**Senior Design II**

**EEL4915**

**University of Central Florida**

**Department of Electrical and Computer Engineering**

# **SMART Garden**

# Senior Design 2 Report

**Group 34**

**Ryan Hassan** **[CPE]**

**Kevin Lorch** **[CPE]**

**Lauren Melancon** **[EE]**

**Jonathan Wallhauser** [**CPE]**



Table of Contents

[Table of Contents 2](#_Toc133295182)

[1. Executive Summary 9](#_Toc133295183)

[2. Project Description 10](#_Toc133295184)

[2.1 Motivation 10](#_Toc133295185)

[2.1.1 Existing Projects and Products 10](#_Toc133295186)

[2.2 Objectives 14](#_Toc133295187)

[2.3 Engineering Requirements and Specifications 15](#_Toc133295188)

[2.3.1 Project Goals 15](#_Toc133295189)

[2.3.2 Project Requirement Specifications 16](#_Toc133295190)

[2.4 House of Quality 17](#_Toc133295191)

[2.5 The Garden 20](#_Toc133295192)

[3. Research and Decisions 22](#_Toc133295193)

[3.1 Relevant Technologies 22](#_Toc133295194)

[3.1.1 Moisture Sensor 22](#_Toc133295195)

[3.1.2 Ultraviolet Light Sensor 24](#_Toc133295196)

[3.1.3 Humidity Sensor 24](#_Toc133295197)

[3.1.4 Temperature Sensor 25](#_Toc133295198)

[3.1.5 Displays 26](#_Toc133295199)

[3.1.6 Number Pad 27](#_Toc133295200)

[3.1.7 Solar Panels 28](#_Toc133295201)

[3.1.8 Batteries 29](#_Toc133295202)

[3.1.9 Waterproofing 29](#_Toc133295203)

[3.1.10 Water and Fertilizer Pump Mechanism 31](#_Toc133295204)

[3.1.11 Water and Fertilizer Reservoirs 32](#_Toc133295205)

[3.1.12 Relay Module 33](#_Toc133295206)

[3.1.13 Wi-Fi and Bluetooth Modules 34](#_Toc133295207)

[3.2 Microcontrollers and Microprocessors 36](#_Toc133295208)

[3.2.1 Development Boards with on-board Microcontrollers 37](#_Toc133295209)

[3.2.2 Development Boards with on-board Microprocessors 38](#_Toc133295210)

[3.2.3 Development Board Comparisons 41](#_Toc133295211)

[3.2.4 Moisture Sensor Module 43](#_Toc133295212)

[3.2.5 UV light Sensor Module 44](#_Toc133295213)

[3.2.6 Humidity Sensor Module 44](#_Toc133295214)

[3.2.7 Temperature Sensor Module 45](#_Toc133295215)

[3.2.8 On-Unit Display 45](#_Toc133295216)

[3.2.9 Reservoir Pump 46](#_Toc133295217)

[3.2.10 Reservoir 46](#_Toc133295218)

[3.2.11 Multi-Channel Relay Module 46](#_Toc133295219)

[3.2.12 Wi-Fi and Bluetooth Module 47](#_Toc133295220)

[3.3 General Positioning 48](#_Toc133295221)

[3.4 Possible Designs and Related Diagrams 48](#_Toc133295222)

[3.4.1 Sensor Positioning 51](#_Toc133295223)

[3.4.2 Strategic OLED Display Positioning 53](#_Toc133295224)

[3.4.3 Water and Fertilizer Pumps Positioning 53](#_Toc133295225)

[3.5 Power Module 54](#_Toc133295226)

[3.5.1 Non-rechargeable Batteries 54](#_Toc133295227)

[3.5.2 Rechargeable Batteries 54](#_Toc133295228)

[3.5.3 Solar Panels 55](#_Toc133295229)

[3.5.4 AC to DC Outlet Power 55](#_Toc133295230)

[The Decision 55](#_Toc133295231)

[3.6 Communication Modules 56](#_Toc133295232)

[3.6.1 Range 56](#_Toc133295233)

[3.6.2 Power Consumption 56](#_Toc133295234)

[3.6.3 Security 57](#_Toc133295235)

[The Decision 57](#_Toc133295236)

[3.7 Printed Circuit Board 59](#_Toc133295237)

[3.7.1 Creating Printed Circuit Board Basics and Terminology 59](#_Toc133295238)

[3.7.2 Composition of a Printed Circuit Board 60](#_Toc133295239)

[3.8 Fertilizer Remedies 61](#_Toc133295240)

[3.9 Parts Selection Overview 63](#_Toc133295241)

[4. Related Standards and Realistic Design Constraints 64](#_Toc133295242)

[4.1 Related Standards 64](#_Toc133295243)

[4.2 Team’s Coding Standards 64](#_Toc133295244)

[4.2.1 Coding Language requirements 65](#_Toc133295245)

[4.3 Realistic Design Constraints 66](#_Toc133295246)

[4.3.1 Economic Constraints 66](#_Toc133295247)

[4.3.2 Time Constraints 67](#_Toc133295248)

[4.3.3 Environmental Constraints 68](#_Toc133295249)

[4.3.4 Social and Political Constraints 68](#_Toc133295250)

[4.3.5 Ethical Constraints 69](#_Toc133295251)

