used by 802.11b. To maintain backwards compatibility whilst ensuring maximum capability, four layers are used, three of which are defined as Extended Rate Physicals (ERPs), which help to link exchanges [5]. The first layer ERP-DSSS-CCK is a layer used by 802.11b to communicate with 802.11g and is capable of data rates similar to the legacy 802.11b. The second layer, ERP-OFDM, allowed 802.11g to acquire the data rates at 2.4GHz that were created by the 802.11a at 5.8GHz. ERP-DSSS/PBCC is an optional layer similar to the DSSS/CCK layer, but the primary difference is that it can end up with the capability to extend data rates to 22Mbps and even 33Mbps. The fourth and final layer DSSS-OFDM is another optional layer to send the packet head using DSSS while the payload is transmitted using OFDM. This layer can also travel at the fastest possible speed for the 802.11g[5].

The IEEE 802.11n was launched in early 2006 and sought to increase the achievable higher speeds than the 802.11g through tailored use of OFDM. It comes in three modes, Legacy, Mixed, and Greenfield. Legacy mode can occur as a 20 MHz signal (the original legacy packet size), or a 40MHz signal that is composed of two halves of the original 20MHz legacy packet, allowing for High Throughput (HT). Legacy mode only enables backwards compatibility with 802.11a, 802.11b, and 802.11g. Mixed mode is similar to Legacy in that it allows the transmission of the previously stated standards; however, it differs by also allowing the new 802.11n. Rather than the old legacy style, Mixed mode uses a new Multi Input Multi Output(MIMO) training sequence format [6]. The final mode is Greenfield mode, which offers the highest data throughput given that it doesn't need to worry about any legacy elements. Only 802.11n can use this mode, but the maximum data rate is significantly higher.

The primary issue with the 802.11n is that using the new MIMO format, which is utilized in all modes except Legacy mode, greatly increases the power used by the hardware circuitry. This is somewhat counteracted by an improvement to the power efficiency of the 802.11n systems due to the introduction of a pseudo-hibernating state to reduce power consumption. This occurs by actively recognizing moments when there is no data being transmitted, or such little data that MIMO is unnecessary, and allowing parts of the circuit to become inactive. Without this, the power consumption to data transmitted ratio would be high enough that it would see far less use.

Given that most Wi-Fi modules include the 802.11b, 802.11g, and 802.11n, the B3 will have great flexibility in what devices it can communicate with. As such many the mediatory application will be able to run through any device and effectively communicate with the Butler and Bartender, as well as between the Butler and the

Bartender in order to follow out the commands necessary for the user to receive an order they input.

4.2.2. OASIS standards- MQTT v5.0

Message Queuing Telemetry Transport, or MQTT, is one of the most commonly used protocols for Internet of Things (IoT) projects. This lightweight messaging protocol uses publishing and subscribing operations to exchange data between client devices and the server. This IoT platform is often referred to as cloud-based software. The general objective of this is to provide consistent and efficient information transfer from machine to machine. MQTT is a method of how this information transfer takes place.

MQTT's basic function involves the connected devices, known as the "clients", to transfer information to and from the central MQTT server, known as the "broker". The actions of "publishing" and "subscribing" are what allow the system to know what information is relevant to both clients and the broker. Subscribing refers to the client requesting the broker to send it a specific information that the client thinks the broker has. Publishing refers to data being sent from one device to the other; a client can publish information to the broker, or the broker can publish information to a client based on the subscription the broker received.

The number of times this information is communicated is based on the quality of service, or QoS, controls applied to a message. QoS 0 is called a "fire and forget;" the message is sent at most once and no confirmation of message delivery takes place [7]. QoS 1 is the opposite of QoS 0, that is, the message is transmitted at least once, and is repeatedly sent until an acknowledgement signal is received. This is known as "acknowledged delivery". QoS 2 is the medium between the two. The message is sent exactly once in an "assured delivery," meaning the sender and receiver engage in a two-level handshake to ensure that one and exactly one copy of the intended message is received.

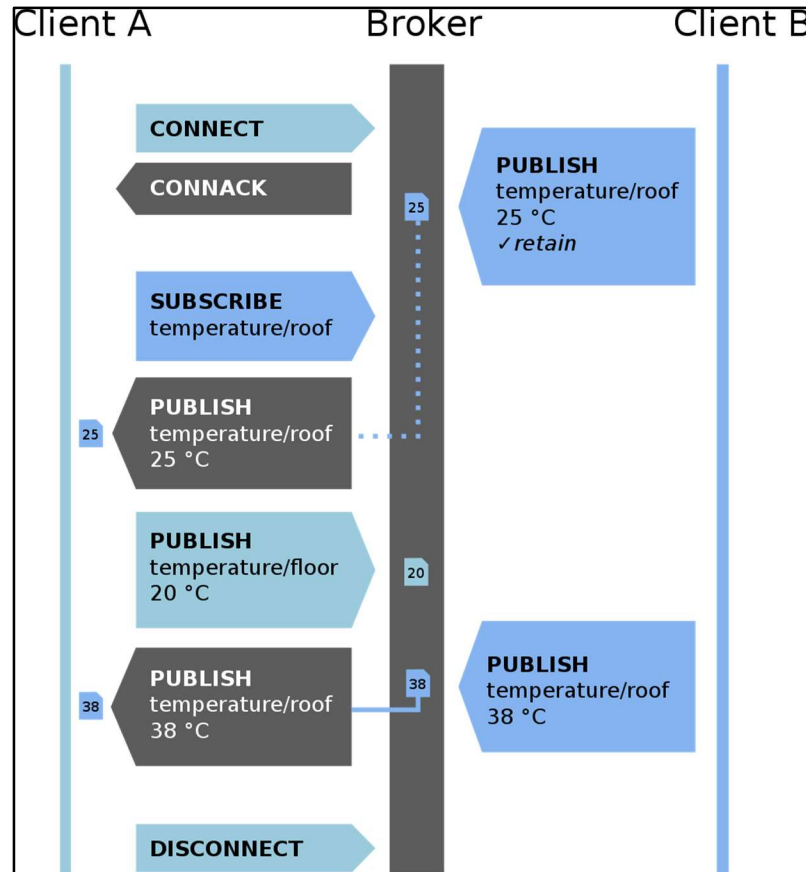


Figure 4-A: The basic series of interactions an MQTT server goes through.
Image courtesy of: [Wikipedia](#)

4.3. Key Design Constraints

Any given system must have realistic design constraints in order to implement it successfully. The sections listed below cover various constraints that will be utilized in the design and construction of the B3. The constraints will first be considered as an individual restriction; each individual constraint must be realistic within the scope of the design. After all constraints have been considered, a complete set of design and production restrictions will be applied and evaluated in order to ensure the constraints remain realistic with respect to each other.

4.3.1. Economic Constraints

The economic constraints will limit the quantity and quality of the parts that will be viable. The abilities and flexibility of components that are possible or desired will be restricted based on the total budget of the project. Furthermore, the total value of the design should not exceed the perceived economic value of service that it provides; in other words, if the monetary cost and opportunity cost of the project does not exceed alternatives, then the project will have no relevance. The ability to deliver drinks automatically is a significant portion of the \$1300 budget allocated for this project, but it also provides users an opportunity cost not seen by similar designs. Given that most of the budget is allocated to the delivering of the drink, it

also restricts other features that would be desirable to add but become unavailable due to limited budget. Particularly, the expansion of the pump system, allowing many more drink variations to be made automatically, will have to be restricted to only a few spirits being available to the Bartender at any given time.

4.3.2. Time Constraints:

Time constraints are relevant in this case since both the design phase and the prototyping phase each last only a single semester. This implies the need for several benchmarks for each phase of the project, creating a list of hard and soft deadlines. Though there is an imposed 60 page (due October 10, 2019), 100 page (due November 1), and final version (due December 2) of the design document, several design decisions, revisions, and other processes need to happen prior to reaching those definite deadlines. These other processes are given deadlines that are self-imposed by the group, in order to ensure that higher quality reports can be created for those hard deadlines. By having a fully fleshed out design prior to the start of the prototyping phase, we aim to have ample time to manufacture and adapt to any mishaps or flaws before April of 2020, the rough due date of a working prototype. This also gives us the flexibility of reworking a segment of the project, should a previously designed portion be determined to take too long to complete before the deadline.

4.3.3. Environment, Social, Political Constraints

Though there are few automated bartending designs, they are completed designs that are proven and are ready to be manufactured. The social and political constraints of this project would be to provide features that are not readily available or inefficiently implemented through other automated bartending designs. The Butler side of B3 will provide a service that has only really been implemented in pet projects and has yet to be created in a system that works in tandem with an automated bartending service. In keeping with the idea of completely automated service of alcohol, an application that can communicate between the Butler, the Bartender, and the user will be designed such that a rapid response from minimal user input will produce the exact desired result. This will give users a different and more convenient option compared to the pre-existing automated bartending designs.

4.3.4. Ethical, Health, Safety Constraints

Due to the nature of alcohol, the ethical, health, and safety constraints are quite significant for this project. Alcohol is a depressant and has been known to foment addictions. Substance abuse can make people less aware of the impact of their actions, which can cause many problems. Furthermore, side effects of heavy drinking are numerous and varied. Some of these effects include anemia, or low red blood cell count, which could induce lightheadedness, fatigue, and shortness of breath. Increased risk of blood clots due to blood platelets clumping together, which can cause a stroke or a heart attack, are also a common side effect of heavy drinking. It is also believed that excess alcohol in the bloodstream is converted into

acetaldehyde, which is a known carcinogen. One of the more commonly known conditions is Cirrhosis. Cirrhosis is caused by scarring of the liver tissue due to filtering the toxicity of alcohol, which permanently inhibits the liver's ability to function properly

Given that B3 will be delivering alcohol directly to users, abuse of this ability is possible, and thus can cause safety issues due to the reasons listed above. However, users will have to personally supply the system with bottles of alcohol. This means that the user will have to be able to legally purchase alcohol, and there are already several laws in place to guard against illegal purchase and possession. The B3 is also, in the most basic sense, a device that mixes beverages and delivers them. Though its obvious intended use is with the idea of mixed alcoholic beverages in mind, it is still absolutely possible to find use of the system with non-alcoholic beverages. For instance, it could be delivering a cold glass of lemonade on a summer day, some orange juice to go with breakfast, or even be one of the world's most expensive water fountains and simply bring the user a glass of water. The target audience for this is for use in private, non-traffic heavy areas. Given that individual use is the focus of this project, the necessity of a breathalyzer for each use is somewhat redundant, given that the sale of alcohol itself is already regulated by the government. Given that the B3 only simplifies the delivery process, the sale of the B3 itself will not need to be restricted by age.

There exist a number of other safety concerns unrelated to the impact of alcohol. Due to the electronic components that operate the system, we will need to ensure that all components remain within their required power ratings in order to prevent potential electrical or fire related injuries. Our system will be functioning on mostly DC voltages, which will require the power from the AC wall outlet to be converted and properly regulated to avoid such instances. Physical harm is also possible due to the fact that Butler will be physically moving, thus running the risk of accidents occurring, especially if the user is intoxicated. In order to ensure safety, the glass that is being delivered will be secure and the Butler must not move at a speed that is too fast to avoid high momentum collisions.

There is also a certain health risk inherent when dealing with liquids that will remain stagnant for long periods of time. The bottles of mixed drink ingredients will be effectively sealed in our design, but the primary issue remains in the maintenance of the tubing used to dispense the liquids. An automated process would run the risk of back flowing cleaning fluid into the input bottles. As such we will need to specify in a user's manual that a cleaning process should be run whenever a bottle is switched out.

4.3.5. Manufacturability and Sustainability Constraints

Though there are a few similar products to the B3 there are none that have been widely commercialized. Most of these have been targeted at public bars, and as such included the use of a breathalyzer as to assist bars with the legality of public distribution of alcohol. However, a publicly used breathalyzer requires cleaning

after every use, and even so users might be hesitant to put that in direct contact with their mouths. Requiring the use of a breathalyzer before drinking not normally a process involved in modern times and as such disrupts the mood desired by users. Rather than focusing on public use, the B3 is designed for private use and expands on that via the implementation of a delivery service, something that is more ideally applied in a low traffic, private setting. Though it does add to the overall construction cost of the product, it is still a unique service that other designs have been unable to offer and provides a much more seamless addition to the drinking experience. For this reason, it is believed that if this product was to be commercialized, that it would have much more success compared to other similar designs.

The technologies used in the B3 are well developed technologies that have proven to be very robust. The pump system used in the B3 consists of peristaltic pumps. Peristaltic pumps have been used since the 1800s and are even popular in the medical field due to their accuracy. The quantity and quality of these pumps are a non-issue and will be fairly easy to acquire over a long period of time. Both the power supply and the electronic components, though combined in unique ways, are standard parts. The loads and electrical needs for this design are straightforward and relatively small scale. Given that there are no custom components, future reproduction will not be inhibited by failure to acquire any custom components.

The application, designed to interact with the user and give orders to, will be written in Python; while the language used for the microcontrollers, to communicate actions, will be written in C. Both of these languages are common in modern times and have growing numbers of applications, as technology becomes more and more prevalent. There is widespread support for both languages, and even if these languages were to fall out of practice, there is no foreseeable reason that this will cause a failure of the B3.

5. System Design & Phased Implementation

The functional requirements defined or otherwise constrained by the design objectives and engineering specifications, constructed about the backdrop of contemporary components, render a complete engineering problem. Constrained by the set of accessible design parts, any design efforts must yield to or otherwise follow the flow of actions demanded by the functional requirements and enabled by the general design architecture.

The general design architecture is motivated by the B3's functionally distinct aspects. It must be on constant standby, ready to receive orders for beverages at the request of the host or their guests. It must be ready to produce beverages, as the user's taste may require, based upon its given ingredients. It must be capable of retrieving and delivering an order, without direct human interaction. These functionalities are each independent tasks, which may be manifested by functionally distinct subsystems.

Simultaneously, the B3 must adhere to strict design objectives and technical motivations: The design must meet strictly defined, quantifiable objectives. The design must be financially viable. The design must be along the frontier of contemporary technological achievements; whereto the B3 must remain within the scope of what may be reasonable to successfully implement for a contemporary baccalaureate team of Electrical Engineers within the course of approximately three months. The entirety of the design must be meticulously documented, diligently developed, and thoroughly validated.

All of these incentives and motivations, explored in depth in sections prior, generate a clearly defined range of design boundaries. The intrinsic nature of the engineering problem compels a clear general design architecture. From this, a roadmap for achieving increasingly sophisticated levels of functionality about a core set of functionalities is also discernible.

In this section, a core conceptual functional description is outlined. The B3 appliance, by its very definition, must yield to or otherwise functionally contain the functional description. A core conceptual system component block diagram and overall cross-functional system flow chart are thereby rendered. Functionalities are broken down by subsystem, and an assortment of components are selected from those available on the market to attain those functionalities. The available components are then assembled unto an initial design which minimally satisfies the functional requirements of the B3. This Phase I design is explored in depth. Finally, a framework is constructed to bound development efforts for features and functionalities that extend the initial core functions for a Phase II design.

5.1. Functional Description

The Bartender Butler Bot (B3) is an integration of various off-the-shelf and open-source subsystems. The focus is on rapid deployment across the integration of existing systems. The appliance itself can be thought of as two mechanical subsystems overseen by a core software application. The following conceptual narrative contains the core design concepts for the prototype:

One of the mechanical subsystems consists of a compact and configurable drink mixing station capable of consistent and efficient dispensing, dubbed the Bartender. The Bartender is capable of simultaneously pumping fluid from multiple available bottles of beverage ingredients in precise amounts in a minimal time based on the assigned drink it is commanded to pour. It can ascertain the presence of a beverage and respond to precise recipe orders as prompted by the unifying core application.

The other mechanical subsystem, dubbed the Butler, consists of a quickly configurable drink delivery bot with a simple, built-in user interface for conveniently and efficiently ordering from a customizable menu of available drinks. The Butler is capable of travelling with a beverage order between the user's preset location and the Bartender, carrying a built-in user interface which allows the selection of drinks, user feedback and interaction. It navigates with an array of sensors, can identify the presence of a cup in its serving tray, and avoids coming into contact with obstacles in its path.

The two mechanical subsystems each have an array of sensors and mechanical actuators that are each controlled by microcontrollers running an optimized and purposefully written firmware. The firmware communicates – such as selected data from the available sensors, recipe orders, pump commands, etc. – across an internet of thing (IoT) framework. The IoT framework will be supported by a host machine that will then also host an integrating software application – or user's interface and other necessary backend services.

In a typical use case, the user, at some location remote from the Bartender, selects their beverage from a menu available on the Butler's touch panel interface. The Butler leverages the core application and its backend services to provide menu options from the bottles of ingredients it has available.

The Butler accepts an order from the user with an approving notification, and then checks for the presence of a drink container on its custom serving tray. If a drink container is discernible, the Butler notifies the core application and begins its journey to the Bartender. If a drink container is not discernible, the Butler alerts the user to ensure one is placed and remains until the alarm is cleared. The core application also notifies the Bartender to expect the Butler.

The Butler will navigate across an explicitly laid out path – one preconfigured by the user to be nominally free of static obstacles. Upon successfully arriving at the Bartender, the Butler will ensure its drink tray is positioned on the appropriate plane beneath the Bartender’s dispensing nozzle(s) and notify the core application. The Bartender will acknowledge the Butler’s arrival and simultaneously verify that a valid container is indeed secured at the appropriate position and notify the core application.

Once the core application receives notification that all checks have been made and the system is ready, the recipe order will be communicated to the Bartender and the mechanical subsystems will engage to dispense the beverage – as specified by the user – within some tolerance. After the beverage has been successfully dispensed, the Butler disengages from the Bartender and returns to its path. The core application monitors the process and reserves the right to pause, interrupt, or otherwise terminate the procedure – such as by user request, or system error. The core application also maintains records of inventory – to facilitate maintenance and operation for the host or configurator.

Finally, after the Butler has returned to its initial location – such as an entertainment room – it alerts the user of the arrival of their order. The Butler releases the order with an affirming notification and enters into a standby state to await new orders. If the Butler is met with unforeseen difficulties on its journey, such as a malfunction in motion, the sudden loss of the beverage, or the discovery of unavoidable object, it will alert the user. If the Bartender’s inventory is depleted, or it has been some time since the last cleaning cycle, it will alert the user.

5.2. System Conceptual Block Diagram

The following block diagram conveys an overview of a conceptual model divided into the three main systems: The Bartender, the Butler Bot, and the core Application. The block diagram is a visual representation of the functional description above and serves as a conceptual framework for setting reasonable boundaries on the scope of open-ended design.

This conceptual framework provides adequate boundaries on the design, such that a wide variety of options for initial implementation can be explored, but also such that the scope is reasonably limited for the design and development time.

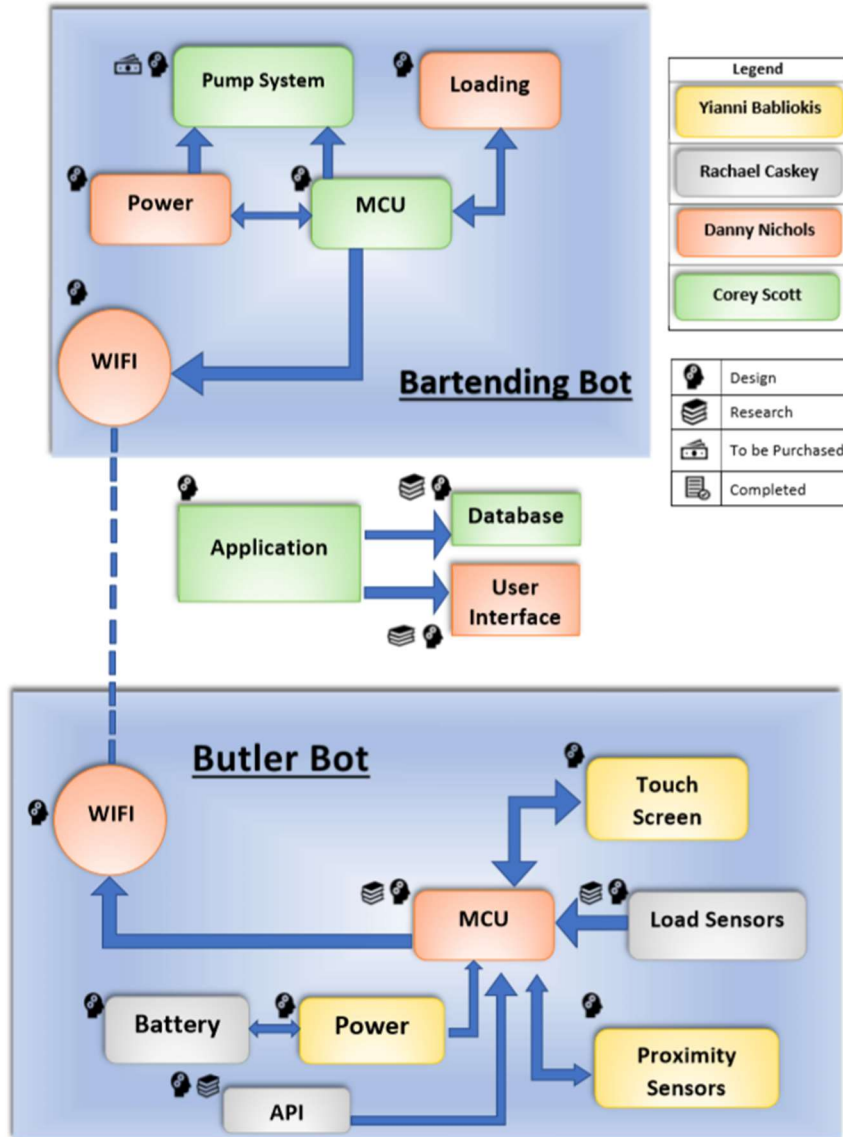


Figure 5-A: Overview Block Diagram

For administrative purposes, **Figure 2-A** maintains a color-coded legend for the separation of accountability and labor in the implementation of this project.

5.3. Strategic Components & Parts Selection

This section details the options that were considered for components in each distinct piece of the appliance, separated by major appliance purpose, with a particular focus on the electronic or mechatronic hardware which may be required to achieve the requirement specifications for a Phase I implementation.

A motivation or function is presented, its purpose is explained, and contemporary hardware options that may suit the functional purpose of the B3 are discussed from the relevant components & technologies in the previous section. Ultimately, the final component selection for the requisite function is revealed with additional technical details and reasoning.