[4.3.6 Health and Safety Constraints 69](#_Toc133295252)

[4.3.7 Manufacturability and Sustainability constraints 69](#_Toc133295253)

[4.3.8 Problem Solving 71](#_Toc133295254)

[4.4 Arduino Communication Protocols 72](#_Toc133295255)

[4.4.1 Serial Peripheral Interface (SPI) 73](#_Toc133295256)

[4.4.2 Inter-Integrated Circuit (I2C) - 2 Wire System 74](#_Toc133295257)

[4.4.3 Universal Asynchronous Receiver-Transmitter (UART) 76](#_Toc133295258)

[4.4.4 Decision on the Communication Protocol 77](#_Toc133295259)

[5 Hardware Design 78](#_Toc133295260)

[5.1 PCB Design 78](#_Toc133295261)

[5.1.1 Software Used 78](#_Toc133295262)

[5.1.2 Schematic 80](#_Toc133295263)

[5.1.3 Layout 84](#_Toc133295264)

[5.1.4 Manufacturing 87](#_Toc133295265)

[5.2 PCB Manufacturing Vendors 87](#_Toc133295266)

[6 Software Design 89](#_Toc133295267)

[6.1 Arduino IDE 89](#_Toc133295268)

[6.1.1 Libraries 90](#_Toc133295269)

[6.1.2 Bootloader 91](#_Toc133295270)

[6.2 Software Communication integration 92](#_Toc133295271)

[6.3 Website development 94](#_Toc133295272)

[7 Hardware and Prototype Testing 95](#_Toc133295273)

[7.1 Hardware Testing Environment 95](#_Toc133295274)

[7.1.1 Sensor Testing 95](#_Toc133295275)

[7.1.2 OLED Display Testing 100](#_Toc133295276)

[7.1.3 ATMega328p Testing 101](#_Toc133295277)

[7.1.4 Power Supply Testing 102](#_Toc133295278)

[7.1.5 Relay Module Testing 103](#_Toc133295280)

[7.1.6 Water Pump Testing 105](#_Toc133295281)

[8 Software Testing Environment 106](#_Toc133295282)

[8.1 Software Development Collaboration 106](#_Toc133295283)

[8.2 Specific Service Testing 108](#_Toc133295284)

[8.2.1 Wi-Fi Module Testing 108](#_Toc133295285)

[8.2.2 Database Testing 109](#_Toc133295286)

[8.2.3 Website Testing 110](#_Toc133295287)

[9 Project Operation 112](#_Toc133295288)

[9.1 General Information 112](#_Toc133295289)

[9.2 Using the Device 112](#_Toc133295290)

[10 Administrative Content 113](#_Toc133295291)

[10.1 Milestone Discussion 114](#_Toc133295292)

[10.1.1 Senior Design 1 Milestones 116](#_Toc133295293)

[10.1.2 Senior Design 2 Milestones 117](#_Toc133295294)

[10.2 Final Budget and Finance 118](#_Toc133295295)

[10.3 Bill of Materials (BOM) 120](#_Toc133295296)

[10.4 Project Roles 122](#_Toc133295297)

[10.5 Content Distribution 124](#_Toc133295299)

[10.5.1 Organization 125](#_Toc133295300)

[10.5.2 Virtual Communication 127](#_Toc133295301)

[11 Components Not Used in Final Product 129](#_Toc133295302)

[11.1 Batteries and Solar Panels 129](#_Toc133295303)

[11.2 UV Sensor 129](#_Toc133295304)

[11.3 OLED Display 129](#_Toc133295305)

[12 Appendix 131](#_Toc133295306)

[12.1 References 131](#_Toc133295307)

**Table of Figures**

[Figure 1: House of Quality 18](#_Toc121515632)

[Figure 2 Basic Solar Panel Outline 26](#_Toc121515633)

[Figure 3: Hardware Block Diagram 34](#_Toc121515634)

[Figure 4: Angled view of SMART Garden 47](#_Toc121515635)

[Figure 5 :Top-down view of SMART Garden 48](#_Toc121515636)

[Figure 6: PCB Schematic 79](#_Toc121515637)

[Figure 7: ATMega328p Pin Mapping 81](#_Toc121515638)

[Figure 8: Fusion 360 Initial PCB Design 82](#_Toc121515639)

[Figure 10: PCB Development Board 95](#_Toc121515643)

[Figure 11: Activity Diagram 89](#_Toc121515640)

[Figure 12: Soil Moisture Sensor Testing Circuit 92](#_Toc121515641)

[Figure 13: Humidity and Temperature Sensor Testing Circuit 93](#_Toc121515642)

[Figure 14: Arduino Connected to OLED Display 96](#_Toc121515644)

[Figure 15: OLED Display Online Simulation 97](#_Toc121515645)

[Figure 16: ATMega328p Testing Procedure Set-up 98](#_Toc121515646)

[Figure 17: Multimeter Volatge Regulator Circuit Testing 99](#_Toc121515647)

[Figure 18: Relay Module Testing Circuit 101](#_Toc121515648)

[Figure 19: Water Pump Testing Circuit 103](#_Toc121515650)

[Figure 20: Organization of Discord text channels 124](#_Toc121515651)

[Figure 21: Example of the 'topic overlap prevention' channel 125](#_Toc121515652)