5.3.1. Bartender

This sub-section details the parts selection process for the Bartender, broken down by functional sub-grouping. A core function is identified, and its technical requirements are qualified. A suite of components that suit the purposes of that function are selected from the contemporary market are then compared. A final selection is made based upon the incentives or technical merits evident: a component is chosen such that it provides both the shortest development time to achieving the overall functionality given in the B3's functional description and qualifies to meet the engineering design specifications.

5.3.1.1. Bartending Firmware Host

The bartending station requires a computing device to host its firmware. The firmware must be on a platform robust and capable enough to simultaneously interpret sensor information and orchestrate the precisely timed digital signals that actuate and control the pumps of beverage ingredients, sense the Butler's presence, and communicate across a core IoT framework. Further, the firmware hosting embedded device must be optimized for development and rapid implementation. It must be sufficiently capable of fulfilling the operational demands of the bartending station, including processing speed, flash memory capacity, and GPIO functionalities. It is thereby natural to turn to off-the-shelf microcontrollers. There are a few possible platforms - all accessible via the C++-based Arduino IDE:

- Atmega328 + Bluetooth
- MT 7697
- ESP8266
- ESP32

Microcontroller

In the contemporary era, embedded system applications rely on general-purpose programmable ICs. These chipsets represent the backbone of any embedded system design, and the market offers a bountiful range of options across a wide range of target applications. The following contains a comparison of the options entertained for the B3's Bartending unit:

	Atmega328p	MT 7697	ESP32	ESP8266
Price	\$15.00	<\$40.00	<\$20.00	<\$17.00
GPIO Pins	23	28	34	17
I2C SPI	✓	✓	✓	✓
802.11 b/g/n Wi-Fi	w/ module	✓	✓	✓
Bluetooth	w/ module	✓	✓	
Accessible Libraries	✓	✓	✓	✓
Open Schematics	✓		✓	✓
Community Support	✓		✓	✓

Table 5–A: Microcontroller chipset comparison

Option A: Atmel's ATmega328p & Breakouts

Atmel's ATmega328p is the default contemporary choice for prototype development delivering 23 GPIO pins. The pins are characteristically multiplexed with functions including I2C, I2S, UART, PWM, and 8 channel 10-bit ADCs. It also has wake-up and interrupt pins, a programmable watchdog timer with an independent clock, and a snappy AVR 8-bit processor with a full featured and highly optimized ISA running at 16MHz. The primary limitation of this microcontroller is the lack of wireless communications modules. Hence, it might be paired with an ESP8266, or an adjacent Bluetooth module. Crucially, however, the open-source platform has a massive and vibrant community with support on a wide range of topics - which immensely facilitates development.

Option B: Espressif's ESP8266 & Breakouts

The ESP8266, is a low-cost small-profile 802.11 b/g/n Wi-Fi enabled module that can be embedded in any PCB, has a 1MB flash memory, a Tensilica L106 32-bit processor at 52MHz, and 17 GPIO pins with functions including I2C, I2S, UART, PWM, and 10-bit ADCs. With minimal effort the ESP8266 base could be expanded to include the broadest range of applications and firmware on its own; the modularity of the ESP8266 enables more flexibility with integrating optimized and special purpose SMDs on a complete system PCB. Fortunately, easily

accessible and well documented libraries exist for directly and easily interfacing with IoT communications protocols. Paired with the Atmega328p chip, the duo could coordinate a host of complicated maneuvers or logic; alone the chip provides a lightweight basis for plugging directly into an IoT core service.

Option C: Espressif's ESP32 & Breakouts

The next generation ESP32 pushes the parameters of the ESP8266 forward for a slight premium, with 34 GPIO pins, 12-bit ADCs, and a better Wi-Fi antenna. Naturally, its pins are multiplexed with functionalities including I2C, UART, PWM, and SPI. Again, the modularity of this approach enables a wide range of flexibility with integrating special application ICs or SMDs. At this stage, the ESP32 is nearly a mature system on a chip (SoC) on its own. Its open documentation is already rich and the IoT communities it is popular in are both robust and vibrant. The Arduino IDE, the accessible development platform of choice, has easy compiling integrations and constantly updated libraries. This popular chip - conveniently prepackaged by many - offers a powerful platform to base the bartending station's firmware and is at a value high enough to be an attractive option.

Option D: MediaTek ARM 7697

The MT 7697 is an extremely low-cost but powerful ARM Cortex-M4F based system on a chip (SoC) board that runs at 192MHz, with 4MB flash memory, has both 802.11 b/g/n Wi-Fi and Bluetooth. It supports 28 GPIO pins with multiplexed functions including I2C, I2S, UART, PWM, and 12-bit ADCs. The developers that maintain performance is superior to all comparable units and even ported a simple-to-integrate library for the Arduino IDE to improve user accessibility and encourage its usage. Its community is somewhat niche, and the support is limited, but its performance characteristics are superb for the value.

Microcontroller Final Selection: Esp8266 Breakout

The Feather HUZAZH with ESP8266 is a complete open-source development board from Adafruit based on the ESP8266. At 51mm x 23mm x 8mm, with only one side of surface mounted components, the board has a compact profile. Further, access to 9 broken out GPIO pins with I2C and SPI functionality enables the use of an adjacent multiplexer to control a large number of peripheral devices. A pre-existing on-board regulated power supply with room for an optional battery pack provides a ready safeguard to safely powering the microcontroller and its digitally controlled peripheral devices from a less robust or carefully designed system power source. Further, the onboard USB-Serial converter IC supports a blazing 921600 maximum baud rate for boot loading, while convenient status LEDs and an auto-reset button provide yet another quick way to facilitate development.

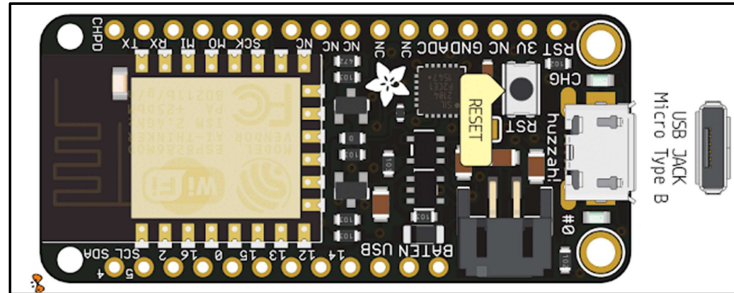


Figure 5-B: Feather HUZAZH with ESP8266

Image courtesy of: [Adafruit](https://adafruit.com)

The most important characteristics of the microcontroller platform for this project is the accessibility of its documentation, the breadth and scope of support from its vibrant community, and the ease with which one might integrate the complete and entire development board design with an application-specific circuit on a PCB. Adafruit's breakout of the ESP8266 comes with vendor-developed board definitions for the Arduino IDE, and easy integrations to popular IoT frameworks. For these reasons, the Feather HUZAZH with ESP8266 from Adafruit is an obvious and functionally appropriate choice for the B3.

5.3.1.2. Dispensing Fluids

The bartending station must dispense fluids from bottles of spirit, soda, juice, syrup, or flavoring. The fluid must be dispensed precisely, accurately, and preferably at a sufficiently prompt rate. Further, the dispensing system needs to be hygienic and easy to clean. It would be convenient for its control to be both precise and immediate. Fortunately, variations on the approach to automating a drink mixing station generally converge neatly upon some combination of the following:

- Pump(s)
- H-Bridge Motor Drivers
- Optocoupler Relays
- Multiplexers
- F&B Flexible Tubing

Pumps

This component is the specific electronically controlled mechanical sub-system that will push beverage ingredient fluids from their respective bottles through a nozzle to the beverage container. This component must be compliant with hygienic standards, be cost effective, and maintain a sufficient flow rate to support the design objectives. It is a standard and generically available part with a large number of suppliers, across a wide range of performance characteristics. The following contains a brief technical and qualitative comparison of pumps available on the market, such as those which may meet the particular functional requirements of the B3 aforementioned.

	Uxcell Self-Priming	Generic Peristaltic	Uniquers Peristaltic
Price (ea.)	\$7.12	\$24.95	\$16.49
FDA		✓	✓
Voltage	12V DC	12V DC	12V DC
Flow Rate	~2.8L/min	100mL/min	100mL/min
Unit Weight	110 grams	200 grams	200 grams

Table 5–B: Pump comparison

Option A: Self-Priming Pump by Uxcell

Self-priming pumps appeared multiple times in the documentation available for the projects and products explored. This warranted further investigation into the product as an option for this system. This type of pump clears its lines, if/when air gets in them, on its own without outside/manual intervention. This method could be beneficial to the pump system designed for this project. The pump also provides a flow rate of about 2.8L/minute, is relatively small in size, and is on the lower end of the cost spectrum.

Option B: Peristaltic Liquid Pumps by Adafruit

By design, the liquid being dispensed never touches the inside of a peristaltic pump, thus facilitating compliance with FDA standards. A high-quality motor for this pump can be hit with a PWM signal to achieve the desired flow rate, within the range of the pump output capacity. This pump is also said to integrate well with other components that are being considered for this system.

Food & Beverage Flexible Tubing

This component is complementary to the pump selected above. It serves as the dedicated conduit from the bottles of beverage ingredients through the pumps and to the dispensing nozzle. This tubing must be amenable to the same hygienic standards, accessible to quick cleaning, and optimally sized for the pumps' fittings.

	Adafruit Silicon Tubing	Vinyl Tubing	Kynar® PVDF	McMaster-Carr Silicon Tubing
Price (per Meter)	\$3.50	\$0.89	\$14.58	\$3.16
Material	Silicon	PVC	PVDF	Silicon
FDA Approved		✓	✓	✓
Offered in various sizes and lengths		✓	✓	✓

Table 5–C: Flexible tubing comparison

Option A: Silicon Tubing distributed by Adafruit

Silicon tubing conveniently sized for and bundled with the peristaltic pump offering from Adafruit was a natural choice. Tube length of one meter and a diameter of 2.5mm (internal diameter) / 4.7mm (external diameter), for this particular tubing, was used in this combination to ensure maximum efficiency with the pump. Due to this combination being sold together, the ease of integration of the two was thought to be seamless. Additionally, relative to the other possible options, the tubing fell within an ideal price range. The fall back of this option was the fact that it was not a sterile tubing, and thus was not FDA approved. Our project must meet certain standards to be considered safe. By the Health and Safety standards, the tubing for our design must be FDA approved to meet these standards.

Option B: PVC Tubing (Vinyl Tubing) manufactured by FreelinWade

The FreelinWade company offered a product made of an alternative material with promising benefits. This tubing is made up of Poly-Vinyl Chloride, also called PVC or simply shortened to Vinyl. This material provides a clear tubing that is prized on providing efficiency, cleanliness, and longevity. Additionally, it is an environmentally friendly and bio-based material. This product is also very versatile in the available options of the size and lengths, and it also offers the option for custom sizing. As another high point, this tubing checked off a major box for our project by being compliant with regulatory standards such that it is an FDA approved product.

Option C: Kynar® PVDF by FreelinWade

The Kynar PVDF, or Polyvinylidene Fluoride, is another option available through FreelinWade. This Kynar option is comprised of a material that is an alternative to Teflon and is also available in a wide variety of sizes and lengths. Also, a key product feature in which this tubing possesses is being FDA approved.

Option D: Silicon Tubing by McMaster-Carr

Silicon Tubing by McMaster-Carr is offered in various sizes and lengths that would be optimal for the B3. Their tubing is frequently used in combination with peristaltic pump applications, such as the one selected for the pump system, to provide efficiency. A notable feature of this tubing is that it is FDA approved. Taking this into consideration along with the features previously stated, this tubing appears to be a valid option to be implemented into our design.

Pump & Tubing Final Selection

After considering the options, the silicon tubing from Adafruit was first selected and purchased. It was then used within our testing phase. Unfortunately, upon further investigation, the tubing was found to be not FDA approved. This prompted an additional purchase of an FDA approved tubing. The chosen FDA approved tubing was an offering of generic silicon tubing by McMaster-Carr.

Taking into consideration that fluids may not come in direct contact with the pumping mechanism itself, peristaltic pumps were found to provide a neat, prepackaged, electrically powered mechanism for transporting fluids hygienically from bottles to drinking glasses. With a focus on rapid implementation, the low-cost 12V DC peristaltic pumps distributed by Adafruit are an obvious selection. Paired with FDA approved silicone tubing from McMaster-Carr, the system is sanitary for food and beverage usage.



Figure 5-C: 12V DC Peristaltic Pump
Image courtesy of: [Adafruit](#)

A conventional peristaltic pump runs on DC power; optocoupler relays offer a convenient on/off power switching mechanism for safeguarding and controlling the pump, while dual H-bridge motor drivers provide the electronic mechanism for accurately and precisely controlling the available range of flow rates via an application of PWM. The options identified for these two components are listed and described below.

Relay

Relays are electromagnetically controlled switches capable of rapidly and accurately controlling the flow of current in a circuit. Many times, this takes the form of a MOSFET or BJT as they are popular methods for using voltage and current to open and close paths for current to freely flow. For the pump system of the Bartender, the circuits controlling the pumps themselves will need to be opened and closed based on digital logic from the microcontroller so that the pumps can be turned on and off on command.

	Optocoupler Relays by KNACRO	Optocoupler Relays by MCIGICM
Price	\$7.00	\$8.00
Operating Voltage	12V DC	5V DC
Multiple Channel	4	2
Accessible Schematics		✓
Accessible Documentation	✓	✓

Table 5-D: Relay module comparison**Option A: Optocoupler Relay Module by KNACRO**

The first option for relays is a 2-Channel Relay Module DC 12V with Optocoupler Isolation, manufactured by KNACRO. This relay has a trigger current of 2-4mA, as well as a trigger low level of 0-1.5V and high level of 2.5-5V. The main motivation for this option was its compatibility with Arduino, at 5V logic. This compatibility provides an ease of integration with the testing board and the rest of the components of the pump system.

Option B: Optocoupler Relay Module by MCIGICM

The second option for relays was found to be a DC Relay Module with Optocouplers on a SONGLE platform, manufactured by MCIGICM. By observing the comparison **Table 5-D**, the boards are very similar in most aspects, including price range. Again, as a module compatible with Arduino, that being a sought-after feature, this provides a simple process of integration with the testing board and all the additional components that make up the pump system.

Relay Module Final Selection

A 2-Channel DC 5V Relay Module with optocouplers distributed by MCIGICM based on a SONGLE platform provides a simple development component for prototyping and proof-of-concept validation. An open-source relay module or functionally equivalent alternative, such as a power switching MOSFET, with documentation and resources ready for integration in a wider system PCB has yet to be identified and is earmarked for further research and validation.

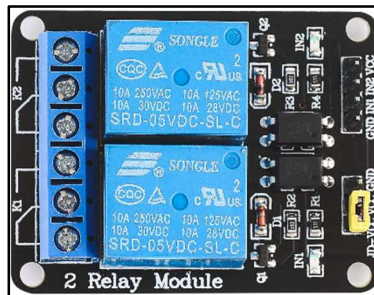
**Figure 5-D:** 5V 2-Channel DC Relay Module

Image courtesy of: [Amazon and MCIGICM](#)

DC Motor Drivers

Although relays provide circuit-based control of the open/closed nature of circuits such as those controlling the pump system's motors, the motors themselves draw such high DC voltage that they cannot be directly controlled through the same circuits being used to run the digital logic which opens and closes these circuits. As such, DC motor drivers serve to selectively raise the operating voltage provided to the motors from that presented in the digital logic circuits to an actually useable level. These motor drivers will in turn be what is directly controlled by the relay

modules. As described above, a common method for easy control of these types of components is the H-bridge motor driver setup, seen again in the options considered below.

	L293D Breakout by Texas Instruments	L298N Breakout by Qunqi
Price	\$3.34	\$6.89
Current	600mA	2A
Voltage Range	4.5V to 36V	5 V to 35V
Compatible with Microcontroller	✓	✓
Accessible Schematics	✓	✓
Number of Channels	4	2

Table 5-E: DC motor driver comparison

Option A: L293D Motor Driver Breakout by Texas Instruments

The L293D quadruple high-current half-H drivers from Texas Instruments is a small-profile IC designed to provide bi-directional currents of up to 600mA across inductive loads such as solenoids, DC motors, or pumps. As seen in **Table 5-E** above, it carries 4 DC output channels, and has a voltage range of 4.5V to 36V.

Option B: L298N Motor Driver Breakout by Qunqi

Although similar in overall functionality to the L293D Breakout, it can be seen in **Table 5-E** above that the major differences seen in the L298N are a higher price point at \$6.89, a significantly higher maximum output current (with a similar voltage availability leading to much higher potential maximum output power), and a lower number of available channels for output, with only two against the L293D’s four.

DC Motor Driver Final Selection

When it came to the final decision between these two breakouts, the greater number of controlled channels at a lower overall cost provided by the L293D Breakout outweighed the greater power delivery available with the L298N in terms of applicability for this project. As multiple pumps will all need to be controlled, and they will each require relatively low power well within the range provided by the L293D, the L298N is noticeably less cost-efficient for overall control of a set number of pumps and does not provide any necessary greater functionality. Therefore, the L293D Breakout from Texas Instruments was selected.

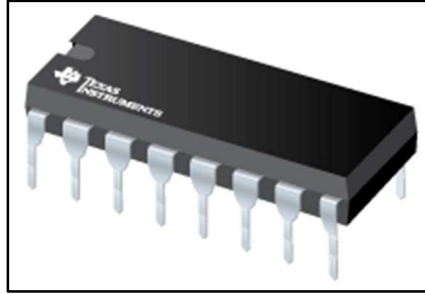


Figure 5-E: TI L293D H-Bridge Motor Driver

Image courtesy of: [Texas Instruments](https://www.ti.com)

Multiplexers

Multiplexers are extremely versatile components which can be applied in many circuit systems to process multiple inputs or outputs in order to manage more channels of information than a circuit may initially be capable of. The Bartender's base design only necessitates three distinct bottles of ingredients to select from, but based on the similar projects explored, a clear improvement to be made in later models will be the addition of new pumps. A reliable multiplexer will assist greatly in the scalability of this project, avoiding mass redesigning and retreading when trying to expand the number of pumps in use.

	3:8 MUX IC by Nexperia	3:8 MUX Breakout by SparkFun
Price	\$0.52	\$2.50
Compatible with microcontroller	✓	✓
Accessible Schematics	✓	✓
Accessible Documentation	✓	✓
Voltage Range	2V - 6V	2V - 10V

Table 5–F: Multiplexer comparison

Option A: Through-hole 3:8 Multiplexer IC by Nexperia

Considering the limited number of GPIO pins of the selected microcontroller, the multiplexer is used to overcome these limitations. The 8-input, 3-state multiplexer from Nexperia provides quick compatibility, a supply voltage range of 2V to 6V, and an abundance of documentation. Though it is significantly cheaper, we would have to break it out ourselves, costing time and effort into learning to process of breaking out the multiplexer. Furthermore, there is the potential of incorrectly breaking out the circuit since group members have not done so before, which could lead to delays and errors in the B3.

Option B: Broken-Out 3:8 Multiplexer module by SparkFun

The option from SparkFun is an open-source breakout of the 74HC4051 MUX from TI. It is designed for either digital or analog signals and has a voltage supply range within typical use by digital circuits. Hence, it is easily compatible with any development board and easily integrated into a larger system PCB. Though this breakout is for a 3:8 mux, the logic used essentially requires 4 input pins to create the 8 outputs. The reason for this is that 3 of the input pins are simply to control which output pin will receive data transmitted by the fourth. Though this is not optimal, it still increases the number of available pins and is established as a finished product capable of immediate use.

Multiplexer Final Selection

Given the limited number of GPIO pins on any microcontroller, in general, multiplexers would provide a simple GPIO pin switching mechanism to allow for control of multiple pumps from a limited number of pins. An open-source 74HC4051 8-channel MUX breakout from SparkFun offers an accessible and easily integrated component for this purpose. Given that this project will likely only need a few more of the GPIO pins, the slight inefficiency of this pre-made breakout is greatly outweighed by its ease of use and the marginally more expensive cost. Therefore, this option was chosen to be implemented into the prototype design of the B3.

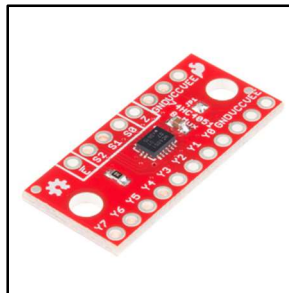


Figure 5-F: SparkFun 3:8 MUX Breakout

Ultimately, the bulk of the open-ended development effort regarding the controlled dispensing of fluid from the bartending station is related directly to the level of sophistication in the firmware of the station's electronic system and the physical/structural layout of the bartending station in relation to that of the Butler bot.

5.3.1.3. Proximity Sensing & Ranging

When the Butler bot arrives at the bartending station, the Butler will have to “dock” into a standard and predetermined location under the fluid dispensing nozzle to ensure that fluids are being released in the desired container - without spills. Hence, the bartending station must be able to verify the Butler's presence and validate the presence of a container. To achieve this functionality, it is necessary to ascertain the exact physical location of the Butler in relation to its “docking station” and

identify the presence of a drink container in its strict predetermined location. As discussed in the previous sections, some combination of the following can be used:

- Hall Effect Sensors
- IR Near-Field Proximity Sensor
- Ultra-Sonic Proximity Sensors
- Digital Camera + Open-CV

Hall Effect Sensors Final Selection

Wireless communications and systems on the Butler shall independently navigate to the “docking” location, but for redundancy, the bartending station should be able to independently verify that the Butler “docked” appropriately with a valid and open drinking container. Hall effect sensors placed in strategic locations around where the Butler’s cup-holding serving tray meets the bartending station would be activated when Butler’s base docks with the bartending station appropriately.

	MPU9250/6500 Module by: HiLetGo	MPU9250 Breakout by: SparkFun
Price	\$8.49	\$14.95
Compatible with I2C	✓	✓
Compatible with Arduino	✓	✓
Degrees of Freedom	9 DOF	9 DOF
Accessible Library	✓	✓
Accessible Schematics	✓	✓
Accessible Documentation	✓	✓
Voltage Range	4.4V - 6.5V	2.4V - 3.6V

Table 5–G: Hall effect sensor comparison

Range Finding Sensors for Validating Cup Placement

The Butler shall independently ascertain the presence of a beverage container, and algorithmic safeguards in the Butler’s firmware shall prevent the Butler from navigating to the Bartender if a valid container is not present. However, for redundancy, the Bartender shall itself validate the presence and position of the cup. As the fluid dispensing nozzle on the bartending station does not move, it is necessary to ensure an open beverage container with adequate volume is placed directly beneath it. As with automated water bottle filling stations, infrared (IR) proximity sensors or ultrasonic (SONAR) proximity sensors placed strategically at the height of a beverage container and directly facing where the beverage container is expected to be.

	VL6180	TCRT5000 3-pin
Price	\$25.95	\$9.50
I2C	✓	✓
Voltage Regulator	✓	
Measuring Distance Range	~100mm	1mm - 8mm
Accessible Library	✓	✓
Accessible Schematics	✓	✓
Accessible Documentation	✓	✓

Table 5–H: Infrared rangefinding sensor

Option A: VL6180 infrared range finder breakout by SparkFun

The first option for the IR sensor is the VL6180 infrared range finder. This module is compatible with Arduino and can achieve communication by an I2C interface. To accomplish valid communication, the module consists of an integration of an IR emitter, a range sensor, and an ambient light sensor. Having these features, as well as a few others, increase the desirability of this product.

Option B: TCRT5000 3-pin IR sensor module by Diymore

The second option in this section is the TCRT5000 3-pin IR Sensor Module. This module is also compatible with Arduino. By observing the comparison table below, various similarities between the two options are noted; however, a key difference that is evidently seen is the measuring distance.

	LV-MaxSonar (MB1000) EZ0 by Maxbotix	Ultrasonic sensor by Seeed
Price	\$29.95	\$15.00
Compatible with Arduino		✓
Voltage Regulator	✓	✓
Detecting Range	0 - 6.45m	3cm - 4m
Accessible Library	✓	✓
Accessible Schematics	✓	✓
Accessible Documentation	✓	✓
Voltage Range	2.5V - 5.5V	5V

Table 5–I: Ultrasonic rangefinding sensor

Option C: LV-MaxSonar (MB1000) EZ0 range finder by Maxbotix

The first option for an ultrasonic sonar sensor is the LV-MaxSonar (MB1000) EZ0 range finder sold by Maxbotix. This sensor has variations used for short- and long-range detection. It has a substantial detection distance measuring from zero to approximately six and a half meters. This is a key feature to consider for this system. This product also comes with the means of reducing interference when integrating multiple sensors into a system. The product-based solution is called chaining, and this has three optional methods that can be applied to achieve this solution. The noise mitigating and easily implemented features offered by this product line could facilitate development.

Option D: Ultrasonic sensor by Seeed

The second option of this section is the Ultrasonic sensor by Seeed. This is included as an option due to its prominent compatibility to Arduino. Using the Arduino community assets, aids the integration of this product. Additionally, this sensor possesses an acceptable range of detection.

Range Finding Sensors Final Selection

A clear option for presence sensing is the open-source VL6180 infrared range finder breakout from SparkFun. It features a linear range-finding curve for a broad range of surface reflectance within approximately 100mm and an easily integrated I2C control interface across an Arduino library. The documentation is thorough and production documentation is available from the vendor, simplifying any possible integration with the system PCB, thus this was the product chosen for the design.

As an alternative to IR proximity sensors, presence sensing on the bartending station may be achieved with the LV-MaxSonar (MB1000) EZ0 range finder from Maxbotix. Implementation requires the use of an RS-232 serial interface enabled pair of GPIO pins on the microcontroller and firmware-based calibration, but the documented range of accurate usage extends well beyond 1m, even for smaller objects like beverage containers, and could further facilitate control of the Butler's approach vector. However, the operational distance must exceed 150mm, which would impose a structural design constraint.

Ultimately, both the IR rangefinding sensor and ultrasonic sonar rangefinding sensors were selected for direct testing. Once the results of the two have been gathered, the team will evaluate the elements of comparison and determine which of the sensors is best for the project and selected to be implemented into the final design.

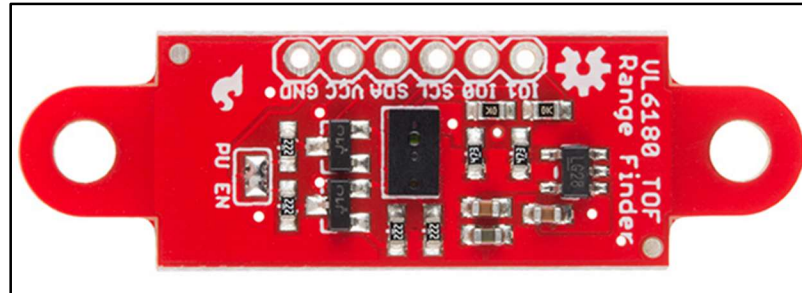


Figure 5-G: VL6180 SparkFun Breakout
Image courtesy of: [SparkFun LLC](#) & [ST Microelectronics](#)

5.3.2. Butler

This sub-section details the parts selection process for the Butler, broken down by functional sub-grouping. As before, a core function is identified, and its technical requirements are qualified. A suite of components that suit the purposes of that function are selected from the contemporary market and then compared. A final selection is made based upon the incentives or technical merits evident: a component is chosen such that it provides both the shortest development time to achieving the overall functionality given in the B3's functional description and qualifies to meet the engineering design specifications.

5.3.2.1. Autonomous Navigation

The Butler bot's key operational requirement specification is that it shall be able to autonomously deliver a mixed beverage to the user at a location remote from the bartending station. This operational requirement requires a sophisticated and multi-faceted approach, which intrinsically includes complex mechanical elements and well-developed firmware. In alignment with the objective to maximize off-the-shelf components and facilitate implementation as much as possible, the following components shall conceivably be used:

- iRobot Create 2.0
- Raspberry Pi 4/Zero
- Infrared Photodiode Sensors
- 6-Axis Accelerometer
- Digital Camera + OpenCV

Robotic Platform

The scale of transport the requirements of the Butler present physically make the course of its design lean heavily towards some programmable robotic base which is motorized to be able to provide both sheer power in travel time and load-bearing while also providing high scalability with regards to the navigation systems which guide it. Through whatever means, the Butler will need a base which can carry itself, a significant frame, and a drink with precision and smoothness that can make this project truly suitable for an actual home.

	Custom Robotics	iRobot Create 2.0
Cost	~\$250+	~\$200
Time Commitment	> 240 hours	> 100 hours
Homing Feature		✓
Flexible Development	✓	
Variety of Sensors	✓	✓

Table 5–J: Robotic platform comparison

Option A: Original Design from Scratch

The first option considered for the motorized base component of the Butler unit was a completely original design, with motors, sensors, and controls all purchased separately and integrated by hand. This option was estimated to have an overall cost in the neighborhood of \$250 but would involve an extremely large amount of time developing and testing each aspect of the design. Because of this, this option was quickly disregarded upon finding the iRobot Create 2.0, and this component selection wound up being one of the fastest made in this project.

Option B: iRobot Create 2.0

The iRobot Create 2.0 seemed to be a perfect fit for this project upon finding it. The unit cost \$200 in total, and with its standard components provided a motorized base capable of all the necessary motions the Butler would need to make, a frame capable of custom mounting procedures for building up the rest of the Butler's frame, and an open IED for programming the entire controls of the unit and integrating them into the Raspberry Pi and other modules used. Crucially, the unit has an array of built-in sensors that allow for recognizing objects at the ground level, impact sensors, and a homing feature for locating its charging station, directly available through the command line terminal interface over UART which can be integrated with the Butler's firmware host for easier navigation.

Robotic Platform Final Selection

As the incentive is to select an existing robotic platform to minimize the design effort regarding the mechanical elements, a natural choice for initial development is the iRobot Create 2.0 Development Roomba, seen in Figure 5-H. Its accessible full-featured serial command-line interface removes the technical imperative to program a functional robotic platform from scratch – freeing up valuable development time. It features predetermined and well-documented mounting points such as for attaching a structural frame for supporting additional electronic hardware and serving tray, along with full control of its locomotive actuators, and access to all of its embedded sensors' data.

Crucially, it already has smooth, well-developed, and commercially validated **built-in functions like “seek docking station”**. Moreover, the iRobot Create 2 has an active support channel maintained by the vendor and a broad array of documented DIY-style projects. Ultimately, the iRobot Create 2 offers the maximum value in terms of both ease of implementation and price.

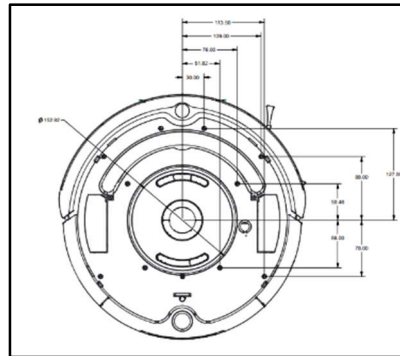


Figure 5-H: iRobot Create 2.0
Image courtesy of: [iRobot](https://www.irobot.com)

Navigation Firmware Host Final Selection

The Butler is the physical host of the navigation firmware and the graphical user interface. Crunching data for autonomous navigation, interfacing with the robotic platform, while simultaneously providing a core for the user interface requires a lot of computing power in one physical unit. In the pursuit of simplifying implementation efforts, minimizing the total cost of the system, and the number of distinct pieces of hardware, it is thereby compelling to base all of the computing resources in a single physical unit. Naturally, the focus turns to self-contained development-oriented platforms with the computing capacity and memory space to handle these tasks. Of these, including Intel’s NUC, Samsung’s NanoPC-T3, or Huawei’s HiKey 4, none are as functionally competitive, cost-effective, or accessible as the Raspberry Pi Foundation’s quintessential Raspberry Pi platform.

In particular, The Raspberry Pi 4, seen in **Figure 5-I**, offers an affordable and neatly packaged development platform with the capacity to run sophisticated peripherals – with a dedicated and conveniently available breakout of its GPIO pins. With a microSD card, it can host a full-fledged operating system all while natively supporting 40 broken out GPIO pins with multiplexed functionalities including PWM, I2C, and SPI. It has native support for HDMI video, touch screen LCDs, and 3.4mm audio. With up to 4GB of RAM and a quad-core Cortex-A72 (ARM v8) 64-bit processor running at a blistering 1.5GHz, the Raspberry Pi 4 is a perfect host for both navigation firmware, IoT core applications, and the user interface. For these reasons, the Raspberry Pi 4 is the obvious choice for development at this stage.

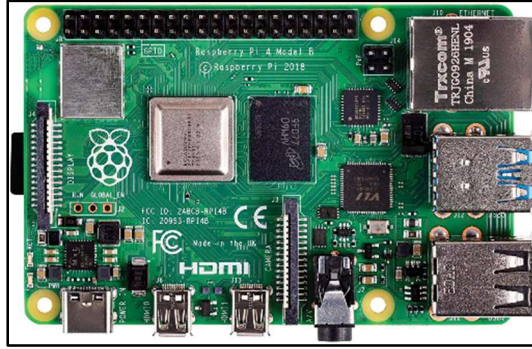


Figure 5-I: Raspberry Pi 4
Image courtesy of: [Raspberry Pi Foundation](https://www.raspberrypi.org/)

Navigation Sensors Final Selection

Navigating autonomously intrinsically requires an awareness of the environment being navigated. One possibility is to employ crude light intensity sensors that could be used to enable the bot to travel along a clearly marked path or otherwise identify overhanging obstacles that could interfere with its movement. Another possibility is to employ a sophisticated digital-camera based computer vision algorithm. Ultimately, the technical approach to autonomous navigation is intrinsically linked to the quantity and quality of information available from the sensing approach. The primary constraint in that effort, invariably, is the explicit objective to frame the inevitable design within the scope of what is achievable with limited resources and experience. Hence, the selection of sensors, and thus approach to navigation, falls into two categories – one that functionally satisfies the engineering requirements specifications, and one that exceeds them. This aligns with the phased, or staggered approach to development.

Phase I: Line-Following

For this purpose, an array of strategically placed IR Infrared Reflective Photoelectric Light Intensity Sensor Modules from GikFun, are quickly employable and easily integrated. The sensors would require access to ADC enabled GPIO pins on the microcontroller (or SoC), as they generate an analog voltage from the measured light intensity. A sudden change in light intensity at any sensor, in relation to the rest of the array, could signal the presence of an obstacle or the identification of a path marker.

These sensors, seen in **Figure 5-J**, and the limited range of navigation algorithms that are enabled by them, are already fairly well-documented. Their schematics and fundamental construction are simple, easy to tailor to suit a specific application's demands, and widely available. Further, the line-following navigation approaches enabled by these sensors vary in sophistication – range from simple conditional motion algorithms to full control system feedback loops. It is for this reason that these sensors are selected for a Phase I, core minimum functionality

design, as the open-ended question of the level of sophistication in implementation is largely unconstrained in software.

Phase II: OpenCV & Path Markers

As noted previously, another possibility would be to use an application of computer vision, via the OpenCV platform across Python in conjunction with a digital camera. The Raspberry Pi has native support for a digital camera and the capacity to host advanced high-level language environments directly. Provided the definitive selection of the Raspberry Pi, or the future integration of an equally – if not more – capable SoC, a Phase II approach would extend the navigation capability to

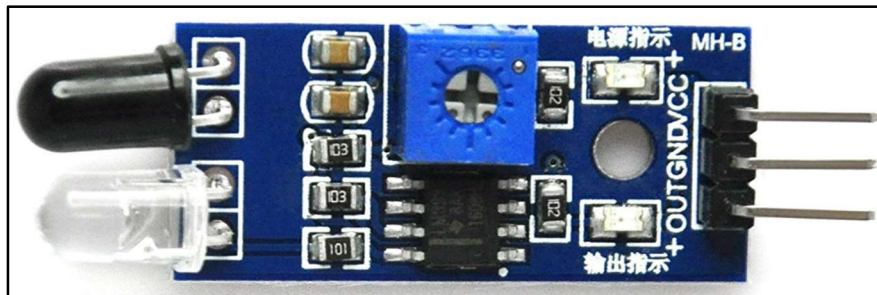


Figure 5-J

Figure 5-K: IR Photodiode Sensor Module

Image courtesy of: [Gikfun](#)

5.3.2.2. Securing & Validating Cup

The Butler represents a large mechanical unit. It must contain a serving tray, or dedicated cupholder, to secure the beverage. The serving tray and its beverage container must be reliably positioned beneath the dispensing nozzles of the Bartender. The Butler must also be aware of whether the beverage is or isn't present – or that a container is or isn't present.

To ensure that the provision of a beverage by the Bartender only occurs when there is a valid container available for filling, the Butler must be able to independently identify and validate its presence and position. To achieve this functionality at various levels of sophistication, the following components may be used:

- Wheatstone Load Cell
- Load Cell Amplifier
- IR Proximity Sensor
- Accelerometers
- Digital Camera + OpenCV

Presence Sensing Sensor

In order to keep track of higher-level inputs of the system past standard drink delivery, such as the successful placement of an empty cup in the Butler or a failsafe to ensure the Bartender is not about to overpour a beverage, some type of weight-sensing is needed. A simple load sensor can be used to determine weights within a certain defined range consistently, and its input can easily be connected to a microcontroller so as to use that data to produce a binary response of the placement of a drink or lack thereof. Alternatively, simple proximity sensors

	TAL220B	TAL221
Cost	<\$11	<\$9
Material	Aluminum Alloy	Aluminum Alloy
Max Weight	5kg	100g

Table 5–K: load cell comparison

Option A: SparkFun TAL220B Load Cell

This SparkFun distributed load cell is sold on their website for \$10.95, and is made from an aluminum alloy, offering sturdiness and a measurable weight range up to 5kg. The unit has four wires that are clearly color-indicated that run from its Wheatstone layout out to open leads for integration into the selected amplifier board. The unit has good reviews, and seems all-around to be a standard, reliable source for a basic load cell, with clear tutorials on their website on how to mount the unit to various objects to create small digital scales.

Option B: SparkFun TAL221 Load Cell

As advertised on SparkFun’s website, the TAL221 module is extremely similar to the TAL220B with regards to overall design and composition. The main key differences being that the TAL221 can measure weights only up to 100g, whereas the 221B can measure up to 5kg.

Option C: Infrared Proximity Sensor

Sensors of the same form and function as described in the Bartender section above; the same options will be considered as described in that section.

Option D: Digital Camera + OpenCV

Alternative or parallel approaches to sensing the presence of a beverage container involve the use of a strategically placed digital camera with OpenCV via Python. These components have already been discussed but are noteworthy in this context also; it may be prudent to validate the presence of the cup in multiple ways for redundancy’s sake.

Presence Sensor Final Selection

The ultimate selection for this functionality was a combination of the TAL221 Load Cell with an HX711 Load Cell Amplifier. Whereas the TAL221 is sold for only \$8.95

on the SparkFun site, as opposed to the \$10.95 TAL220B. Though cheaper, the TAL221 simply does not offer the necessary weight range for this project, as a filled cup will most definitely weigh more than merely 100g. For this reason, the TAL220B was selected as the load cell for this project.

It was realized earlier in the research for a load cell amplifier that nearly every amplifier of the general size, price point, and range of applications relevant to this project that could be found was a variation of the HX711 chip, with different breakout boards being the only true selection that was open to make. Of the breakouts available, the options were limited down to two variations. One manufacturer was selected primarily because of their reviews, which proved to reflect them being the most reliable relative to their competitors. Since each breakout board was virtually identical in the actual design of its components, this reliability factor proved very important, as some options were excluded early on for negative reviews stating malfunctioning or damaged parts being received by past customers.

The first of these two HX711 breakouts was the version offered by SparkFun, an established supplier of electrical components. They described their breakout on their website as having a clear 2-wire interface for clock and data, and a simple connection procedure for integrating a Wheatstone bridge load cell, with the four wire connections clearly labelled. A single unit cost \$9.95 as advertised on their website, and they additionally offered many in-depth, step-by-step guides for connecting the load cell to the amplifier and performing tests on it.

Serving Tray Tilt Sensors Final Selection

Ideally, the Butler bot shall deliver the beverage without spilling it. An elevated serving tray, with a dedicated cup-holder, may not be sufficient to ensure that the beverage remains entirely in its container along its journey from the Bartender to the user's remote location. Ensuring that the serving tray and beverage remain relatively level is paramount to preventing spills. For this purpose, an I2C enabled MPU6050 6-Axis accelerometers from TDK InvenSense, placed strategically at the same elevation and orientation of the serving tray can enable the firmware host to prevent, prohibit, or otherwise mitigate motions which could cause spilling. A prepackaged breakout of the MPU6050 by HiLetGo is an accessible solution for development.

5.3.3. Core Application

This sub-section details the parts selection process for the Core Application, broken down by functional sub-grouping. As in the previous sections, a core function is identified, and its technical requirements are qualified. A suite of software packages, services, or applications that suit the purposes of that function are selected from the available suite of public options and are subsequently compared. A final selection which fulfills the functional demands is made based upon the incentives or technical merits evident.

5.3.3.1. Application Backend Environment

The core software application which shall interface with and effectively control the mechanically actuating firmware on both the Butler and the Bartender requires a host. The necessarily powerful SoC available on the Butler, which is already responsible for hosting the IoT core functions and the autonomous navigation firmware is also a natural place for the overseeing software. That overseeing software, which is itself responsible for directing the Bartender on what and how much to pour for each recipe, validating docking the sequence, and maintaining an active record of the possible concoctions from the available ingredients, requires a dynamic and open environment to reside. The primary components of that environment include:

- Operating System
- Programming Language
- Persistent Database
- IoT Core

Operating System

	Windows 10 IoT	Raspbian
Open-Source		✓
Vendor Support	✓	
Customizable Services	✓	✓

Table 5–L: operating system comparison

Option A: Windows 10 IoT

Designed for helping developers make connected devices, Windows 10 IoT is an operating system that is meant to prototype internet connected devices using a Raspberry Pi and Windows 10. In order to further help developers, the licensing fee is waived for use in prototyping, making it attractive for use in this project. It can run programs on smaller devices with or without a display. Though you cannot get the full benefits of the Windows OS, this is actually a benefit for use in lightweight devices that utilize minimal user input to generate a functional output. Furthermore, it has ties to Visual Studio, a well-developed IDE that greatly assists in developing headless (lacking a graphical user interface) programs.

Option B: Raspbian

A lean but full-fledged operating system, such as the Raspbian distro customized for the Raspberry Pi from Debian (a Linux variant) by the Raspberry Pi foundation, is the default and most easily accessible option. Raspbian was designed for the Raspberry Pi and it already comes with Python and other useful shell environments. It uses PIXEL, or Pi Improved X-Window Environment, Lightweight, as its primary desktop environment. This environment is based off of the old LXDE desktop environment, which is known for its comparatively low resource requirements. It includes support for the peripheral devices on the Raspberry Pi. Further, the operating system could be expanded or streamlined as needed to suit the application specific demands.

Operating System Final Selection

Given the overall dynamic flexibility, Raspbian was selected as the final operating system. It's compatibility with the other major components of the application core also lends itself to easy integration. Notably, MQTT can easily be configured through Raspbian.

Application Programming Language

Option A: C++

With many libraries and materials available to it, development and deployment is relatively simple. It is portable across a wide range of operating systems, especially across iterations of Windows, OSX or popular distributions of Linux. The debuggers available are detailed, robust, and can significantly assist in ensuring the viability of the program prior to actual implementation. Crucially, C++ provides great control over the resource and memory management of the system, allowing for the optimization of algorithms for particular hardware.

Option B: Python

Known for its flexibility relative to other languages, Python has the capability to save time and space when running scripts. Unlike other languages that require variables to be declared before they can be assigned, Python can compile regardless of this. Similarly, it has the capability of compiling even with errors, as the type checking is performed during run time rather than compile time, also known as a "dynamically typed" language. Though will prevent the script from running properly, the ability to compile code even with errors allows for a large amount of flexibility and testing while creating code. Overall it is a good general-purpose language that can be tailored to nearly any specific need.

Option C: Java

As an older high-level language, it has extensive libraries and materials available, allowing for dynamic and extensive creative flexibility. One of the primary attractions of Java is its automated garbage collector system, that is, the automated deallocation of memory based on what java determines as "no longer relevant." Java is optimized for object-oriented coding, allowing for variables and methods to be more easily created and referenced in code. The method of memory address pointing is also far easier compared to C++ and can simplify the process of writing code. However, due to these many features, Java programs tends to perform poorly compared to other languages.