[Figure 22: Virtual Communication - Voice Channel 126](#_Toc121515653)

# **Table of Tables**

[Table 1: Project Requirement Specifications 16](#_Toc121515654)

[Table 2: House of Quality Key 17](#_Toc121515655)

[Table 3: Microprocessor Comparisons 39](#_Toc121515656)

[Table 4: Microprocessor Comparison - Pins, RAM and Temperature 40](#_Toc121515657)

[Table 5: Moisture Sensor Module Comparison 41](#_Toc121515658)

[Table 6: UV Light Sensor Module Comparison 42](#_Toc121515659)

[Table 7: Humidity Sensor Module 42](#_Toc121515660)

[Table 8: Temperature Sensor Module 43](#_Toc121515661)

[Table 9: On-Unit Display 43](#_Toc121515662)

[Table 10: Reservoir Pump 44](#_Toc121515663)

[Table 11:Reservoir 44](#_Toc121515664)

[Table 12: Relay Module 45](#_Toc121515665)

[Table 13: Wi-Fi and Bluetooth Module 45](#_Toc121515666)

[Table 14: Bluetooth and Wi-Fi Module 55](#_Toc121515667)

[Table 15: Part overview 60](#_Toc121515668)

[Table 16: Problem Mitigation and Solution 69](#_Toc121515669)

[Table 17: Problem Mitigation and Solution Continued 70](#_Toc121515670)

[Table 18 The Data Frame 74](#_Toc121515671)

[Table 19: Soil Moisture Content Representation 92](#_Toc121515672)

[Table 20: Software Test Parameters 108](#_Toc121515673)

[Table 21: Senior Design 1 Milestones 114](#_Toc121515674)

[Table 22: Senior Design 1 Milestones continued 115](#_Toc121515675)

[Table 23: Senior Design 2 Milestones 116](#_Toc121515676)

[Table 24: Final Budget 117](#_Toc121515677)

[Table 25: Final Budget Continued 118](#_Toc121515678)

[Table 26: Bill of Materials 119](#_Toc121515679)

1. Executive Summary

Food price inflation has already become an issue in 2022 and it is only projected to get worse over the next year. In fact, the food-at-home CPI, which measures the consumer price index for grocery store or supermarket food purchases has increased 0.6 percent from August to September of this year and is 13 percent higher than it was September of last year. That isn't even the worst of it, in 2022 all food prices are predicted to see an increase of upwards of 10.5 percent, with food-at-home prices increasing 11 to 12 percent and food-away-from-home prices increasing 7 to 8 percent. This issue isn't just local to the United States, supply chain issues caused by climatic events and political conflict have caused economic issues all over the world. Food insecurity has already begun to become a serious issue and we are only at the start. Problems like these are some of the large-scale issues we hope to alleviate with our cost-friendly SMART Garden.

Our vision is to create an automated plant watering system that uses soil moisture, ultraviolet light, temperature, and humidity sensors to monitor herbs and vegetables in an at home garden. The goal of this data collection would be to notify the user when corrective measures in regard to sunlight exposure or watering frequency for any given plant are needed via a mobile app or web application. An online user interface will also allow for the implementation of user-controlled water scheduling capabilities and open up the possibility for the user to be able to access large plant databases or view live status updates on their personal garden. The previously mentioned peripherals matched with the UI for the end user will provide a great user experience which optimizes the way the general public grows and harvests plants. This system will allow the user more time on a daily basis to be spent doing higher priority needs, while still allowing them to enjoy the many benefits of growing their own plants. This device will give users a cost friendly gardening option in hopes of increasing access to healthy food alternatives.

This document serves as a tool to report the SMART Garden design process. To start off, we will discuss the motivations behind why we chose this project and provide a comparison of competing products. Once this is complete, we will go over the objectives, project goals, and requirement specifications. The next section of the paper will be an overview and explanation of our House of Quality matrix followed shortly after by a review and analysis of all relevant technologies as they relate to the project's hardware set-up. After part selection and comparison, we will discuss decisions related to the software and communication systems. Directly after we will go over our groups initial PCB design and schematic and lightly touch on future manufacturing plans. Wrapping up the paper we will have an in-depth discussion on project operations as well as finalize all administrative content.

2. Project Description

The following sections will go over the motivation, objectives, and requirement specifications tied to this project. This section will display our House of Quality matrix coupled with an in-depth analysis of our choices. Lastly, we will discuss the five following competing products: Smart Garden Controller, SmartLeaf, Mushroom Nursery, EasyHerb, and Arduino Garden. The purpose of which is to cultivate a deeper conceptual understanding of our own project material in order to optimize the SMART Garden project design.

2.1 Motivation

As inflation in the United States continues to rise, and global supply chain issues persist, working class people are finding it harder and harder to be able to find or afford healthy food options at their local grocery stores. In fact, according to Morning Consult's latest U.S. Supply Chains & Inflation report, over half (51%) of consumers surveyed in March 2022 reported experiencing product shortages of specific types of groceries and food, up from 43% in September 2021. With that being said, the looming fear of large-scale food scarcity coupled with the steadily declining economy have forced many Americans to turn to unhealthy food alternatives. The smart garden presents a small but impactful way to help counteract this issue and benefit the lives of those in need.

Our goal in creating this product doesn’t stop at financially benefiting the lives of the people who use it. The SMART Garden serves as a way to better the health of the individual, both physically and mentally. The automation that this system offers was made and implemented to give time back to the user in a world where almost every social media platform or product is trying to do the opposite. Utilizing this tool will help the user develop a potentially important skill such as gardening while simultaneously helping them avoid the unhealthy grocery store alternatives to food that can be easily grown at home. Not only that, but gardening has been proven to have a positive effect on mental health, mood, and even brain chemistry. In fact, in a 2015 study published in the journal Ecopsychology researchers showed that “gardening provided a space of one’s own... increased feelings of connectedness, and improved physical and mental health”. The SMART Garden is tool that will have a positive and real impact on those who use it consistently.

2.1.1 Existing Projects and Products

**Smart Garden Controller**

This group from Summer 2018 seems to have a very similar approach to our own. Both groups decided to measure the conditions the plant is in and adjust them accordingly as much as is possible. While some of the statistics they record could be a little excessive, such as measuring the wind speed and humidity, their dedication to documenting the conditions must be admired. One large difference is that this group is measuring and watering a plant already in the ground, while our SMART Garden is a self-contained device that doesn’t require a viable plot of land. Their aims are to “take away the manual labor involved in watering plants and automate the watering maintenance,” which is rather similar to our goal of simply making it more possible for the average person to be able to grow plants for themselves.

One interesting aspect of the project that I appreciated is that their sensors seemed to be on a tool that is inserted into the ground, resembling one of the tags that identify plants in the store. Whether this is coincidence or thoughtfulness, I enjoyed it.

Despite our slightly different approaches, our projects are similar in more ways than they differ. Both measure soil moisture and water the plant accordingly, and both provide presets to choose from to optimize the care of specific varieties of plants. We both integrated an application to monitor the device from afar and change the care schedule remotely. Battery life is a strong consideration for both groups, since convenience is a large part of the motivation.

**SmartLeaf**

SmartLeaf is an indoor greenhouse project from Summer 2019 that is more focused on controlling the conditions of the plant’s environment than our SMART Garden. This project is a rather large unit that has devices to maintain the needs of the plants inside it. Instead of responding to the changing weather and reacting accordingly, they took the weather into their own hands, and the required hardware and setup is proportionally more intensive. The decision to make it a separate unit does move it away from an undertaking of architecture and closer to the hands of an average consumer, thus technically making gardening accessible to more people. They made their project with the intent of people using this large unit to grow plants, which is definitely a viable option, but our SMART Garden aims to make gardening not just easier, but more accessible. This large unit is more costly than ours, which is designed to be easily placed on a desktop.

Though it appears extremely different from the start, it is actually quite similar once you get into the comparisons. Both devices aim to measure the conditions of plants and water them, accordingly, have remotely configurable watering schedules and conditions, and are individual units that don’t require existing gardening-viable plots of land. Their dedication to making a device that can grow plants in any environment due to having its own self-contained environmental control and plant maintenance devices must be applauded.

**Mushroom Nursery**

This project is just about as far from our project as can be while still being similar enough to compare. Our SMART Garden aims to grow plants to help the average person garden in a world where homes with viable gardening space aren’t easy to come by, and people are too busy to dedicate time to gardening. The mushroom nursery on the other hand, simply wants to let people grow specific mushroom types that may not be local to your area or are too expensive to buy from the store. However, the possibility of the mushrooms being used for medicinal purposes is brought up, which can be argued would help the average person, but I’m not sure that counts enough for the goals of our projects to be considered similar.

What is definite however, is the fact that this is a small, self-contained unit that aims to cultivate the growth of fungus via careful monitoring of the conditions the fungus is in. The project is meant to be as hands off as possible during the growth stage of the mushrooms, with the user having very little to do to maintain the conditions, with the primary responsibility being to ensure the humidifier’s water tank is filled, since mushrooms thrive in environments that have enough humidity.

Unlike our own project, this mushroom nursery regulates the light cycle in the enclosure, claiming that this light cycle can help foster fruitful fungus. Additionally, the circulation involved in this project was not a consideration in ours. CO2 is denser than oxygen and will settle inside the nursery, but the fans built into the enclosure guarantee that the airflow will be sufficient to leave the fungi breathing clean.

The Mushroom Nursery is a very creative idea. While many groups wanted to grow plants, this group wanted to grow fungus, which requires a bit of a different setup than other garden–based projects.

**EasyHerb**

This is another irrigation project not unlike our own, but this one is designed as a hydroponic growing system instead of just a way to maintain conditions for a plant in a more standard growing environment. This is a stark contrast to our own project, which will sense the conditions of a plant in some dirt contained in our device. Though both projects measure the conditions of the plants’ environments, the measurements taken are different between groups. This group tracked the ambient temperature and humidity, light intensity, water temperature, water pH value, a nutrient level sensor, and a water level sensor. Our planned design only tracks soil moisture, ultraviolet light, temperature, and humidity to keep our plants healthy.

In addition to water and nutrients, this group also provides light to the plants so they can undergo photosynthesis even when the sun is not out and tied the power state of the lights to whether or not there is light available from another source to save power and bulb life.