Programming Language Final Selection

Python offers the most flexibility with its libraries for the MQTT core and SQL wrapping for Python. Furthermore, it is also a coding language that many among the group is familiar with, allowing for an easier time with the actual construction

of the application. The GUI library, PyQt is a well known library that is useful for seamlessly and easily constructing workable GUI.

Persistent Database

A database would facilitate the development of an application by providing a persistent storage location for information including configuration parameters, beverage recipes, and user history. The methodology for storing information would lend itself to quick

	SQL	Mongo
Community support	✓	✓
Mature technology	✓	
High transaction rate	✓	
Optimized for scalability	✓	✓
Optimized for failure recovery		✓
Native data validation		✓

Table 5–M: Persistent database comparison

Option A: SQL

Oracle Corporation, the same company that made Java, created SQL to be a full-featured open-source relational database management system (RDBMS). The purpose of this was to create an easily workable system of tables that could store a variety of data and perform queries across various tables. Although it's made by Oracle, it is compatible with nearly all operating systems, as well as having many libraries and resources for many coding languages. The basic structure of the database is a series of tables and rows and allows for large volumes of data to be frequently updated and modified.

Option B: MongoDB

Developed by 10gen, MongoDB is an open source, document-oriented database that stores its data in JavaScript Object Notation (JSON). This document format allows the database to be versatile, allowing easy connection to applications of various languages via its drivers. Because of its JSON format, the data transferred is in a human readable format, and its efficiency and reliability are greatly enhanced. The cost of this is that the setup for this is lengthy, and given that many existing databases use SQL, the additional process of utilizing a tool to process a SQL to MongoDB query is required.

IoT Core Framework

The IoT core framework is absolutely crucial to the communication between the Bartender, Butler, and core application. Without a consistent method of communication, the flow of the system can be broken and the intended desire of

the B3 will fall short. For this wireless system to be realistically implementable, some sort of IoT core will need to be run in the background of the core application.

Option A: MODBUS-TCP

Modbus-TCP is a byte-oriented, application layer protocol originally developed in 1979 by Modicon. It was originally designed as a simple way for sensors to transfer data to and from a control setup; however, through separate specification it now supports other communication such as TCP/IP (Transmission Control Protocol/Internet Protocol). The Modbus standard protocol does not specify how its register values are sent, and as a result there is wide variety in the interpretation of its data. Since the data types are not strictly defined, knowledge of how a device sends data is crucial for Modbus devices to accurately communicate, adding a layer of complexity. Because of its limited functionality, most Modbus device manufacturers add their own custom extensions to increase the functionality beyond what standard Modbus provides. Though this is beneficial, it also means that a familiarity with a Modbus device's unique extensions is required in order to properly work with it.

Option B: MQTT

MQTT, or Message Queuing Telemetry Transport, is an open, lightweight machine-to-machine protocol. The core network, known as the broker, mediates the interactions among MQTT agents or clients. After connecting, the clients are then able to publish information to the broker and receive information they are subscribed to. When finished, clients will then disconnect from the MQTT broker. MQTT is designed for use by resource-constrained embedded devices, and as such has a minimal footprint. The microcontrollers that will be used in this project are essentially what MQTT is designed to be run on, making it very appealing for application in the B3. On top of this, MQTT was designed to communicate efficiently, with various qualities of service that can be designated to messages to determine how it is communicated and confirmed based on both the data contained within the message as well as the importance of that message to the system. There are also a variety of libraries for all major languages for initializing and integrating an MQTT server into an application or device, simplifying the process of creating a functional prototype.

Option C: DNP3

The Distributed Networking Protocol, DNP3, similar to Modbus, is a byte-oriented application layer protocol developed by Westronic, Inc. in 1990. Unlike Modbus, it also contains a Data Link layer with a pseudo-transport layer and its protocol is simply encapsulated within TCP/IP. Also unlike Modbus, DNP defines a large number of data types, so communication between DNP devices is far more seamless. Furthermore, within each datatype there is are variations available, such as different data sizes, timestamps, and quality indicator flags. DNP3 also supports high security two-step control operations, ensuring the integrity of the control commands given to the "Slave" device.

DNP3 was specifically developed for use in supervisory control and data acquisition (SCADA) applications and is a dominant protocol in the field. Due to this, it is highly standardized, allowing for high compatibility and inter-operability between devices constructed by different manufacturers. Because the B3 will be connected to many electrical devices, this is quite an appealing method of inter-device communication.

IoT Core Framework Final Selection

MQTT is a dynamic and lightweight IoT framework. It operates as a background service with a small footprint. Configuration of tags is on-the-fly, and data exchange is centralized. It is easy to integrate with most high-level languages and often has community support for Wi-Fi enabled microcontrollers. It is open-source and free to implement. As a result it is the obvious choice for the B3.

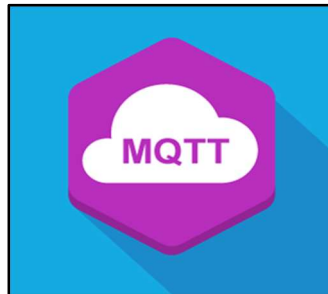


Figure 5-L: MQTT icon logo
Image courtesy of: MQTT.org

5.3.3.2. User Interface

The Butler is intended to be the primary point of contact with users or partygoers. The user experience is thus centered about the interface on the Butler. The ideal interface needs to be easily accessible, simple-to-use, and also appealing. Providing the user with an ideal interface requires thoughtful consideration of the user during what is expected to be a typical interaction with the appliance. It is conceivable that the user may not always be the owner of the appliance, who may yet be more intimately familiar with the configuration and capabilities of the appliance than a house guest. The interface needs to be simple enough for a first timer to quickly use, but robust and flexible enough to offer a gratifying experience. However, constraints on resources, especially those related to development time, also compel a focus upon ease of implementation. Ultimately, to provide a rapidly deployable, easy-to-use, and flexible user interface, the following components shall be used:

- GUI Application Platform
- Display
- Input Device(s)
- Speaker(s)

GUI Application Library

High-level programming languages, such as C++ or Python, have various libraries, modules, or packages for implementing a graphical user interface (GUI) based application. The library, package or module of choice shall be a platform for the development of the GUI application; and there are a wide range of options available, each offering unique incentives for their adoption.

The ideal platform to be within the natural constraints imposed, and to enable the achievement of the design objectives and specifications, has several distinct ideal characteristics. It must be well-documented, easy-to-use – such as by maintaining graphical tools for facilitating design, and flexible enough to provide a broad range of creative options. Further, the ideal development platform for a GUI application should also be light-weight – or built with careful and dynamic resource management in mind.

	GTK+	Qt	pyQt5	appJar	Swing
Open Source Dev.	✓		✓	✓	✓
Free Visual Dev. Toolkit		✓	✓	✓	
Community Resources	✓	✓	✓	✓	✓
Native Multi-Threading, Pipelining, Queuing	✓	✓	✓		✓
Custom Widgets	✓	✓	✓		✓
Interrupts		✓	✓		✓
Broad Compatibility with External Libraries	✓		✓		✓
Project Member Experience	✓		✓	✓	

Table 5–N: GUI library framework comparison

Option A: The C++ GTK+ library

The GTK+ library is an implementation of the popular GTK object-oriented widget toolkit and open-source GUI development library for use in a C++ development environment. The core is derived from decades old GNOME project; it sustains a robust community, an accessible GitHub-based technical/development issue handling system, and thorough documentation. There are tutorials and manuals on implementing GUIs in C++ via the GCC compiler, and on integrating the library with most C++ IDEs.

Further, the Wayland or X11 backend is popular and operational on the most popular OS platforms, including Windows, OSX and the main Linux distros. It supports the creation of sophisticated graphical applications, but the learning curve appears steep.

Option B: The C++ Qt library

Qt is a well-developed platform for implementing GUIs written natively in C++. It was a platform built by the Qt Company in the early 1990s that was later pushed forward by the increasing number of stakeholders represented by the Qt Project. The official website provides ample documentation, encouraging tutorials, and access to a dedicated community. It is a convoluted platform with many bells and whistles, including multithreading, pipelining, and event handling and is popular with advanced developers. There are great features available through the Qt and its dedicated development apps – but the best features are behind paywalls.

Option C: The Python pyQT5 library

pyQt5 is an open-source Python wrapper for the core functionalities of Qt. This version is accessible to beginners and advanced users alike, as the documentation is both vast and meticulously thorough. Further, the documentation from the original library and the free visual development tools provided by the Qt Company are compatible with this library, which may simplify design and development efforts. All of the bells and whistles from the C++ native Qt, including the object-based widget-class inheritance, multi-threading, procedural page generation, and pipelining/queuing enables are easily accessible. The toolset hence offers broad creative range, a powerful core set of functions, and the best of the rapid development environment provided by Python.

Option D: The Python appJar library

appJar is a simple GUI toolkit designed for beginners and those seeking to streamline the implementation of a GUI as much as possible. The features are limited, the widgets are strictly defined, and the backend is not accessible, but development in the creative range afforded is effectively trivial. There are various helpful tutorials and a plethora of online resources encapsulating almost the entire swathe of possibilities rendered by the platform. The best argument for this toolkit is succinct: “It just works”.

Option E: The Java Swing library

Java’s Swing library is the default “rapid implementation” option for those dedicated to development in Java. Swing is built upon Abstract Windowing Toolkit and is designed specifically for Windows-based applications. Though using this platform constrains the project to a Windows-based OS, such as Windows 10 IoT, for the application layer, it does provide a ready and full-featured platform from which to develop.

The community for the Swing platform is active, the documentation is thorough, and there are a multitude of open examples for quick implementation. Further, the modularity of the platform enables the construction of custom widgets and the usage of advanced features like ODBC integrations. However, a clear pitfall is the steep learning curve.

GUI Library Final Selection

PyQt5, a Python wrapper for the popular Qt graphical user interface development platform, offers an open-source and full-featured API for generating that user interface. It is the natural and ultimate choice for the development of a GUI application overlay to handle the interactions between the firmware on the Butler, the firmware on the Bartender, the IoT data traffic, and the user's inputs. The crucial deciding factors in the selection of PyQt5 is its accessibility, that it can be developed in the rapid development platform of simple Python script, and that members of the project already have experience working with this library - so the learning curve is not quite as steep as it might be otherwise.

Interface Display, Input & Audio

The GUI platform shall thus render its application on a display for the user to interface with. Given the circumstances of what is expected to be the typical usage, the display on the Butler ought to be vibrant, clearly visible, simple to use, and simple to integrate as part of the system architecture. Again, the predominant deciding factor in the selection process for peripheral devices supporting core functionalities is indeed the ease of implementation; options for the primary user-interface display embedded on the Butler should be as close to "plug & play" as possible, or otherwise offer clear configuration steps and documented libraries.

	TFT LCD Resistive	IPS LCD Capacitive	Mini-OLED Array
Price	\$40.99	\$79.99	\$10.99, ea.
Touch Capable	✓	✓	
Size	4"	7"	.96", ea.
Color Range	Full, RGB	Full, RGB	White
Resolution	320x480	1024x600	128x32
Speaker(s)	Purchase Separately	Built-In, Native ✓	Purchase Separately
Drivers & I/O	SPI w/ Library	HDMI USB-C	I2C w/ Library
Warranty	✓		✓

Table 5–O: Interface comparison

Option A: TFT LCD with Resistive Touch by waveshare

There are various vendors offering TFT LCD displays with resistive touch, all at various sizes and performance characteristics. However, a handy 4" offering by waveshare has key performance characteristics that are ideal, including a simple SPI interface and a configurable FBCP software driver. This display is desirable over the plethora of alternatives because of its simplicity, it's high contrast capacity for vibrant colors, and it's mounting points for a microcontroller shield. Paired with a generic off-the-shelf speaker, the option entertains a whole user interface hardware package.

Option B: IPS LCD with Capacitive Touch by EVICIV

A 7" portable USB-powered 16:9 Monitor with capacitive touch, built-in speakers, and a solid glossy plastic frame with mounting brackets by EVICIV (or similar discount brand) is another clear option. Capacitive touch provides the input capability and the built-in speakers offer the opportunity for audio feedback, all powered through simple drivers. The EVICIV monitor also features HDMI ports, flexible power options, a high-definition resolution, and overwhelmingly positive feedback. Though significantly more expensive than the rest, the completeness of the packaging and the native support for HDMI make this display simple to implement.

Option C: Array of mini-OLEDs by MakerFocus

A final alternative to displaying the graphical user interface is to distribute the tasks across an array of small, cheap, OLEDs. Receiving input could then be restricted to generic off-the-shelf piezoelectric or capacitive touchpads, and audio feedback provided through a generic off-the-shelf speaker. The I2C interface enables the simple control of multiple displays, and the low price offers a reasonable path towards implementation. This approach, however, lacks the appeal of a full-range color display and may yet prove difficult to implement in software.

Interface Display Final Selection

The complete 7" Portable USB Touch Screen 16:9 Monitor by EVICIV, is the obvious selection. Though it is more expensive than any of the other options, its native support for HDMI, capacitive touch, a high-resolution display and its built-in speakers meet the entirety of the requirements for a display device, input device, and also provide room for optional audio feedback. The mounting brackets impose a clear design constraint in overall packaging, but their existence may yet prove to be more helpful than not.



Figure 5-M: 7" IPS Capacitive Touch Screen, Portable LCD

Image courtesy of: [EVICIV & Amazon](#)

5.3.4. Structural Components and Aesthetics

Though the focus is on successful implementation of the desired functionality, manufacturability, size, and the visual appeal of a practical enclosure for both distinct pieces of the total appliance are considered. The packaging of the Bartender and the Butler yields an opportunity for maximizing the allure of the system with ergonomic and aesthetic considerations in conjunction with practical ones. Thus, a rapidly deployable structural framework with room for all of the necessary electronic hardware, interfacing cabling, and the flexibility to include additional aesthetic features is ideal. To achieve this, the following components may be used:

- ¼" Medium Density Fibreboard
- ⅛" Hardwood Paneling
- 80/20 Aluminum T-slot & Fasteners
- Hardwood Stain & Sealer
- Vinyl/Acrylic/Matte Paint
- OLED/TFT Screen(s)
- Various LEDs
- Speaker(s)/Microphone(s)

5.3.5. Parts Selection Summary

This section is intended as an accessible, quickly informative reference. The component parts selected for each of the Bartender, Butler, and unifying Core Application are enumerated and a brief description of its technical purpose is provided. Details on precise sub-system functionality, cross-functionality interfacing, and other descriptive information are omitted.

Bartender:

- Microcontroller – ESP8266
- Dispensing Fluids – Peristaltic Pumps (Generic from Adafruit)
- Tubing – Generic Silicon (McMaster-Carr)
- Relay – MCIGICM
- Motor Driver – L293D
- Multiplexer – 74HC4051 8-Channel (Sparkfun)
- Docking – Hall Effect Sensor (Analog)

Butler:

- Robotic Platform – iRobot Create 2.0
- Navigation Firmware Host – Raspberry Pi 4.0
- Navigation Sensors – IR Photodiode Sensors
- Load Sensors – TAL221 Load Sensor with HX711 Amplifier Chip

Core Application:

- Operating System – Raspbian
- Programming language – Python
- Persistent Database – SQL
- IoT Core – MQTT

- GUI Library – PyQt5 (Python)
- Interface Display – USB Touchscreen by EVICIV

5.4. Phase I: Core Functionality

This section details the initial design architecture which is intended to provide a minimum proof of concept prototype and which satisfies the engineering requirement specifications (ERS) and design objectives. The core functionalities of the two mechanical subsystems can largely be developed independently, until system integration of their respective firmware is enabled by the unifying Core Application. Meeting the minimum ERS requires mutual and simultaneous development of each distinct system. To simplify development and facilitate successful integration, the overall Phase I system design is broken down by its component subsystems, Bartender, Butler, or Core Application.

For each subsystem: An architecture with a component block diagram meeting the minimum ERS defined by the overview block diagram, **Figure 5-A**, is proposed based upon the available component selections. A functional block diagram is given, where system-level inputs and outputs are labeled, and each internal process block is clearly identified. Each process identified in the functional block diagram is then rigorously broken down, describing sensor inputs, conditional outputs, data formats, and firmware functions. Finally, an integration strategy is specified with explicit regard to the IoT framework and I/O strategy therein.

5.4.1. Bartender: Simple Cocktails

This section details the Phase I architecture for the Bartender unit of the B3. The Bartender unit is responsible for key functions such as, confirming the presence and exact location and positioning of a valid drinking glass and, based on the beverage chosen by the user, dispensing the liquids from various bottles of available beverage ingredients into the awaiting drinking glass. This will be done precisely, accurately, and at a prompt rate, in order to achieve the specified time requirement.

5.4.1.1. Architecture

As with each sub-system in the design, the bartender unit plays a key role. The bartender is the entity that physically hosts the system that ultimately dispenses the beverage ordered by the user. This goal is achieved by the integration of carefully selected components.

First, the microcontroller to host the firmware was chosen to be the Feather HUZAH with ESP8266. Following this important selection, is that of the FDA approved peristaltic pump with silicon tubing and to round out the dispensing system itself, the following were also chosen based on desired characteristics and ease of integration with the testing board and each of the other components; the L293D Motor Driver Breakout, the open - source 74HC4051 8-channel MUX breakout, and 2-Channel DC 5V Relay Module with optocouplers. These

components integrating with the ESP8266 will be implemented, tested, and finely tuned as necessary to achieve the desired beverage within the time requirement with accuracy and precision.

Sensors are also implemented on the bartender in order to independently confirm that the butler docked accurately, as a supplementary check in regard to the butler's verification. Hall Effect sensors, more specifically the MPU-9250 Magnetometer Breakout module, was chosen to integrate with the ESP8266 microcontroller to achieve this feature.

Range finding sensors were researched for another supplementary check of the butler's verification, however this one is aimed to confirm, not only the presence of a valid drinking glass but the exact positioning of the glass as well. Information was gathered on two types of range finding sensors, infrared (IR) proximity sensors and ultrasonic (SONAR) proximity sensors. In this instance, one of each type, the LV-MaxSonar (MB1000) EZ0 range finder and open-source VL6180 infrared range finder breakout, were both chosen. Each sensor will be tested with the system and the results will be evaluated by the team before a final decision is made on which will be implemented with the ESP8266 microcontroller to achieve this feature.

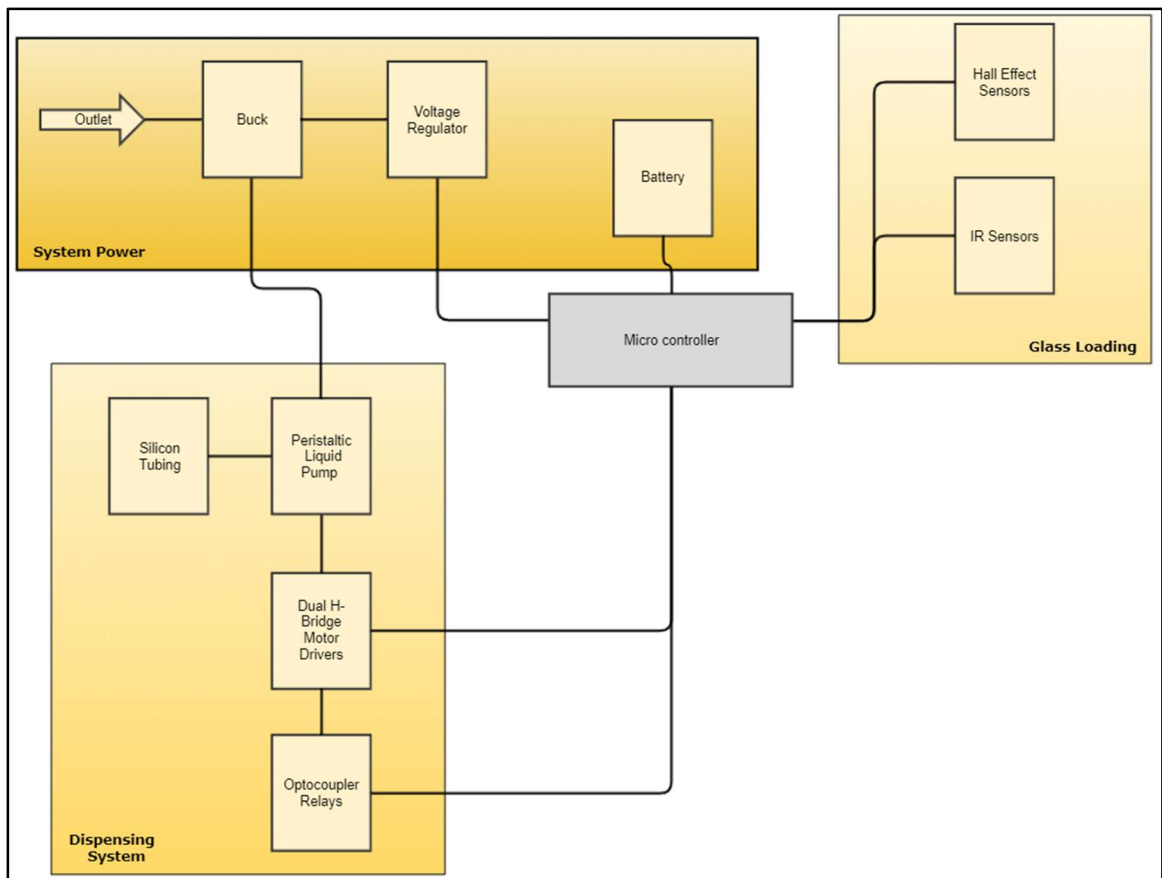


Figure 5-N: Bartender Components Block Diagram

Overall, the Bartender's appearance is based upon its need to facilitate the housing of the Butler and the drinking glass held in its cupholder when docked. The silicon tubing connected to each pump will be suspended from above into the center location designated to accurately dispense the liquids in the awaiting drinking glass. The Bartender's main structure will be large enough to house the components that physically make up its dispensing system, where the largest components to account for are the three pumps, and the overall size must be kept below the volume limit as specified in the design requirements.

5.4.1.2. Functional Block Diagram

The block diagram below is a representation of the functional process that the bartender unit aims to achieve. The process integrates with the microcontroller to transmit and receive data from each input and output, successfully and coherently. The following section breaks down the individual aspects of the process and explains how they integrate with each other cohesively.