The herbs are drip-watered, and thus always have a significant supply of water, as is traditional of hydroponic systems. Our design will sense when there is not enough water and actively release water to correct that value to one above the minimum soil moisture content. One of the most obvious differences between the two devices is the speaker system this group included. The speakers can play music while the system is on, which our SMART Garden cannot do, nor is that added as a feature in the foreseeable future. This would be a bad idea for our project, however, since we would like to keep the batteries being drained as little as possible and powering a speaker of any significant volume or quality would likely draw far more power than we are comfortable with. EasyHerb on the other hand, is powered via wall outlet, so they can ‘waste’ as much power as they want.

Another difference is that this project only senses the conditions the plants are in once per hour by default. The user can request a measurement, but it never checks more unless instructed. Our project aims to keep optimal conditions for as large a percentage of time as is feasible.

**Arduino Garden**

There are a great number of automated gardening projects available to be referenced online through various websites such as YouTube and Indestructables Living. They use a variety of processing units ranging from Arduinos to MSP’s to microPC’s. One that is notable is the project by the group Practical Engineering on YouTube. Their automated garden utilizes many sensors including: a moisture sensor, sunlight sensor, humidity sensor, and temperature sensor.

The Arduino checks the moisture level of the soil daily. Should the moisture reading be below the designated level, the system will activate the watering procedure to increase moisture to the desired level. The housing for the system is a small wooden construct, basically a box with ventilation holes.

Examples such as this are greatly beneficial for our project as the creators share the specifications of their technical equipment as well as their experience in creating the project. They offer useful suggestions as to how to approach aspects of the project. Examples of such include using a photoresistor and voltage divider in tandem to achieve a sunlight sensor. By calculating the sensor’s resistance, it can be determined if the sensor is in a high light or low lights situation and the system can respond accordingly. A large amount of resistance indicates low sunlight levels. Conversely, a small amount of resistance indicates high sunlight levels.

One of the main lessons to be learned from their design is to not overcomplicate the design in trying to make it fancy. Simple, clever solutions are ideal such as the one involving the voltage divider. Also, it is advised that we be very careful when attaching a water source to the garden as a failure in implementation may cause the garden to flood.

2.2 Objectives

The primary objective of this design project is to design and assemble an automated gardening assistant that can be utilized in a manual or automatic fashion. This system should be able to apply for gardens of varying levels of sophistication. It should be scalable to operate from a very small indoor garden setup to a larger more robust setup. Meaning that users can use a single SMART garden for a very simplistic garden setup, or they can use multiple SMART gardens in conjunction for a more robust setup. Our project is separated into multiple different subgroups that work in unison. As such each group is designated its own specific objectives whose collective mechanisms produce the desired results. Listing these objectives individually is necessary to clarify the requirements of this project’s design.

**Hardware**

The hardware for this project will consist of a microprocessor, which will host a Wi-Fi module, power system, moisture sensors, UV sensor, moisture sensor, humidity sensor, containers for the plant food and water, and a laptop or mobile device. It will also include an LCD display and number pad. This setup will allow the user to see the sensor reading and adjust the system settings from the SMART gardens system itself, using the number pad and LCD display, or remotely using a computer or mobile device. Finally, it includes the container of the plant itself.

**Software**

The software for this design will include many different subcomponents used to operate the controls, sensors and data management. The software will collect data and transfer it from the microcontroller to the computer or mobile device. This will allow the user to set up the automated water/feeding as well as trigger those processes remotely when desired. This includes the IDE used to program the Arduino Uno we will be using to control the SMART garden. This also includes the Backend and Frontend coding which is comprised heavily of Java Script code. This will also include Redux code that will be utilized in the mobile application.

**Control**

The microprocessor will act as the intermediary between the smart garden system and the user’s device, whether it be a computer or potential mobile device. The microcontroller will relay information to the computer which will do all the calculations and processing and return instructions for the microcontroller to enact on the system itself.

**Power Supply**

The main goal of the power supply of this project is intended to be as self-sufficient as possible. The power system will consist of a battery-based power source. Ideally, in optimal sunlight, the solar panel will power the system completely. The solar panel will also recharge the rechargeable battery of the system if it has been discharged. In the absence of sunlight, the battery will discharge to keep the system operating until it can be recharged.

2.3 Engineering Requirements and Specifications

2.3.1 Project Goals

* The system shall be able to maintain accurate moisture levels in soil for the plant that is being maintained.
* Levels for moisture should have the ability to be fetched from plant specification database.
* Levels for humidity should have the ability to be fetched from plant specification database.
* Levels for temperature should have the ability to be fetched from plant specification database.
* Levels for sunlight should have the ability to be fetched from plant specification database.
* The system shall allow for two feeders, i.e., one for water and one for nutrients to the system.
* The system shall be able to fit in most common household locations such as the kitchen.
* The system shall be portable.
* The system shall be able to read data from an engineer-created Smart Garden database and apply the information to the overall system.
* The system shall be able to charge its battery via AC outlet or through the on-board sunlight PV panels.
* The rechargeable battery shall hold a minimum charge of 12 hours to get through a single cycle.
* The system shall have a portability factor.
* The system shall be water resistant.
* The system shall implement an integrated PCB.
* The system shall implement an interactive LCD with peripherals for navigating certain options within the system user interface.

2.3.2 Project Requirement Specifications

Below are the project requirement specifications for the Smart Garden. This table is dynamic in the sense that more project requirement specifications may be added to this as the engineer's progress through their prototype. The size specifications were tailored to be as user friendly and mobile as possible. These specifications allow the device to be versatile in its indoor placement through an end-user's home.

The list of fauna below are well documented plants that are kept in the database for the Smart Garden. The engineers varied from vegetables and herbs to give a broad range of options for the end user to choose from. The lithium-Ion battery will provide a reliable source of energy for the system to rely on.

Table 1: Project Requirement Specifications

|  |  |
| --- | --- |
| **Requirement ​** | **Specification​** |
| Accurate Soil Moisture Readings​ | Averaged readings from 3 sensors​ |
| Accurate Temperature Readings​ | +/-  0.5 Degrees Fahrenheit ​ |
| Accurate Humidity Levels​ | +/- 0.5%​ |
| Portable Design​ | Length  – 18 in.​  Width – 10 in. ​  Height – 8 in.​ |
| Dual Pump ​ | <70 L/Hr pump rate​ |
| Wireless Communication to Webapp​ | WiFi Connectivity - 2.4 GHz ISM​ |
| Weather Sealed​ | IP01 Baseline Water Resistance ​ |
| Relocatable ​ | Less than 10 pounds​ |

2.4 House of Quality

The House of Quality matrix is a tool that engineers use throughout the lifespan of the development process. The House of Quality, or HoQ, depicts the customers' wants and needs in a particularly useful manner. The HoQ takes these wants and needs and depicts them in a more delivery-oriented fashion stating the relationship between them with how achievable they are. Table 2 shows the house of quality model for this Smart Garden project.

The customer requirements side of our table shows the qualitative metrics that the user would want achieved out of their product. The customer requirements represent the most function-oriented requirements that are desired. In this House of Quality model, the complexity represents the inverse of how easily the device is to operate. In our model, this includes the initial installation process as well as its ease of use to the end-user. Cost, function variety and battery life also become important factors when viewing this Smart Garden from the end-user's perspective.

All of the targeted engineering requirements are very important factors to the innovators of this Smart Garden. These metrics are baselines for the engineers to reach throughout the research and development process. Things such as total installation time, cost, and overall accuracy of this machine all come into play with the engineering requirements section of the House of Quality model. Below in Table 1, is the key for the relationships used in our House of Quality model. It includes different Correlations, the direction of improvement and the relationships that are used in the House of Quality model.