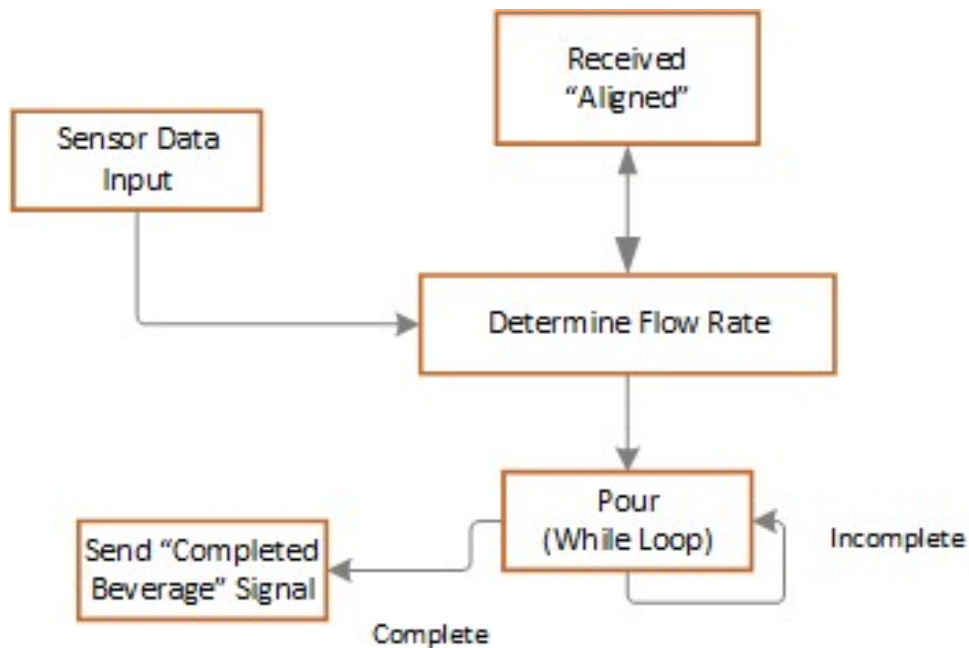


Figure 5-O: Bartender Functional Block Diagram

Dispensing

The goal of this process is to dispense the desired beverage with the correct ingredients and the appropriate volumes of each ingredient. **The first input** seen in the diagram of this process is the “aligned” signal. The “aligned” signal encapsulates the confirmation of the successful docking of the Butler unit, the verification of a valid drinking glass and its position in the exact location necessary for the following process to occur. This process is activated once the Bartender receives the recipe of the beverage chosen by the user and the notification that the verified drinking glass is “aligned”.

The dispensing of each ingredient continues as the condition of the function applies. **The three conditions of the function** are: (1) the “aligned” signal remains, (2) the “is dispensing” signal remains, and (3) the “overflow interrupt” signal is not present. The “aligned” signal will continue to remain as long as the Butler remains docked – as actively verified by the input data of the sensors, both the hall effect and those built into the iRobot. The “is dispensing” signal remains for the length of time determined by the flowrate and the volume of liquid needed per pump in use, based on the chosen beverage. The “overflow interrupt” signal will not occur unless a specified load limit is exceeded on the Butler. This signal is included to ensure that if a malfunction occurs the signal becomes an emergency abort flag which ceases all dispensing to avoid any unnecessary messes. The “Completed Beverage” signal is the **output** of this process.

5.4.1.3. Integration Strategy & I/O Summary

The Bartender receives a notification from the core application to expect the Butler’s arrival. Upon the Butler’s arrival, the Bartender confirms the accuracy of the docking process and verifies the validity of the drinking glass and its exact positioning; the Bartender then notifies the core of the success. Once the core receives all confirmations necessary, the recipe of the beverage chosen by the user is sent to the Bartender.

At this point, the dispensing process is activated. The dispensing process is actively monitored throughout its duration. Upon completion of this process, the notification is sent to the Butler to activate its process to deliver the finished product to the intended user/location.

MQTT Subscribe Tag List:

- Overflow_Interrupt_Flag
- Order_Recipe
- Emergency_Abort_Flag

MQTT Publish Tag List:

- Butler_Go_Flag
- Bartender_Docked_Flag

5.4.2. Butler: Line-Following

This section details the Phase I core functionality of the Butler unit of the B3. After successfully reaching this level of development in the Butler unit, the subsystem will be capable of using a load cell to accurately determine whether a valid drink container has been placed in its cupholder, and will be able to autonomously navigate a path between the user and the Bartender via a line-following system marked by a simple electrical tape path. The information read from the load sensing system and the completion of a navigated path will both be transmittable to the rest of the B3 through the MQTT core.

5.4.2.1. Architecture

The Butler unit's core design will be composed of the following major units: 3-5 IR Photodiode sensors, a simple load cell and associated amplifier circuit, the iRobot Create 2.0 unit, a Raspberry Pi module, and a custom PCB shield mounted to the Raspberry Pi for pinout connections to the other components and additional peripheral device support. The Load Cell selected for this prototype has four standard color-coded leads which will be soldered to the associated pins of the selected amplifier circuit. The output pins of this amplifier circuit will in turn be connected to the PCB shield of the Raspberry Pi.

Similarly, the IR sensors, which come already mounted to small PCBs, will be integrated through the PCB shield of the Raspberry Pi. Lastly the iRobot Create will have its serial ports connected to the available output pins of the Raspberry Pi's PCB shield to allow for control of the unit's motors from the Raspberry Pi module.

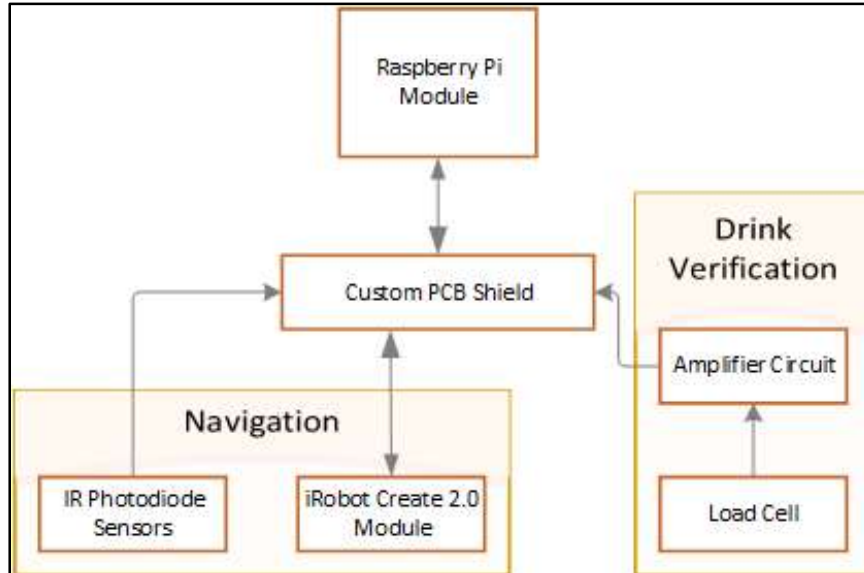


Figure 5-P: Butler Architecture Block Diagram.

Physically, the load cell will itself be mounted to a custom-made cupholder and calibrated for use as such, located at the top surface of the Butler's overall frame. The Base of this frame is mounted to the top of the iRobot Create 2.0, with the IR sensors mounted to the front of the iRobot in order to properly trace the path's line. The Raspberry Pi, its PCB shield, and the amplifier circuit of the load cell will all be contained within the Butler's frame, so as to keep the components covered throughout use.

5.4.2.2. Functional Block Diagram

The Butler subsystem's primary task is to navigate between the Bartender's and the User's locations. As seen below in **Figure 5-O**, the majority of the specific

processes defined in the Butler's functional block diagram are in fact incremental portions of the greater navigation system. Note that these "building blocks" of the navigation are in fact identical regardless of the direction the Butler is travelling. Because of this, the navigation processes are grouped within this section as Undocking/Pathfinding and Line-Following/Docking, following a reasonable chronological order.

Outside of the navigation itself, the Butler makes use of its load sensor to develop a failsafe preventing false starts to the drink preparation process or any chance of overpouring to the point of spilling. Unlike the navigation processes, these processes are specific to the Butler's current location. If the Butler is located at the user, it uses its load sensor to ensure a valid, empty drink container has been placed in its cupholder before accepting an order from the Core Application. If the Butler is located at the Bartender station, it constantly checks the load sensor for an overpour, and is prepared to shut down the Bartender as a failsafe if it believes this is about to occur.

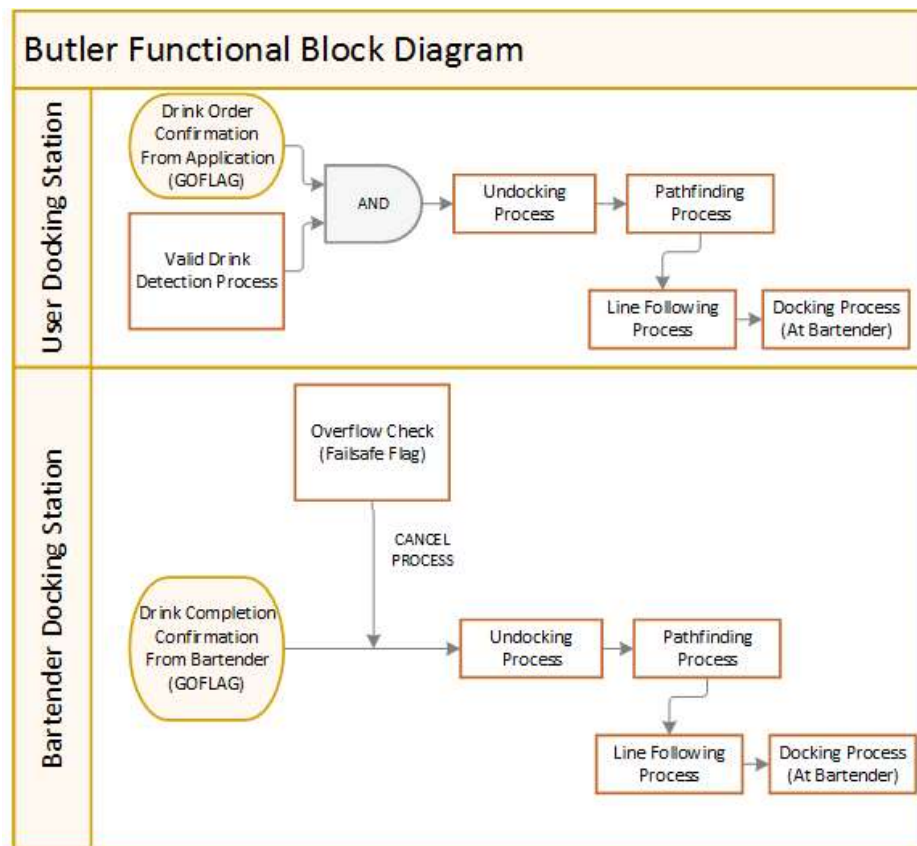


Figure 5-Q: Butler Functional Block Diagram.

Valid Drink Detection

The valid drink detection process determines whether the Butler's cupholder is presently holding a valid and empty drink container. This is used to determine

when to allow the Butler to begin its path to the Bartender, acting as a failsafe to avoid sending the Butler without a drink container, or with a partially filled container. This is crucial as either of these cases would likely result in serious spills which could damage the system and ruin the device's intended experience.

This process takes an input from the Load Cell through its amplification circuit to the Raspberry Pi, which has been calibrated to read this input as a legible and unit-quantified value. This value is then compared to a stored reference value with previously selected range of acceptable error. When the measured value falls within the acceptable range of passing value, the process will return an output signifying that a valid drink container has been placed. In all other situations, a failure will be returned as output continuously.

Drink Overflow Detection

Equally as important as the valid drink detection and providing a similar function is the drink overflow detection of the Butler's load cell. After the load cell has confirmed the object placed in its holder to be a valid empty container, the sensor will zero-out its reference upon reaching the Bartender. This serves to allow the Butler to track the overall weight of the fluid the Bartender pours into its cup alone, meaning if the value becomes greater than a certain threshold established in the Butler's Raspberry Pi unit, it will execute a failsafe operation commanding the Bartender to immediately cease pouring.

This is accomplished by having previously calibrated this threshold weight in the unit, and upon any instance of the measured weight exceeding that threshold, raising a flag which will be published to the MQTT core. Naturally, the Bartender is subscribed to this flag, and will respond by stopping the pour from all pumps immediately.

Undocking and Finding Line Path

When the Butler's Raspberry Pi receives published information from the MQTT core which shows the Butler_Go_Flag has been raised, it will begin its undocking and line-path locating procedure. Whether it is initially at the user docking station or Bartender docking station, it will begin the process of undocking and locating the line path laid out in front of its current station. This begins by the Raspberry Pi sending a command to the iRobot Create 2.0's serial input ports to command it to reverse off of the included charging station (docking port).

After undocking, the unit will again be commanded by the Raspberry Pi to slowly rotate clockwise, while the Raspberry Pi begins monitoring the data input of the IR photodiode sensors. Upon the IR sensors producing an input which corresponds to the line-path being centered between themselves, the undocking/pathfinding process will be considered complete, and the Raspberry Pi will begin its Line-Following process. This change between processes does not necessitate raising any flags or publishing information to MQTT, as it is merely a logical decision performed by the Raspberry Pi and is not relevant to the other subsystems.

Overall, this process requires both control over the motors of the iRobot Create 2.0 by the Raspberry Pi, and communication of the data from the IR sensors as an input to the Raspberry Pi as well. Although no output is produced by the process, its completion triggers the Line-Following process. The Line-Following process is activated only upon the completion of the Undocking and Pathfinding process, as observed by the Raspberry Pi.

Line-Following/Docking

The Line-Following process is activated only upon the completion of the Undocking and Pathfinding process (a change which will be controlled through the Raspberry Pi directly). Upon beginning the Line-Following process, the Raspberry Pi will make calculations based on the inputs received by the IR photodiode sensors in order to determine what path-corrections need to be made via control of the iRobot Create 2.0's motors. These corrections will be based on the deviation of the line from the center of the IR sensors in an attempt to keep the Butler aligned on the path while moving forward continuously.

The Raspberry Pi will command the motors to adjust in speed accordingly, and this process will continue successively until all of the IR photodiode sensors simultaneously register encountering a marked off path of tape at once. This will only occur at a marked "Tee" at the two ends of the line path and will signify to the Raspberry Pi that the Line-Following process has been completed. This Tee will be carefully placed at a distance from the docking station such that when the Butler reverses off the station at a later time, it will be centered physically over the Tee.

At this point, much like at the end of the Undocking/Pathfinding process, the Docking procedure will be signaled to begin by having the Raspberry Pi relinquish control of the iRobot Create 2.0's motors, and command the iRobot to begin its pre-programmed Docking command, in which the top-mounted IR sensor of the device is used to measure its orientation relative to the nearest of the two docking stations. Throughout this part of the process, the IR Photodiode sensors mounted to the front of the Butler will not be used.

It is worth noting that these last two described processes are part of a continuous cycle that forms the path the Butler will take back and forth between its two destinations. As seen below in **Figure 5-P**, as the Butler follows the line continuously in (A), it is purely under the command of the Raspberry Pi based on inputs from the IR Photodiodes, but in (B), upon reaching the Tee, it recognizes the end of its line-following process, and hands control off to the preprogrammed docking feature of the iRobot. Once docked, as in (C), it can be called upon by a flag-raise to reverse off the docking station, by a distance pre-determined to place it over the Tee, and begin rotating as in (D) until it is once again facing in parallel with the line it is centered on. At this point the Butler can once again begin its line-following procedure.

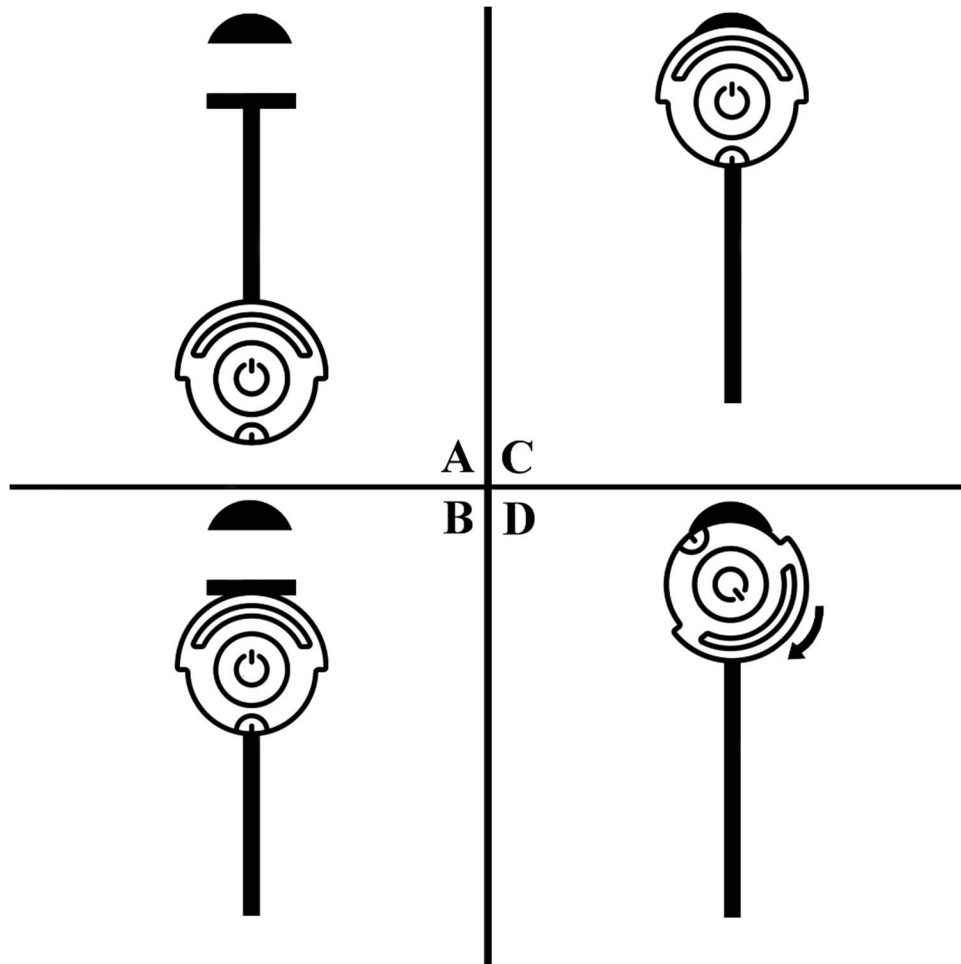


Figure 5-R: Butler Navigation

5.4.2.3. Integration Strategy & I/O Summary

The completed Butler system will, like the Bartender, primarily be integrated with the other subsystems through the use of input/output tags published through the MQTT core. The only true outputs produced by the Butler subsystem for use by the rest of the device is the flag which represents having completed the navigation and docking at either end of its defined path, so the rest of the system can respond accordingly and prepare for the next phase of the overall process, and the emergency failsafe flag in the case of overpouring from the Bartender. However, input signals from both the Bartender (in the case of the drink being completed) and the Application (in the case of the order having been completed) will be used as prerequisites for beginning the navigation processes, along with the Butler's own failsafe load sensing process. The flag raised by either one of the other two subsystems is in fact the same GO flag, but it is raised by different parameters depending on the Butler's location. Finally, a last-resort abort flag will be reserved for the Application such that in the case of any emergency the flag can be raised, causing the Butler to immediately stop its path and await a full reset.

MQTT Subscribe Tag List:

- Butler_Go_Flag
- Emergency_Abort_Flag

MQTT Publish Tag List:

- Butler_EmptyCup_Flag
- Butler_Returned_Flag
- Overflow_Interrupt_Flag

5.4.3. Core Application: System Integration

The purpose of this subsystem is to mediate between the Butler and the Bartender, as well as to interact with the user. The user will be required to interact with the Core Application pertaining to the configuration of the Bartender, drink order selection, and various notifications for cleaning, errors and restocking the Bartender.

5.4.3.1. Architecture

There are three main components needed for the Core Application to accomplish its goals. First, a workable and intuitive GUI is needed so the user can easily understand what inputs are required from them as well as what the outputs mean. Second, a persistent database that can store a table of all available drink recipes, and user input configuration parameters. Finally, the MQTT server needs to be complete and effectively able to communicate with both the Bartender and the Butler to ensure that critical data is being transmitted and applied.

The integration of these main components is the basis for how the entire system will run. Based on the inputs from the user, and configuration of available drinks will be filtered out from the database. This will then be used in the GUI to describe to the user what drinks are available based on the configuration they input. Following an actual order made by the user, the core application will then publish various flags to the MQTT broker to retrieve the recipe for the drink selected from the database and use the MQTT server to transmit this information to the Bartender.

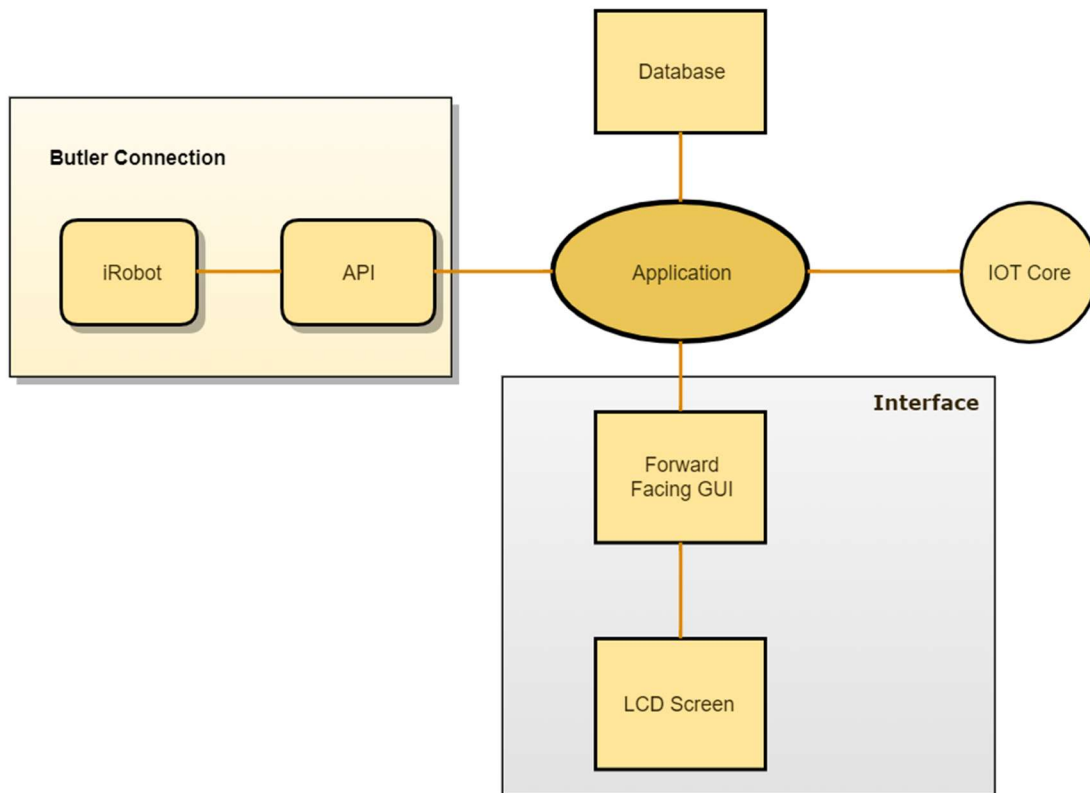


Figure 5-S: Core Application basic structure

The physical layout of what the core application will be directly interacting with is displayed in block diagram in **Figure 5-J**. The Raspberry Pi that will host the application will be directly attached to the Butler and use the power supply of the iRobot base to power it. Connected to this physically will be an LCD screen with touchscreen capabilities that will display the GUI to the user and receive input from them. The IoT core, database, and forward-facing GUI will all be run internally on the Raspberry Pi.