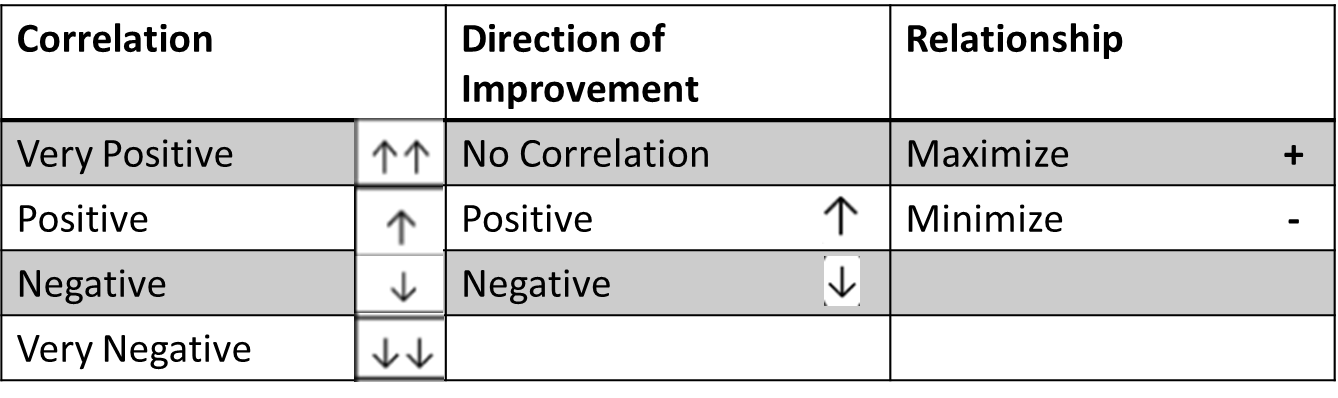


Table 2: House of Quality Key

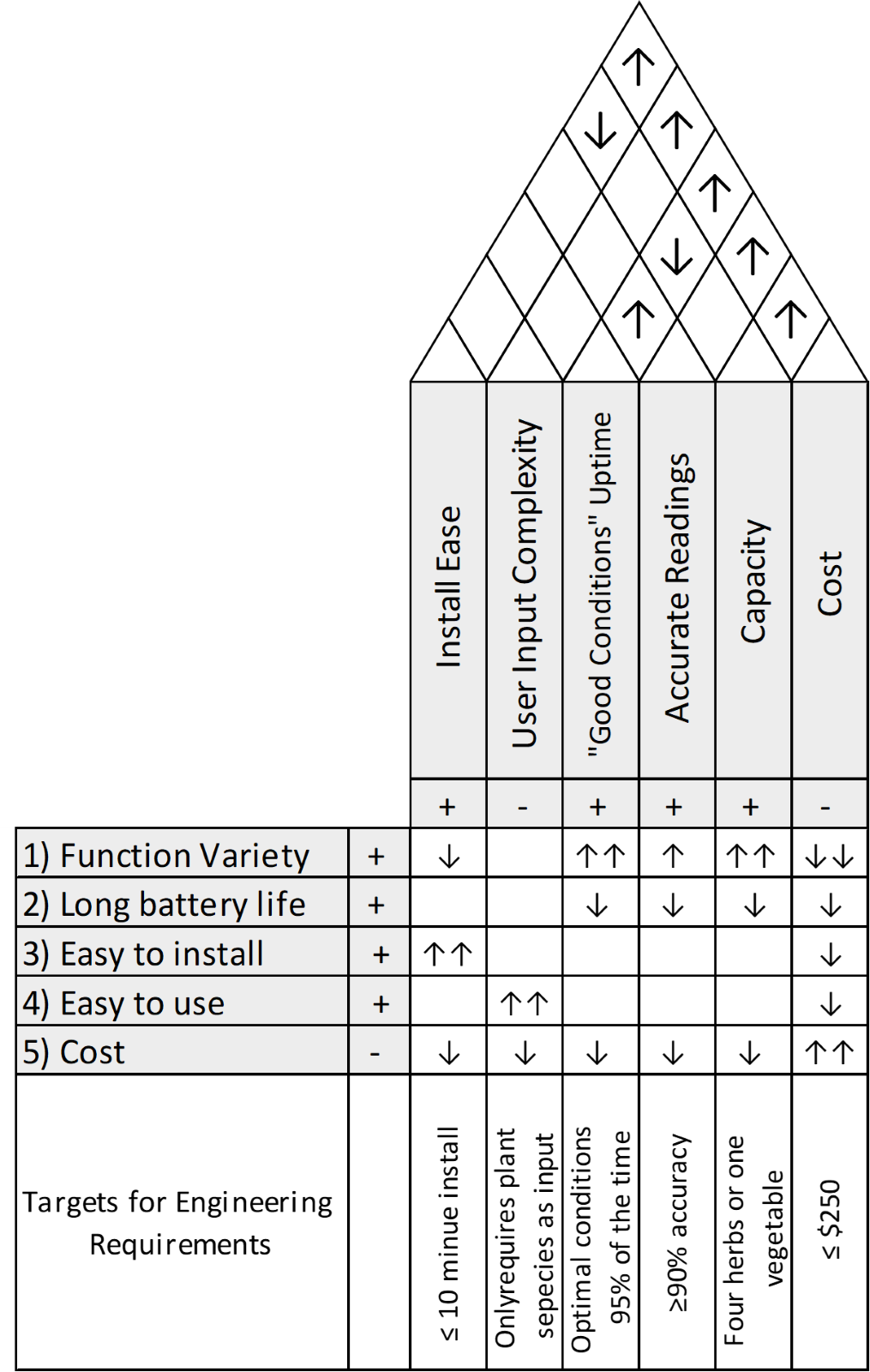


Figure 1: House of Quality

2.5 The Garden

There is a lot of value in the end user understanding how to properly operate and maintain the SMART Garden. This ties into the importance of capturing how the Smart Garden systems garden itself will be designed in the unit. Understanding the physical design of the garden which lies in the SMART Garden will help with overall cohesion on both the developers and end user's sides.

It is widely known, especially near coasts and more dry areas, that the soil found in the typical homeowners back yard may not always provide the nutrients necessary for ideal growth of the plants desired. This is becoming even more true on Florida's east coast, where the soil is more like sand rather than the proper sediment required. Of course, there are some remedies to this, including the use of plant feeder and other fertilizers. However, within a small closed and indoor system, the SMART Garden can optimize the growth and development of the end-users' favorite herbs and vegetables.

The SMART Garden will accompany up to 4 different plants, depending on size, growth needs and overall user desire. The water and nutrient pump lines will hang a number of inches above the soil and provide the proper amount of nutrients and water for the plant, respectively. These amounts are predetermined by the integrated dynamic database that the SMART Garden comes equipped with. Through this database, the user will have the option of inputting their own custom plant. However, the user must know the appropriate values to set for the plant respectively.

**Indoor System**

While this system is water resistant in all areas and water resistant in key areas, this system is equipped for indoor use only. Using IP01 standards will help keep the water in the housing and away from any important electronics or countertop. The SMART Garden is water resistant to ensure reliability and robustness through the end of its and the plants life cycle. However, this does not make the unit an outdoor device. This system is not designed to be left outside for extended periods of time. The AC to DC wall adapter should be a tell tale sign of that. It is designed in a one-size-fits-most for universal use within homes no matter how large or small they may be. With its compact and lightweight dimensions, the end-user should be able to accommodate the system to a variety of settings. This maximizes the effectiveness of this device and helps fit in to the normal user’s everyday life.

**Waterproofing and Water Resistance Needs of the System**

The Smart Garden system is waterproofed in the housing and made water resistant in key areas to ensure the robustness and reliability of the unit itself. The water resistance standards maintained through the SMART Garden are IP01 standards. This is effective to vertical water drips. Many of the garden's components are not susceptible to water damage. However, it is obvious that some of the main components are in a high-risk category if water were to come in contact with them. This includes the printed circuit board, the wiring, some of the sensors, and many of the wire connection points. In order to reduce water damage, these particular components were engineered to be as water resistant as possible. Moreover, these high-risk components (HRC) are placed separate from the hosuing in this first-generation prototype of the SMART Garden. Later iterations of the SMART Garden may boast on board fully waterproofed housing for the electronics. Sections below detail the different methods of waterproofing the SMART Garden unit.