The GUI will have four primary pages, consisting of the main menu, a configuration page, an order confirmation page, and a loading page. The main menu page will have a few important components, the primary one consisting of a short list showing the available drink options. These drink options will be selected based on the user inputs for beverage ingredients. If the list of available drinks exceeds a certain length, the list on the main menu will have the ability to scroll down to see further options, as well as a side bar to indicate that this list is scrollable.

Selecting a drink from this list will bring the drink confirmation page to overlay the main page. Other available components include a hibernate button, which will send the touch screen into a page-less, low power mode, and a configuration button that will take the user to the configuration page.

The configuration page will allow the user to add or remove beverage ingredients from the internal list of “available ingredients” that the application uses to determine what recipes are available to be mixed. The actual layout of the configuration page will consist of a central list, similar to the beverage order list on the main page. This list will include all the ingredients the user had previously added and will allow the user to see an approximation of how much of that ingredient is left, as well as the option to remove an ingredient from the list.

Separate from the list will be a button that allows the user to add a new ingredient to the system. When the user selects this option, a secondary page will overlay the entire configuration page requesting the details of the new addition. The specific details that will be requested will be the name of the ingredient, the amount of liquid that will be added to the system, and the pump number that the user will be connecting it to. If the pump is already in use, an error notification will appear warning the user of the conflict and recommend removing that ingredient before adding a new one in its place. Once the information has been logged, a notification will appear indicating the user should now connect the new addition to the Bartender’s system, and briefly pause before allowing the notification to be dismissed to encourage immediate action.

The loading page will appear for the duration of the Butler’s travel to and from the Bartender. While the loading page is active, the user will be incapable of interacting with any components except for an emergency stop button. Pressing this button will halt all actions and have the Butler return to its default docking station.

5.4.3.2. Functional Block Diagram

Configuration

The persistent database will initially be filled with a series of tables that consist of several dozen recipes, collectively referred to as the “Blackbook”. The Blackbook will contain a large variety of ingredients. The user will need to configure the B3 with the ingredients that they have purchased independently. Using the GUI of the core application, they will need to input the ingredient they wish to add to the system. Once an ingredient is selected, the application will request the user to enter the quantity of the ingredient being added to the system, either in ounces or milliliters, and subsequently assign it to a dedicated pump. After the information has been entered, information will be stored in a table consisting of “available ingredients,” and the user must physically connect the ingredient to the Bartender’s chosen pump.

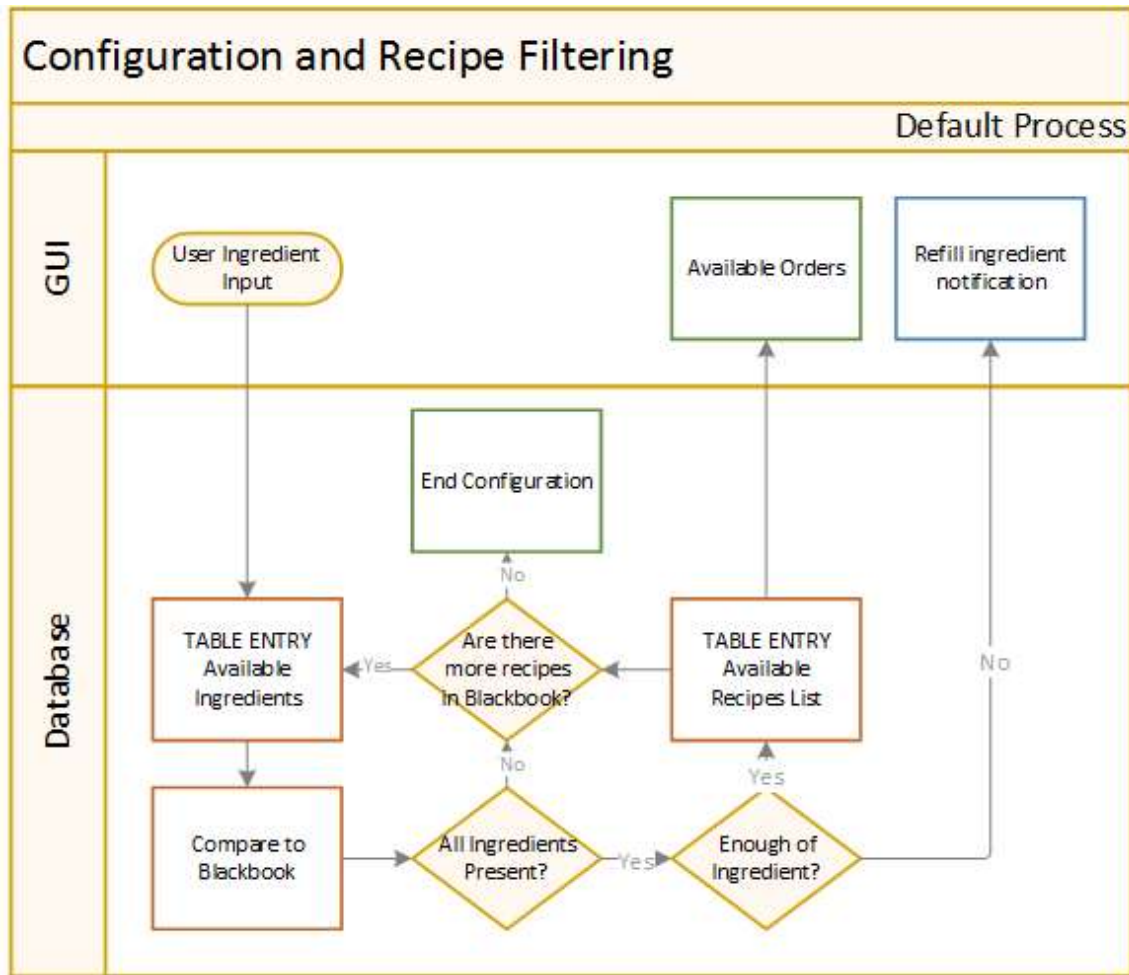


Figure 5-T: Configuration and Simple Drink Select Block Diagram

Select from Available Drinks

To accomplish this task, the SQL database containing all known recipes will have a related sub-table for each recipe consisting of a column of ingredients and a column of the quantity of the ingredients. After the “available ingredients” table has been updated, a sub-function of the configuration process will then compare the list of ingredients to all recipes in the Blackbook. Any recipe that has all of ingredients listed in the “available ingredients” table will then have its name selected to an active in-memory table for “available recipes.” This list of available recipes will be used by the GUI to give the user a list of options to order from. It is also worth noting that if a recipe requires an amount of an ingredient larger than that which is available, the recipe will not appear on the “available recipes” list.

GUI Modes

There need to be two primary modes of function for the GUI. When there has been a significant period without user input, or without completion/error flags from other subsystems, a hibernation mode will be activated in order to save power. During

this hibernation, the display will be turned off, and the GUI will go into a standby mode that will need to constantly await any touchscreen input to “wake” the device.

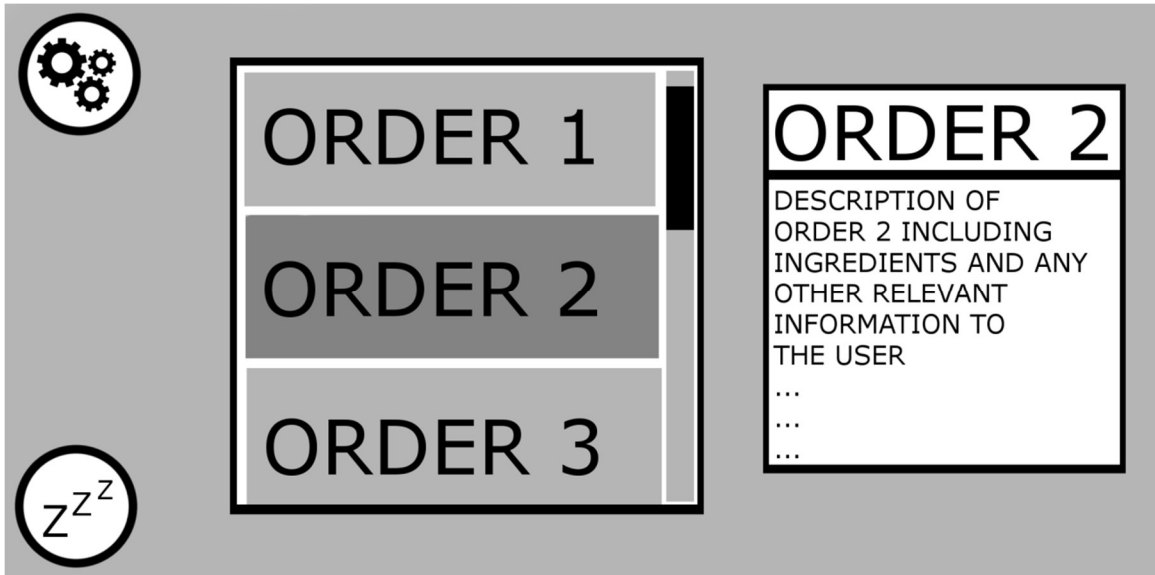


Figure 5-U: GUI mockup, initial screen

The other primary mode of function will be the active use state, in which the application has recently been updated, either by user input or from activity from either the Butler or the Bartender. Upon transitioning from the hibernation state to the active use state, the initial screen that will appear will contain a large menu that consists of the list of available drinks. Other, small items on this screen will include a “return to hibernation” button and a button for editing the current configuration of the Bartender (for adding or removing ingredients). Once a drink has been selected, a pop up will appear that will give a short description of the drink that was selected alongside a confirmation button and a cancellation button. The cancellation button will return the user to the drink selection menu, whereas the confirmation button will bring the user to a loading screen, shortly after which the Butler will begin its journey to the Bartender.

The loading screen will have an additional cancellation button (one that is distinctly different from the previously mentioned one). If at any time this second cancellation button is pressed, the core application will send an emergency signal. Upon receiving this signal, the Bartender will immediately stop any dispensing taking place and the Butler will turn around and return to its starting docking platform.

5.4.3.3. Integration Strategy & I/O Summary

Many of the inputs for the core application will come from the user, and there are only two inputs that it receives from the MQTT; however, those two that it does receive from the MQTT are crucial to the function of the B3 as a whole. The first input is a simple flag from the Bartender that is sent once the Butler arrives at the Bartender and is confirmed to be aligned. Upon receiving this flag, the core

application will then and only then send the recipe that the user ordered to the Bartender to start the dispensing process. The other input is sent by the Butler when it returns from the Bartender. Upon receiving this flag, the core application will create a notification (preferably with an audible notification) to alert the user that their beverage has arrived.

The total amount of outputs from the core application to the MQTT is also low in number, totaling at three. The first is a simple flag indicating that the user has made an order. Only the Butler will be subscribed to updates on this flag and is essentially telling the Butler that it should start making its way towards the Bartender. The second output is the recipe for the drink that the user has ordered. Only the Bartender will be subscribed to this information, and it will be sent once the core application has received the flag from the Butler saying it has arrived at the Bartender and is correctly aligned.

MQTT Subscribe Tag List:

- Bartender_Docked_Flag
- Butler_Returned_Flag
- Butler_EmptyCup_Flag
- Overflow_Interrupt_Flag

MQTT Publish Tag List:

- Butler_Go_Flag
- Emergency_Abort_Flag
- Order_Recipe (actual recipe, not a flag)

5.5. Phase II: Advanced Features

This section details a set of mostly mutually independent expansions of core functionalities at each subsystem of the B3. Each expansion of functionalities increases the overall capacity of the B3 to meet its design objectives, with incremental steps in the degree of sophistication. Once the Phase I design has been completed and validated, development will continue along the clearly defined roadmap outlined here. Each distinct Phase II sub-system design may yet require coordinated changes to the overall IoT framework and I/O integration strategy, but the roadmap has been laid out such that a fundamental redesign is unnecessary. The key focus is on generating an achievable target for implementation, given the functional basis from Phase I.

For each sub-system: The next phase of implementation is motivated and briefly summarized. The updated Phase II features, functionalities, or methods are then roughly outlined in what effectively amounts to a brief narrative addendum to the

Phase I functional block diagram. Finally, changes to the component or design architecture are discussed and conceptual groundwork for implementation is set.

5.5.1. Bartender: Scaling Up

A key commercial point for the SirMixABot is its scalability; it is clear that more simultaneously configurable ingredients yields a greater diversity in beverage options – and so capacity for maximizing user utility. Further, by design, it is technically straightforward to scale the Phase I implementation of the Bartender and its dispensing system. Hence, an obvious next step from the Phase I design of the Bartending station is to provide additional space for more beverage ingredients – and thus expand on the available beverage recipes.

Phase II Objective(s):

- Provide at least 5 configurable pumps.

5.5.1.1. Advanced Functional Description

The functional block diagram for the Butler differs in a few aspects from the . The Butler shall qualify and dispense a beverage order in the identical manner it did in Phase I – namely, after the system proceeds through readiness checks and the core application delivers the beverage “recipe”. However, it will now behoove the configurator of the device to include additional ingredient options – such as by explicitly listing them in the configuration menu and physically connecting them to the additional pumps. Further, a new variety of pumps could be introduced for specialized beverage ingredient options – such as viscous mixers like sour-mix or simple syrup.

5.5.1.2. Advanced Architecture

To support the additional beverage ingredient options more pumps would need to be integrated unto the Phase I platform. This could be achieved with additional multiplexers, relays, motor drivers. The power system may yet require an update to support the additional maximal load from the new pumps. A modular approach to firmware would facilitate this implementation, but that precondition is not strictly necessary – as the desire for scalability may yet warrant and motivate optimization of the firmware itself.

At the system level, additional IoT framework data tags would need to be generated to support the additional pumps – such as to receive drink orders. However, it is possible to reorganize the IoT framework to be modular in its initial implementation – such that support for scaling is technically intrinsic.

5.5.2. Butler: Path Recognition & Obstacle Avoidance

By the development of Phase II in the B3, the main goal for the Butler unit is to achieve navigation without a guided line to follow. This will begin with a bridging

process of having the unit first run through a course with a guided path, but commit the path to memory. This will not necessarily require any new components, but simply further development of the firmware with which the navigation is run.

An additional goal of the next stage of the Butler is to incorporate object avoidance into its path development, essentially updating its path based on obstacles encountered such as furniture. This will likely necessitate new sensors in order to detect these obstacles and develop a new path based on encountering them, unlike the added memory features.

5.5.3. Core Application: GUI Aesthetics

The Phase I core application is entirely functional in the scope of delivering a final product, which is successfully being able to receive an order, facilitate the process to complete the order, and deliver it to the user. Despite this, there is still plenty of room for improvement. As far as the core application is concerned, a more elaborate GUI would go a long way towards improving the user's experience with the B3

5.5.3.1. Advanced Functional Description

The primary advanced function that is desired for the core application is the ability to add new drink recipes to the pre-established Blackbook that is stored in the core application's database. In its current iteration, the Blackbook is a static set of recipes that cannot be given new parameters, changed, or expanded by the user. Allowing users to edit recipes to their exact taste, adding new recipes, or adding custom descriptions to recipes are features that are realistically achievable but are secondary objectives compared to basic functionality of needed for the successful proof of concept of the B3.

To accomplish these tasks, additional functions would be needed for within the GUI that allow the user to input string that would then have to be interpreted into the data and commands necessary to interact with the SQL database. Slight errors from the user could cause a great discrepancy, such as misspellings and incorrect formatting of data. This could have the potential to lead to the cup overflowing due to a large recipe, null values, and miscommunications through the MQTT core. To mitigate this, a series of "smart" detections would be needed to cover the many edge cases that a human user naturally causes, which could involve heavy coding.

6. Prototype Construction & Coding

This section contains information related to the production and manufacturing of the electronics that constitute the B3.

6.1. PCB Design and Assembly

After extensive research and development is completed, a functional prototype must be made. This prototype will be necessary to completely build a functional unit. One of the components for this working prototype will be a printed circuit board (PCB) designed by the group for specific use in the B3. Eagle is a well-known program will be used to design the PCB as well as various schematics.

Our group has never created a PCB before; there is much that is unknown about the process of creating a quality PCB. Fortunately, Eagle is a program that has the powerful ability to design a schematic and create a PCB based on that schematic. Though our group has heard of it, it has infrequently be used within our group, and as such there will need to be quite a bit of research for our group to be able to utilize this program effectively.

A total of two PCBs will need to be created for the B3. One will be used to shield the Raspberry Pi that will house the core application. The other PCB will integrate the microcontroller with the relay modules, H-bridge motors, and 3:8 multiplexers that will allows the Bartender to function as a whole system, rather than a series of connected parts.

When designing the actual PCB, there are various factors that need to be considered. One of the primary issues with PCBs is the size of the components, simply because a board will not work if all the parts needed for it do not fit. Another crucial factor is the power needed for the board as well as the heat that that power generates.

The various resistors contained in each of the parts generate a lot of the heat, and as a result routing traces will be needed. Though Eagle has an auto router to assist in the placement of these routers, it is usually recommended to route them manually and only use autotracers for precision and bottleneck detection. As this is standard, it is expected that it will be followed for this project's development.

7. Prototype Validation Plan

This section provides an in-depth, step-by-step description of the testing performed over the course of developing a functioning prototype of the B3. These tests are grouped by the three major subsystems of design used throughout this project; the Butler, Bartender, and Core Application. This format was selected in order to break the overall testing plan into several small tests, many of which can be performed in parallel as each subsystem is simultaneously developed. Upon successful testing of each distinct subsystem as described below, the completed project can be integrated as a whole and tested in simple trial-runs of what actual demonstrable use of the device has been defined as. That is, the final test will be a simple attempt at using the completed prototype.

Each subsystem's multiple tests will be defined so as to test each major component for functionality and compatibility with the rest of the design before attempting to integrate it with other components to form the subsystem. This lends itself such that each subsystem's testing plan can be followed as a chronological guide of assuring that no errors have been made throughout the construction of the B3. The tests will be laid out in such a way that if failure occurs, the reason behind it can easily be isolated to the most recent component being integrated or the process used to integrate it. In the event of failure for any one test, the standard protocol will be to search for the cause, attempt to correct it, and repeat the test.

Each test will be described in its respective section with a clearly defined objective listing any and all parameters being tested, as well as what constitutes as a minimum "passing" output, followed by a brief description of the testing environment including the necessary tools or elements to perform the test. A step-by-step narrative of the procedures performed will follow, then finally summary detailing the results of the test for those which have been performed. *For tests which have yet to be performed, there will be no results.*

7.1. Bartender Prototype Testing

This section details the overall testing environment of the Bartender unit. The testing is broken down into two general systems that make up the Bartender, first the testing process of the systems are described narratively with the following section being the overall criteria the systems aim to meet. The two systems of focus in the section for the Bartender are the dispensing system and the sensor system.

Overall Testing Environment

The Bartender testing environment will need to spatially resemble that of a standard home. This is required in order accurately determine if the overall size of the Bartender itself remains confined within the range specified in the design requirements. The final structure will need to be measured once physically built

and it effectively houses the necessary components the Bartender requires. This also needs to be done to determine the height in which the Bartender's placement needs to be in order to allow the Butler to dock and disengage successfully.

7.1.1. Dispensing System Testing

The central task of the Bartender is the administration of beverages that have been ordered. To accomplish this task successfully, the Bartender must be able to dispense fluids from the bottles of various liquors and mixers, quickly, accurately, and precisely into the awaiting drinking glass. Thus, to achieve this desired feature, this section describes the plan of testing to achieve this goal. With respect to the pump system, the following elements of the system will be tested: the microcontroller, the peristaltic pumps, silicon tubing, motor driver, the multiplexer, and the relays.

The main component for the bartender is a ESP8266 microcontroller. Due to its importance, we will be testing the GPIO pins of this component thoroughly by testing each functionality including: I2C, I2S, UART, PWM, and the 10-bit ADCs. This will be tested in combination with the multiplexer by plugging into the interface, integrating the specified libraries correlating to the functionality being tested, and verifying that we can receive coherent data from a component such as a sensor.

The second test, with respect to the pump system, is based on the functionality of the main component of the project, the ESP8266 microcontroller, with the testing process previously described. Once verified, a pass or fail test will be done on the optocoupler relays selected. This is not a test to see if it can be integrated with a certain board but is simply to verify its basic ON/OFF functionality.

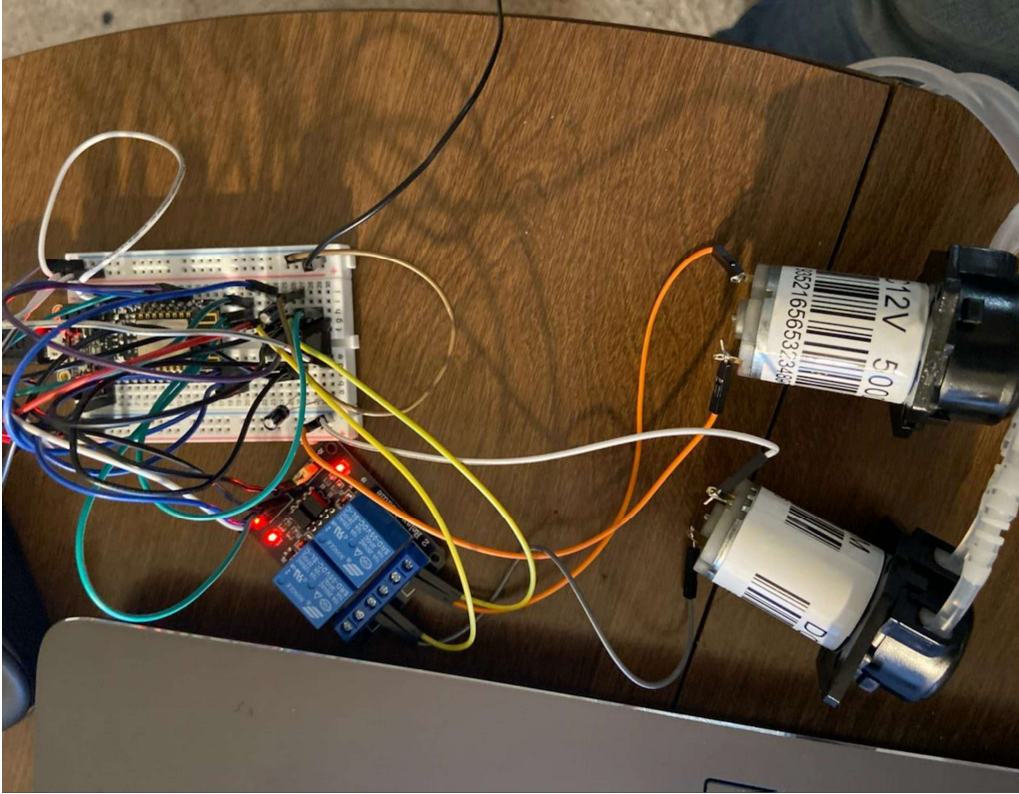


Figure 7-A: Confirming the basic ON/OFF function of both the microcontroller and the two channel relays, as indicated by the red LEDs.

Based on confirmed functionality of the previous tests, the testing process would then move on to utilizing the microcontroller and the peristaltic liquid pumps by integrating the dual h-bridge motor drivers, the silicon tubing, and the optocoupler relays and testing to ensure their individual functionality. Based on the selection of the Arduino compatible components, the testing code will be written in the chosen environment, Arduino IDE. The functionality of each component will be determined based on the flow rate. The flow rate testing will be achieved using the pulse width modulation feature of the motor, testing at 50%, 75%, and 100% power. From these results, the team can determine if the selected components are integrating with the chosen board as desired.

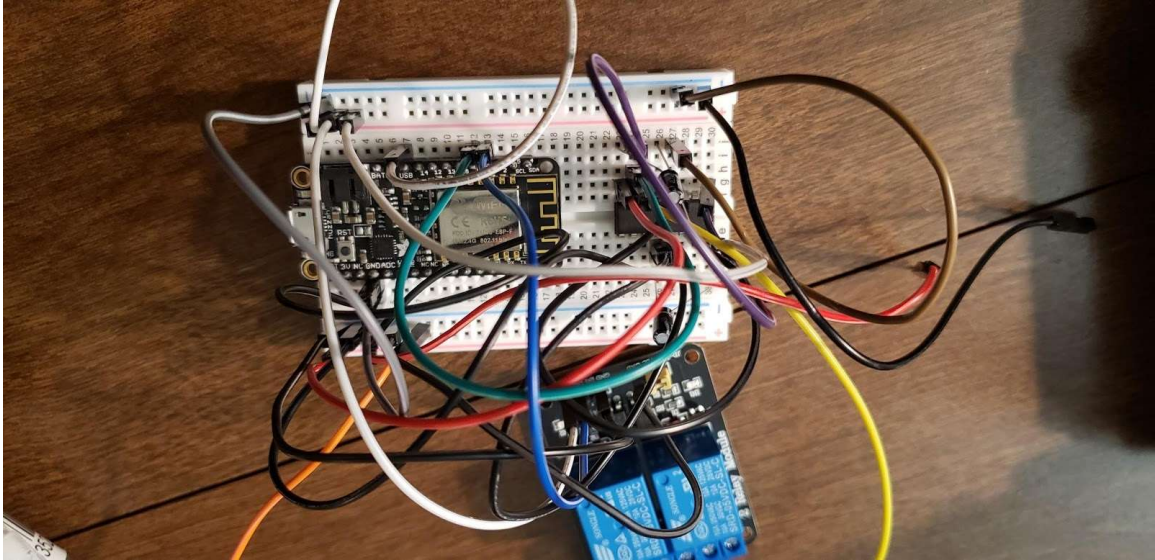


Figure 7-B: Initial circuit design for testing (1) peristaltic pump through dual h-bridge motor driver based on proper microcontroller setup and code.

The previous testing will be repeated with additional peristaltic pumps. First, the testing will be completed with one additional pump. With this addition, the team will modify the existing code in the Arduino IDE to account for the addition and upload the changes to the connected microcontroller. With these changes, the integration will be observed and altered until deemed successful based on previous results of the single pump integration. Upon successful integration of the second pump, the third pump will be added. Once again, the code will be changed, uploaded to the connected microcontroller, and the integration will be similarly observed, altered, and verified.

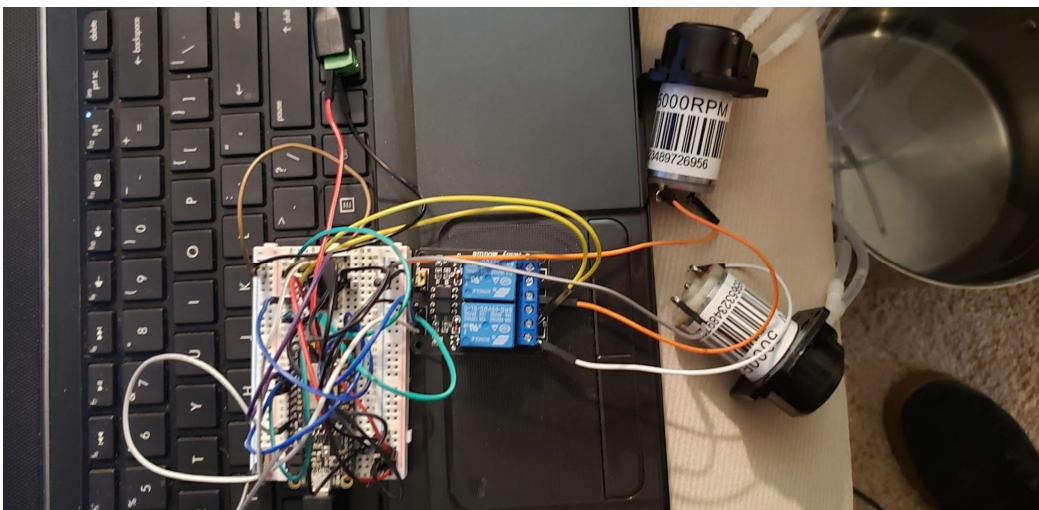


Figure 7-C: Initial circuit design for testing (2) peristaltic pump through dual h-bridge motor driver based on proper microcontroller setup and code. Code alternated between states of both being unpowered, each one individually being powered, and finally both being powered simultaneously.

Further testing of this system will include the use of multiple liquids of various viscosities to gauge and to determine the accuracy of the amount of dispensed liquid that is anticipated. Additionally, while completing these tests, we will be observing the tubing for any existing perforations, as well as ensuring no discoloration occurs. Resistors, capacitors, and other passive components will also be tested primarily by using a multimeter. We will be checking the values stated to ensure the correct part was received as well as for the safety of our main components.

Testing Criteria

The parameters being tested for their basic functionality using a pass or fail testing process are; the multiplexer, the GPIO pins of the ESP8266 microcontroller, the optocoupler relays, and the peristaltic pumps with the silicon tubing. Following the confirmation of basic functionality of the component, the integration is then tested to confirm the ability to send/receive coherent data with respect to the multiplexer, the GPIO pins of the ESP8266 microcontroller, and the optocoupler relays; and the flowrate accuracy using the pulse width modulation feature of the motor, testing at 50%, 75%, and 100% power, with respect to the pumps with its tubing.

The parameters overall integrations being tested are ultimately to determine the bartender's ability to receive a drink order and based on the recipe received, accurately dispensing the required volume of the correct liquids, and within the time specified time requirement.

Procedure

1. Plug the multiplexer into the interface of the ESP8266 microcontroller.
2. Integrating specified libraries correlating to functionalities that need to be tested.
3. Determine the ability of the GPIO pins of the ESP8266 microcontroller to receive coherent data by testing each functionality including: I2C, I2S, UART, PWM, and the 10-bit ADCs.
4. Wire the relay to the power source
5. Logic of relay determined by the GPIO pin logic (pass/fail)
6. Determine if the selected relays successfully display its basic ON/OFF functionality. (pass or fail test)
7. The testing code, based in the Arduino IDE environment, will be written to integrate the pump(s) with the microcontroller.
8. Determine individual functionality of the peristaltic liquid pumps (pass/fail)
9. Determine flowrate accuracy of a single pump using the pulse width modulation feature of the motor, testing at 50%, 75%, and 100% power.
10. Update the testing code to integrate an additional pump with the microcontroller.

11. Determine flowrate accuracy with two integrated pumps using the pulse width modulation feature of the motor, testing at 50%, 75%, and 100% power.
12. Update the testing code to integrate two additional pumps with the microcontroller.
13. Determine flowrate accuracy with all three integrated pumps using the pulse width modulation feature of the motor, testing at 50%, 75%, and 100% power.
14. Repeat step 13 with multiple liquids of various viscosities to gauge and to determine the accuracy of the amount of dispensed liquid that is anticipated.
15. Throughout testing of pumps, observing the tubing for any existing perforations and discoloration.
16. Determine accuracy of components such as resistors, capacitors, and other passive components used in the circuit design by multimeter.
17. Determine the bartender's ability to receive a drink order. (pass/fail)
18. Based on the recipe received, determine the bartender's ability to select the correct liquids.
19. Determine accuracy of dispensing the required volume.
20. Determine the accuracy of the overall time required to complete the dispensing of the beverage desired.

Results

From the functionality results of the main components of the system, based on flowrate accuracy, the team can determine if the selected components are integrating with the chosen board as desired. By this protocol, while testing the peristaltic pumps, we observed that the pumps were not dispensing at a desired rate. Taking into consideration the ratio of alcohol to mixer per drink, the existing pumps can be used to dispense the alcohol. However, for the intended mixers, a pump with a higher flow rate will be needed. Based on this finding, the team is looking into alternate pumps with a higher flow rate.

Upon receiving the data from further testing, the design team can decide if the components are meeting the specified requirements set in place for this design. If any component may fail to reach such expectations, the team will reevaluate and make an alternate selection for the failed component(s). The results and decisions made based upon those results will be found in this and the following results sections.

7.1.2. Proximity Sensor Testing

A task of the Bartender is the to sense the presence of the Butler Bot, independently of the Butler sensors. To accomplish this task successfully, the Bartender must be able to determine the positioning of the Butler Bot as it docks into its station which is integrated into the framework of the Bartender. The intended use of the sensors is to achieve a high accuracy of the Butler's placement. This is necessary to ensure that the awaiting glass is in the exact location prior to

the bartender beginning to dispense the liquids to avoid any spills or messes. Thus, to achieve this desired feature, this section describes the plan of testing to achieve this goal. The following elements of the system will be tested: Hall Effect sensors, IR proximity sensors, and ultrasonic SONAR sensors.

The first test with respect to the system that senses the Butler's presence is also based on the functionality of the main component of the project, the ESP8266 microcontroller, with the testing process previously described. Once the functionality of the microcontroller is verified, the sensors will be integrated and a pass or fail test will be done on the selected Hall effect sensors. To achieve cohesiveness of the system, the testing code will be written in the same environment, Arduino IDE, and uploaded to the development board. Performing this test will determine if we are getting an input from the sensors or not. This is not a test to see if it can be integrated with a certain board, but simply to verify its basic functionality.

Upon confirmation of the basic functionality of the Hall Effect sensors, the testing process would then move on to incorporate testing of the physical presence of the butler. The testing code will be adapted to integrate the sensor on the bartender as well as the components on the butler. Testing the integrated sensors will be done by detecting the magnetic field of the magnetic components placed on the butler bot itself. The test will determine if the Butler's presence is being "seen" by the sensors or not. The outcome of this test will determine if the Hall Effect sensors are successfully achieving our goal or if an alternate component will need to be researched acquired.

The following stage will proceed in a similar manner to that of the Hall Effect sensors. The first step in this testing stage will test and verify the basic functionality of the IR proximity sensors, by way integration with the development board and uploading the testing code. The following step will be to establish the connection between the sensors on the Bartender and the Butler. With a confirmed connection, the team will then evaluate the communication of data and make the necessary alterations until yielding a successful outcome that meets the specified requirements.

This next stage now repeats the process previously stated, with the selected Ultrasonic Sonar sensors. First, testing and verifying the basic functionality by integration with the development board and uploading the testing code. Then, the process of establishing the connection between the sensors on the Bartender and the Butler will be tested. With this confirmed connection, the team will then evaluate the communication of data and make the necessary alterations until yielding a successful outcome that meets the specified requirements. The testing of the Ultrasonic Sonar sensors will be done to compare with the IR proximity sensors. They are both possibilities to be used for the detection of the drinking glass relative to the location anticipated by the Bartender. While taking into consideration that the Butler itself will have sensors used to detect the presence

of a valid drinking glass, the options of the IR and sonar sensors are necessary to provide redundancy as a failsafe for the dispensing system of the Bartender, so as to not create any avoidable messes. Some elements of comparison will be accuracy, limitations, and the overall ease and success of integration with the rest of the unit. Based on the results of each, the team will determine which of the sensors will be selected to be implemented in the final design.

Testing Criteria

The parameters being tested for their basic functionality using a pass or fail testing process are; the hall effect sensors, the IR proximity sensors, and the ultrasonic proximity sensors. Following the confirmation of basic functionality of the component, the integration is then tested to confirm the ability to send/receive coherent data.

The parameters overall integrations being tested are ultimately to determine the bartender's ability to determine the positioning of the Butler Bot as it docks into its station, independently of the Butler sensors. Further testing, to determine a high-level accuracy of the Butler's placement of the drinking glass.

Upon receiving the data from this testing section, the design team can decide between the proximity sensors being tested for comparison as well as determine if the components are meeting the specified requirements set in place for this design. If any component may fail to reach such expectations, the team will reevaluate and make an alternate selection for the failed component(s). The results and decisions made based upon those results will be found in this and the following results sections.

Testing Equipment

These sensors are ultimately intended to be used for the detection of the Butler unit. At the time of these intermediate tests, the Butler is not fully functional or completely constructed. Due to this fact, for these tests to place the additional tools/resources were used to perform this test.

- Neodymium magnets

Procedure

1. Write testing code in Arduino IDE environment for hall effect sensor integration.
2. Uploaded the code to the development board to integrate the hall effect sensor
3. Determine the functionality of the hall effect sensor (pass or fail test)
4. Determine if bartender can receive coherent data from the hall effect sensor
5. Alter the testing code in Arduino IDE environment for IR proximity sensor integration.

6. Uploaded the code to the development board to integrate the IR proximity sensor
7. Determine the functionality of the IR proximity sensor (pass or fail test)
8. Determine if bartender can receive coherent data from the IR sensor
9. Alter the testing code in Arduino IDE environment for ultrasonic proximity sensor integration.
10. Uploaded the code to the development board to integrate the ultrasonic proximity sensor
11. Determine the functionality of the ultrasonic proximity sensor (pass or fail test)
12. Determine if bartender can receive coherent data from the ultrasonic proximity sensor
13. Use the results of the proximity sensors and decide between IR and ultrasonic proximity sensors.

7.2. Butler Prototype Testing

This section details the testing procedures relating to the Butler subsystem of the B3, including not only the process of the tests as planned, but all the necessary tools and associated components, the implications of each test on the system as a whole, and the environment in which to perform the tests. The overall navigation systems and load sensor setup are described throughout this section within this setup.

Overall Testing Environment

The Butler's navigation testing will require a testing environment which accurately reflects a standard home's smooth and hard floor, with the freedom to place black tape where needed. Because of this, any standard tile or hardwood floor can be used, and the tests themselves will likely be performed in available rooms or hallways of the UCF engineering buildings. Prior to beginning the navigation testing, the floor should be verified as being reasonably level.

As the project expands past its initial core functions, the navigation is expected to develop from a simple straight line to a path with turns and obstacles. These obstacles can take the form of chairs, tables, couches, and other furniture, so as to most accurately simulate a typical user's actual living room items. The curves formed by tape paths may need to be experimented with in order to achieve a setup which produces the best possible environment for smooth turns by the Butler.

The Butler's drink identification testing will have an "environment" defined by the structure surrounding the load sensor, which is the Butler's frame itself. Because of this, the Butler's valid drink identification testing's late stages will need to be performed after the frame has been completely constructed.

7.2.1. Valid Drink Identification Testing

The Butler's ability to determine if a valid drink has been placed in its cupholder is a failsafe before allowing the navigation procedures to begin and protects the system as a whole from false starts and potential damages as a result of no cup or an overfilled cup being placed in it. Therefore, the valid drink identification feature needs to be a consistently reliable system which can be depended upon by the rest of the Butler's functions.

After passing this validation milestone, the Butler system should be capable of consistently assessing whether a valid, empty drink container has been placed in its cupholder or not. Additionally, it should be capable of zero-ing out its reading and determining if an overflow is being approached. To accomplish this, the load cell, amplifier circuit, and pin connections to the Raspberry Pi will all need to be tested, and calibration will need to be accomplished through the Raspberry Pi module itself.

Testing Criteria

The efficacy of the valid drink detection system will be tested in steps – across the Raspberry Pi's ability to receive legible data from the Load Cell and its amplifier and its ability to transcribe this data into a quantifiable weight in units with accuracy great enough to discern an empty cup from one with 10mL of water. Finally, the Raspberry Pi will be set up to return a pass or fail based on the system created for determining if the object placed is an empty valid drink container, and the outcome of this standard will be tested for many varied outputs. After completing these three key points of testing, "passing" will be defined as receiving a legible pass/fail output from the Raspberry Pi that can successfully determine whether what is placed in the cupholder is in fact a valid drink for 10 successive trials, and the ability to consistently recognize an overpour for 10 consecutive trials.

Testing Equipment

Outside of the actual components being tested in this process, the additional equipment needed includes a single cup of the standard size and weight selected for the actual B3 to use, access to water to fill the cup, and a measuring cup for determining the exact amount of water being filled.

Procedure

1. Connect the Load Cell's color-coded leads to the corresponding pins of the amplifier circuit.
2. Use a breadboard to wire the output pins of the amplifier circuit to the selected input pins of the Raspberry Pi.
3. Setup the Raspberry Pi's program for processing input from the Load Cell.
4. Determine if the variable used to display the input value from the Load Cell changes significantly in the presence of weight on the cell (pass/fail).
5. Mount the Load Cell to the cupholder and zero out the Raspberry Pi's reading of the input. Use a test cup to develop a scale in measurable units.

6. Determine if the scale can accurately be used to discern an empty cup from a cup with 10mL of water for 10 trials (pass/fail).
7. Define a pass/fail output in the Raspberry Pi based on the input of the Load Cell.
8. Determine if the program can now successfully allow empty, valid cups to pass, and filled cups/invalid cups to fail for 10 trials (pass/fail).
9. Set up the Raspberry Pi's function for zeroing out the load cell measurement.
10. Zero out the measurement and then fill the cup to what has been selected as the maximum acceptable pour before failsafe-cancelling the Bartender.
11. Set this measurement as the maximum before the failsafe flag is raised.
12. Determine if the scale can accurately raise the flag after filling the cup past this amount (obviously stopping before an actual overflow) for 10 consecutive trials (pass/fail).

7.2.2. Navigation Testing: Line Following and Docking

This next set of tests covers the half of the Butler's navigation systems that allow it to follow a line path and recognize the end of its path by the Tee marker, as well as the docking procedure that occurs at this path end. This feature is extremely important, as it makes up half of the Butler's key defining function; the travel between user and Bartender. The components being tested in this section specifically include the IR photodiode sensors, the iRobot Create 2.0, and the Raspberry Pi module which manages them both.

Testing Criteria

The specific parameters being tested for line following in this section are the Butler's navigation system's ability to recognize input from the IR sensors, to successfully use that input to identify the presence of the tape path or lack thereof, and to successfully use that identification to travel a smooth, straight path, stopping at the recognized Tee that concludes the tape. For this portion of the test, "passing" is defined as being able to successfully travel a 10-meter straight line of tape and stop at the Tee that concludes it without spilling a cup that has been filled with as much water as a standard drink provided from the Bartender by volume.

The docking procedure is a preprogrammed function that can be performed by the iRobot Create 2.0, but nevertheless must be tested for both level of quality provided and ability of the Raspberry Pi to call this function on command. These two parameters will be considered at a "passing" level if the Raspberry Pi can successfully call the docking procedure for 10 consecutive trials without spilling a glass that is again filled to the volume of a standard drink from the Bartender. These trials will be performed from the same distance away from the docking station that the Tee marker is placed at.

Testing Equipment

Outside of the actual components being tested in this process, the additional equipment needed includes a single cup of the standard size and weight selected

for the actual B3 to use, access to water to fill the cup, and a measuring cup for determining the exact amount of water being filled.

Procedure

1. Wire the IR sensors to the selected input pins of the Raspberry Pi module
2. Set up the Raspberry Pi's program for recognizing the input of these sensors as quantifiable units.
3. Determine if the Raspberry Pi can be used to determine if a reflective surface is placed in front of the IR sensors (pass/fail).
4. Wire the serial connections of the iRobot Create 2.0 to the Raspberry Pi module.
5. Set up the Raspberry Pi's program for controlling the two motors of the iRobot Create 2.0.
6. Determine if the Raspberry Pi can be used to selectively control motors for forward, backward, and bi-rotational movement (pass/fail).
7. Apply the tape that will be used to test the simple line-following test to the testing environment.
8. Set up the PID control function on the Raspberry Pi that will be used to process the IR input into motor-control output.
9. Determine if the integrated unit can successfully travel a 10m taped distance with no additional weight (pass/fail).
10. Apply the additional weight of the frame, repeat (9) (pass/fail).
11. Apply a filled drink to the unit's cupholder, repeat (10), with success now including no spills (pass/fail).
12. Determine if the Raspberry Pi can successfully initiate the iRobot Create 2.0's homing feature to the docking station (pass/fail).
13. Determine if this process can be used to successfully dock from the taped Tee for 10 consecutive trials without spilling any liquid from a filled drink (pass/fail).

7.2.3. Navigation Testing: Undocking and Pathfinding

The second, equally important half of the Butler's navigation system is made up of the undocking and pathfinding features. Without these features, the previous sections could not even be performed, as the Butler would never be able to place itself on the line path. After completing this level of testing, the Butler system will be able to undock from its station from a command placed by the Raspberry Pi, by reversing off the dock onto the taped Tee. It will then be able to rotate without travelling while scanning the input of the IR sensors in order to locate the beginning of the taped path and orient itself to begin the line-following function. The specific components being tested here are again the IR sensors, iRobot unit, and the Raspberry Pi module.

Testing Criteria

The specific parameters being tested in the undocking procedure are the Raspberry Pi's ability to command the iRobot Create 2.0 to undock from the station, and its control thereafter in reversing the iRobot to be centered on the Tee of the taped path. For the Raspberry Pi's undocking control to have passed this test, the module will need to successfully command an undocking procedure for 10 consecutive trials. Similarly, for the control aspect to be considered passable, the unit will need to reverse successfully onto the taped Tee, without over- or under-shooting the travel, for 10 consecutive trials.

For the pathway-finding feature, the parameters being tested are the IR sensor's ability to notice the appearance of the tape line from rotation, and the Raspberry Pi's ability to control the rotation such that the device stops in a position from which it can begin to run the line-following procedures. For these features to be considered passable, the pathway-finding feature will need to successfully orient the Butler such that the line-following procedure can successfully begin running for 10 consecutive trials. As such, this testing must be completed following the completion of the line-following testing plan.

Testing Equipment

Outside of the actual components being tested in this process, the additional equipment needed includes a single cup of the standard size and weight selected for the actual B3 to use, access to water to fill the cup, and a measuring cup for determining the exact amount of water being filled.

Procedure

Note: Much of the setup for this procedure has already incidentally been performed in the previous section, and as such will not be repeated here.

1. Set up the program with which the Raspberry Pi can send an undock command to the iRobot unit.
2. Determine if the Raspberry Pi can successfully command the iRobot unit to release from its docking station (pass/fail).
3. Adjust the aforementioned program to have a set reverse travel distance calibrated to place the iRobot on the taped Tee.
4. Determine if the program can successfully command the iRobot to leave its docking station and stop on the taped Tee for 10 consecutive trials (pass/fail).
5. Set up the program in through which the Raspberry Pi commands the iRobot to rotate clockwise until the IR sensors receive input corresponding to the taped line being centered on the iRobot's face.
6. Determine if the above program can successfully orient the undocked module with great enough accuracy to successfully begin the line-following procedure immediately after for 10 consecutive trials (pass/fail).

7.3. Core Application Testing

This section details the expected test plans for validating the functionality of the core system application and its supporting backend services. Ensuring the functionality of the front-end application, especially the ability to receive user inputs for configuration and beverage ordering, along with the ability to provide direct feedback to the user via display or audio, requires an incremental test plan that tracks the development steps and verifies functionality at each step. After establishing the viability and stability of the operating environment, testing for the GUI shall track the development of its features and culminate in overall validation.

Overall Testing Environment

The environment in which the following tests will occur will be on the Raspberry Pi that will be used in the B3 prototype. More specifically, the environments will include the Python application and GUI, the MQTT server running in the background, and the SQL persistent database that will also be running in the background. All three of these environments will be run on the Raspberry Pi. As further tests are completed, these various components of the core application will be tested alongside each other to ensure that the various components are able to run simultaneously. It further serves as a way to test that the total resource requirement of the entire application is able to be met by the Raspberry Pi.

7.3.1. Stability of Operating System/Host Environment

As stated above, the various tests will need to be performed in the MQTT server, the SQL database, and the Python application. In order to do this, each environment will need to be stripped down to its bare essentials in order to remove excess demands on resources. After this, it first needs to be verified that these three stripped environments can be created on the Raspberry Pi. After this test has been passed, we will be able to be able to test the various capabilities of each of these components, as well as their functionality in tandem with one another.

Testing Criteria

There are various, basic questions that must be answered to determine if the environment is alive and stable. Given that the operating system is Rasbian, use various commands to determine the following questions. Is the system accessible by the root user? Do the core services supporting the host environment start without prompt and remain stable in operation? Are the audio/visual drivers installed, operational, and configured properly? Are input devices configured properly? Is the file system intact and persistent? Does the system have access to a package library of resources, to integrate new functionalities as needed? Is the software development environment installed and operational with the necessary packages, modules, libraries and services installed?

Procedure

1. Log into root user to ensure administrative system access

2. Check the is-active status of basic application. Services (return 0 is active)
3. File system error check command “*fsck*” on system reboot
4. Using *lspci* to find the driver locations for video and audio, using the */sys* command allows you to find their file location by searching top down
5. A full list of possible file locations for python libraries can be determined by inputting *\$python* into the command window
6. To test that the software IDE, database services, and MQTT services are installed and working, a simple startup and basic command is all that’s required to ensure each is working.

7.3.2. Database access and MQTT service access via Development Platform

Now that it has been established that these three components are able to function and coexist on the Raspberry Pi while remaining stable, the next test is to confirm that the development platform is able to interact with each of them successfully. Communication both to and from the SQL database and the MQTT broker need to be confirmed.

Testing Criteria

The SQL database needs to be able to be added to, edited, and read from in order to completely confirm that it is capable of being used in a meaningful way. The MQTT server is slightly more difficult to confirm; a testing application will need to be subscribed to information that it itself will publish. As a result, there is a need for a function that will constantly listen to the MQTT after both subscribing and publishing dummy test data.

Procedure

1. Start up the development environment for python coding. Import the relevant libraries needed for python to interact with MQTT services and to establish SQL database.
2. For SQL, import the SQL module and create a connection object to represent the database.
3. Create a dummy table with dummy variable of differing types
4. Set up a SQL cursor by calling its method from the connection object.
5. Use SQL command insert to insert some data into the database following the format used for the table created earlier.
6. Use various SQL commands to query the data that was input to ensure the insert and various querying commands are working as intended
7. For MQTT, first install the MQTT client by using *install paho-mqtt*
8. Import *paho.mqtt.client* as *mqtt*
9. Create a client instance for use in testing and a reliable broker address
10. Subscribe to a dummy topic with a QoS of 0. Print a marker statement.

11. Connect to the broker using the client ID and publish data to that dummy topic to test. Print a marker statement
12. Create a callback loop that will listen for the response from the broker due to newly published data on the dummy topic you subscribed to. Print the message topic, QoS, and retain flag to show all relevant data was transmitted.
13. Repeat steps 10 -12 with QoS 1 and QoS 2.

7.3.3. Basic Test of a Simple GUI

Following the establishment of a complete and operational development environment within the GUI application host machine, as validated in part by the tests performed in the previous sections, it is then necessary to execute an incremental testing procedure that tracks development efforts for the front-end interface. The need for this testing plan approach in derives from the multifaceted and abstract nature of the application itself: it carries many responsibilities and is sustained across various levels of abstraction. Hence, it is prudent to test the functionality in the order in which it is expected to be developed. It is noted that the test plan may not reflect the reality of development, so it must also be flexible.

Testing Criteria

There are five basic functions that directly relate to the implementation of any simple GUI. The primary window must be visible, it must best able to respond to user inputs, the data from those inputs should have the capability of being logged, custom widgets must be functional, the application must have some form of terminability and/or restartability.

Procedure

1. Construct a simplistic GUI consisting of some unique form of user input, such as a numbered slider, and a button that allows application termination. Allow for the number from the slider to be printed to a text document for recording.
2. If the GUI appears and is intractable, then the most basic functions of visibility and responsiveness are met.
3. Provide some input for the slider and terminate the application.

Check the application's log to confirm the data that was input was recorded properly. Given that the slider was able to move, and the program terminated correctly, we have shown a workable basic UI.

8. Administrative Content

This section will document the division of labor, planning and scheduling, and overall group coordination involved in this project as a whole.

8.1. Tasks and Responsibilities

This project has the requirement of being completed within two normal school semesters. Given the packed schedules of the team, there are many self-imposed deadlines for smaller tasks in order to keep on track with the development, design, and future construction of the prototype. Initial stages of prototyping are to begin prior to the spring semester. This was deemed necessary to give an excessive amount of time for the implementation of additional desired features. Furthermore, an aggressive timeline also acts as a safeguard against unexpected disasters, such as an accident involving the complete reconstruction of the prototype. In order to stay on top of such an aggressive timeline, each member volunteered to take executive responsibility over various modules and components of the B3. The exact distribution of responsibility is shown in the block diagram from **Figure 8-A** below:

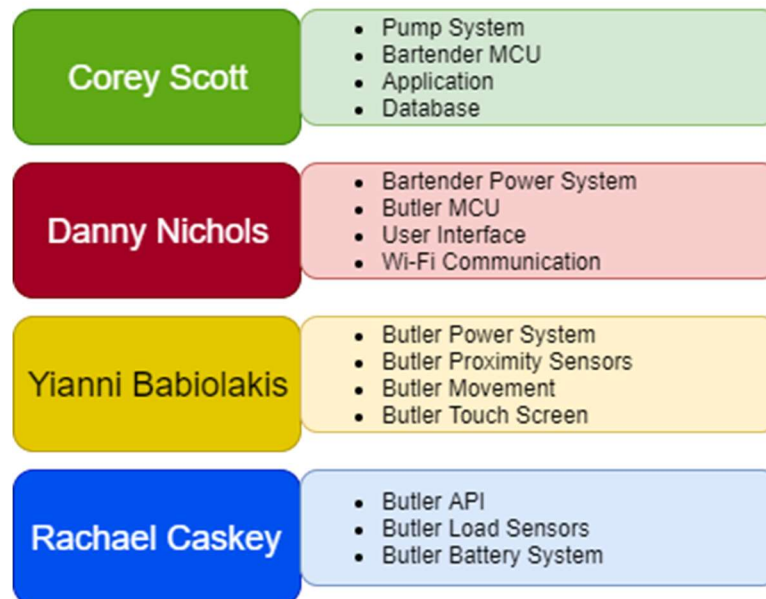


Figure 8-A: Responsibilities block diagram

8.2. Budget and Financing

The B3 is a project that is comprised of Electrical Engineering aspects as well as some Computer Engineering and Computer Science knowledge. Though the entire team is made up of Electrical Engineering majors, the entire team has experience coding and one has a minor in Computer Science. Given that software is something that is created independently, the cost of producing the software needed for this project is \$0. The majority of the expenditures will be the cost of the materials and components needed to construct the B3. Since there exists the possibility of commercializing the B3, the group chose to pay for the prototyping of the B3 out of pocket, rather than seeking funding. Due to this, it is crucial to determine an estimated cost and do as much as possible to remain within this range.

8.2.1. Estimated Budget

The estimated budget shown in Table 9.2 is based on an additional generous value of \$300 due to the anticipation of broken parts and failures due to the natural process of prototyping and changes in ideas or features. An additional table shown in Table 9.3 must be created in order to keep track of the total amount of true expenditures, as well as the total cost of the components that actually ended up being used in the B3, that is, the true cost of constructing the working prototype.

Item	Description	Estimated Cost
User interface host computer and touch screen display	A compact and low-end system for running the underlying application	\$100
Packaging / Aesthetics	Frames, fasteners, paint, veneer, bowties, and other out-facing aesthetics	\$150
Pumps, Valves, Actuators	For hygienically and precisely controlling liquid flow rates in the drink mixing station	6 x \$30
Sensors, Relays, Converters	For digitally monitoring and controlling the mechanical subsystems	\$100
Delivery bot base	To form the mechanical base of the autonomous bot	\$250
Development boards	For prototyping the integration of pumps, sensors, actuators, valves, etc. with the interface host	\$75

PCB manufacturing & shipping	For integrating the prototyping circuits into a final product	\$180 \$45
Final Documentation Printing/Binding, Showcase Props	For presentation and showcasing	~ \$25-65
Miscellaneous	For unexpected failures and minor to moderate pivoting	\$300
Total		~ \$1355

Table 8-B: Estimated Budget

8.2.2. Current Expenditures

Even without the \$300 set aside for mishaps and unexpected changes, we are still remaining almost \$100 under budget current. The additional room in the budget could potentially allow us to pursue implementing other features; however, such ideas should be put aside until a working prototype is put together.

Item	Description	Actual Cost
(6)Silicone Tubing for peristaltic liquid pump	A compact and low-end system for running the underlying application	\$21.00
(3) Peristaltic pumps	12V DC Power peristaltic liquid pumps. Comes with a length of silicone tubing attached	\$74.85
(2) Dual H-Bridge Motor Driver	H-Bridge motor driver that works with DC or steppers. Used for up to 600 mA	\$5.90
(1) Spy cam for Raspberry Pi	Small camera for capturing visual input data. Meant for use on a Raspberry Pi	\$39.95
(1) Raspberry Pi Zero WH	Raspberry Pi zero with headers	\$14.00
(1) Raspberry Pi Zero camera cable	Cable used to connect a Raspberry Pi Spy cam to a Raspberry Pi Zero	\$5.95
(1) Break-away .1" 2x20pin strip Dual Male Header	Soldered through microcontroller pins to allow for use on a breadboard	\$0.95
(1) Mini HDMI to HDMI Cable	Mini HDMI to HDMI Cable	\$5.95

(1) Raspberry Pi Zero	Raspberry Pi Zero... what'd you expect	\$10.00
(1) Adafruit Perma- Proto breadboard	It's a breadboard!	\$0.00
Delivery Charge	2 day shipping	\$10.53
(1) (0 - 1M) Ohm Resistor Kit	Resistors for circuit building	\$10.86
(3) SanDisk 16GB microSD card	16GB microSD card- 98MB/s, Full hD, C10, U1	\$11.58
(1) Assorted Schottky diode kit	200 pieces, 14 values, fast switching	\$7.99
(1) 7" USB monitor Raspberry Pi Touchscreen	Touchscreen IPS display computer monitor. 1024x600, 16:9, HDMI	\$79.99
(2) Lithium Battery pack	Lithium ion polymer batteries. 3.7V, 1200mAh	\$12.87
(2) Jumper Wire 120pc. Assorted Kit	Breadboard jumper cable kit. Comes with male-male, male-female, and female-female	\$7.49
(1) Solderless 400 pin breadboard -6 pack	Total of six 400 pin breadboards designed for raspberry pi and arduino projects	\$9.99
(1) Solderless 830 pin breadboard -3 pack	Total of three 830 pin breadboards designed for raspberry pi and arduino projects	\$9.21
(1) solder tip cleaning wire	Cleaning wire for soldering	\$7.95
(1) Lead free solder tip thinner, 20g	0.8 oz can of tip thinner for soldering	\$6.59
(1) Basic Soldering Kit	60W adjustable temperature soldering kit with ON/OFF switch	\$23.79
(1) AC/DC switching power supply	AC/DC power supply, input 100-240V, output 12V DC, 3A	\$12.99
(2) Multiplexer Breakout	Multiplexer breakout with 8 channels	\$5.95
(1) 500 pc. Capacitor Assortment Box	Electrolytic capacitor assortment ranging from 0.1uF to 1000uF	\$14.99

(2) Relay Module, 2 channel	5V DC, 2 channel relay module for Arduino UNO	\$7.99
(1) Raspberry Pi 4	Raspberry Pi 4 QuadCore 64 bit w/ Wi-Fi (4GB)	\$62.00
(1) Multiaxis Gyroscope, 3 pc. pack	Contains 3 axis and 6 axis accelerometer gyroscope modules	\$8.99
(1) Wire stripping tool	8 inch wire stripping tool	\$13.69
(1) Ultra thin Solder wire	0.3mm solder wire. Rosin Core Flux 2.5% lead free	\$9.99
(1) Rosin Paste Flux	Rosin Paste Flux, 2oz jar	\$16.45
(1) Multiaxis Gyroscope, 8 pc. pack	Contains 3 axis accelerometer gyroscope modules	\$19.90
(1) Wi-Fi transceiver pack	4 piece pack of 1MB serial Wi-Fi transceivers	\$13.98
(1) Adjustable voltage regulator 2 pack	2 adjustable voltage regulators. 4-38V to 1.25-36V step down regulators	\$11.95
(2) Arduino Feather Huzzah microcontroller	Adafruit Feather Huzzah with ESP8266 Wi-Fi	\$19.00
Various coupons	Save some pennies	-\$5.20
Taxes	As certain as death	\$10.19
(1) iRobot Create	iRobot Create programmable robot	\$199.99
Shipping	1 day shipping	\$26.50
Taxes	As certain as death	\$15.86
Total		\$938.80

Table 8-C: Current financial expenditures

8.3. Project Milestones

Monitoring the progress of the design and implementation of the B3 is crucial given the time constraints of the individual members of the team. This milestone table shown below will assist in ensuring that the project is completed in a timely manner over the fall and spring semesters. Individual group members can use this to gauge

their progress in independent assignments as well as set dates for important group meetings. Many of the important milestones include research, project documentation for Senior Design 1, acquisition of components and materials, various construction phases, testing, preparing for the final presentation.

As far as research and documentation, individuals will take charge of properly gathering information and filing in portions of the documentation based on the sections each individual selected. Given that various checkpoints for the Senior Design 1 documentation, completing certain tasks prior to the documentation checkpoint is critical to having enough information and data to complete the task.

8.3.1. Design Phase

This section serves to outline the estimated and idealized project milestones for the design phase of the project. This closely correlates with the Senior Design I term and concludes in December 2019. The items listed deal primarily with the associated documentation for the project, as well as the formation of a design and testing plan.

Week #	Date	Action
3	9/10	Formally initiate project brainstorming
	9/20	Set primary project objectives, scope
	9/20	Cement budget constraint
4	9/20	Initial D&C Document Due
	9/24	Meet with Professor(s) for Initial Feedback
	9/26	Re-evaluate primary objectives and project scope, pivot if necessary
5	9/27	Compile an initial list of all possible components, applicable or desirable technologies in budget
	10/4	Configure project management tracking system for development timeline: GitHub, Asana, etc.
	10/4	Cement primary project objectives, scope
6	10/4	Updated D&C Document Due
	10/8	Identify key prototyping components, tools, supplies
7	10/11	Assess and distribute research load
8	10/18	Acquire/Order key prototyping components
	10/22	Outline critical design report
9	10/25	Standards Assignment Due

10	11/1	60 Page Draft Due
	11/8	Validate control of valves, pumps, and key mechanical elements via development boards
	11/8	Validate relays, power systems, and control schemes via development boards
	11/8	Draft the presentation
11	11/8	Complete functional drinking mixing station breadboard prototype
12	11/15	100 Page Submission Due
	11/23	Formalize BoM, budget
13	11/23	Finalize critical design report
14	11/29	Finalize presentation
15	12/2	Final SD1 Document Due
	12/13	CAD structural components and packaging frames
16	12/13	Identify and acquire additional key components
17	12/20	Complete autonomous drink delivery breadboard prototype functionality

Figure 8-D:Senior Design I timeline

8.3.2. Implementation Phase

This section serves to outline the estimated and idealized project milestones for the implementation phase of the project. This closely correlates with the Senior Design II term and aims to conclude before the final presentation of the course. The items listed deal primarily with the actual construction, testing, and implementation of the prototype, and are meant to add a frame of reference with regards to time to the designs described above.

Week #	Date	Action
1	1/10	<i>Order/machine structural and aesthetic components</i>
	1/10	<i>Integrate functional subsystems via host application</i>
	1/10	<i>Overall system schematic</i>
2	1/17	<i>Mock-Up application and GUI</i>
	1/24	<i>Validate core application functionality</i>
	1/24	<i>Order/machine structural and aesthetic components</i>

3	1/24	<i>Mock-Up project website</i>
4	1/31	<i>Complete initial draft PCB layout</i>
	2/7	<i>Assemble structural framework, enclosures, and packaging</i>
5	2/7	<i>Optimize application GUI</i>
	2/14	<i>Assess possibility of additional scope</i>
	2/14	<i>Finalize PCB design and order it</i>
6	2/14	<i>Achieve overall project prototype functionality</i>
7	2/21	<i>Trial, Adjust or Accommodate</i>
8	2/28	<i>Finalize project and continue testing</i>
	3/6	<i>Continue testing</i>
10	3/13	<i>Integrate consolidated PCB(s) into appliance</i>
11	3/20	<i>Prepare Final Document</i>
	TBD	<i>Final Document Due</i>
	1 Week Prior Final Pres.	<i>Review/Revise/Practice Presentation</i>
	TBD	<i>Final Presentation</i>

*** Preliminary and subject to change.

Figure 8-E:Senior Design timeline

9. Appendices

This minimal section contains references and additional resources that may be required or convenient to the user.

9.1. Appendix A - References

[1] “SirMixABot Robotic Bartender - The Automated Bar For Your Home.” SirMixABot, 4 Jan. 2018, <https://www.sirmixabot.com/>.

[2] “Automated Bartender.” Hackaday.io, <https://hackaday.io/project/12260-automated-bartender>.

[3] “Party Robotics Blog.” Party Robotics Store, <https://partyrobotics.com/blogs/blog>.

[4] Poole, Ian. “IEEE 802.11b.” Electronics Notes, Radio-Electronics.com, 2006, www.electronics-notes.com/articles/connectivity/wifi-ieee-802-11/802-11b.php.

[5] Poole, Ian. “IEEE 802.11g Wi-Fi.” Electronics Notes, Radio-Electronics.com, 2006, www.electronics-notes.com/articles/connectivity/wifi-ieee-802-11/802-11g.php.

[6] Poole, Ian. “IEEE 802.11n Standard.” Electronics Notes, Radio-Electronics.com, 2006, www.electronics-notes.com/articles/connectivity/wifi-ieee-802-11/802-11b.php.

[7] van Heesch, Dimitri. “Quality of Service.” Paho MQTT C Client Library, 13 Sept. 2018, 13:40:20, www.eclipse.org/paho/files/mqttdoc/MQTTClient/html/qos.html.