3. Research and Decisions

3.1 Relevant Technologies

So many advancements have been made in every field that for any issue we might run into, there are multitudes of approaches and solutions. Whether they are tried and true classics or newly discovered techniques, all the possibilities deserve a chance to be considered. Even if an option is not considered, it may have been the ideal approach to the situation. Even if the option is not chosen in the end, it is good to know why it was not chosen. It is also important to acknowledge the shortcomings of an approach that is used in the final decision so that if any issues arise from its implementation, they are already predicted.

Additionally, at this stage in development, there could be many issues we will run into that we cannot predict. It is entirely possible that we realize that one of our choices has inadvertently led to an impasse in finishing the project due to some unforeseen limitation or interaction. In this case, we could very well be forced to consider one of the options we previously dismissed for one reason or another to take the place of our previous first choice. Even if we do not use the alternative choices at any stage in development, learning about them and their implementation can lead us to learn about new approaches to our problems.

3.1.1 Moisture Sensor

In order to get accurate and proper moisture readings, a reliable moisture sensor is essential. Below is a list of moisture sensors that were researched throughout the research and development period. After much consideration, the sensor chosen to move forward with was the HiLetgo LM393 Soil Moisture Sensor. This was particularly due to cost, implementation techniques and overall reliability. The other biggest factor that came into play during the selection process was the microprocessor that was going to be used. The soil moisture sensor was a very thorough research process and because of this the developers have a lot of confidence with the HiLetgo soil moisture sensor chosen.

**HiLetgo LM393 Soil Moisture Sensor**

This HiLetgo branded soil moisture sensor sports a fork shaped design. This is known to increase the robustness of these small sensors. This sensor has great reviews and reliability throughout the engineering community. This sensor comes in a pack of 5 units for about 16 dollars. It is simple, functional, and cost effective all in one small package.

**Waveshare Soil Moisture Sensor Module Detector**

This soil moisture sensor has a fork-like design, which increases functionality and durability of the system. It is 38 mm (about 1.5 in) in length. This sensor would be suited for many of the microprocessors mentioned below. A single unit of this sensor comes in at 10 dollars. This is slightly more expensive than other units that were found and researched for soil moisture sensors.

**Gikfun Capacitive Soil Moisture Sensor**

This sensor is well rated and best when paired with an Arduino unit. The pin layout for this unit is similar to the others having a ground, VCC and signal output. It has an operating voltage of 3.3-5.5 VDC. This unit has a chip built in for voltage regulation and looks to be a good match for the Smart Garden project. A two pack of this sensor comes in just under 10 dollars.

**SparkFun Soil Moisture Sensor**

The SparkFun soil moisture sensor is another forked shape sensor that can be placed into the soil. This unit is 12 dollars for a single unit but has high reliability and proven accuracy. The inputs for wires are much more robust than other units and would be suited well for most MCUs that are on the market. However, getting enough of this unit would drive the overall cost of the project past initial expectations.

**ACEIRMC Soil Moisture Sensor**

The ACEIRMC Soil Moisture Sensor is more of a modular unit than the others in this list. This unit comes in a set of two for 13 dollars. This unit has a working voltage of 3.3-12 volts. This probe would work best with an Arduino or Raspberry Pi. The cost is moderate, and it is reliable from the reviews.

3.1.2 Ultraviolet Light Sensor

When choosing an Ultraviolet light sensor for the Smart Garden the engineers prioritized the following factors: the size, operating voltage, price, reliability and compatibility of the unit. There were many more reliable, compatible and small UV sensors compared to the soil moisture sensors. However, ultraviolet light sensors tend to cost more than soil moisture sensors. This was something that was kept in mind throughout the process of choosing an Ultraviolet light sensor. The project will use the UV light sensor to detect it the plant is receiving the proper number of ultraviolet rays for the respective plant to flourish. The Smart Garden will use the *ML8511* ultraviolet light sensor. This sensor can detect UV/A, UV/B and UV/C rays and has a sensitivity setting on the device to adjust at which sensitivity level the unit lies at. More specifically, this unit can detect UV rays in the spectrum of 10nm to 400nm. This unit has 2mm mounting holes and is exactly 16mm (about 0.63 in)-22mm. This is a small compact unit that would pair well with the portability factor of the Smart Garden. The range of UV that the *ML8511* can pick up is far more than other sensors on the market for this price range. Although this sensor has a slightly higher price point, the quality makes up for it.

Other sensors the engineers looked at did not meet the same requirements as the *ML8511* ultraviolet light sensor module. Other considerations included the Comimark UV sensor, ACEIRMC UV sensor and the Adafruit 1918 UV light sensor. These were all good secondary options if the *ML8511* UV light sensor was not available, but they did not meet the same number of requirements that the *ML8511* met. The engineers also enjoyed that there was no need to solder on pin headers with the *ML8511* UV light sensor. The 3.3-to-5-volt operating range of the *ML8511* is standard to many of the other ultraviolet light sensor modules that are on the market. ￼

3.1.3 Humidity Sensor

The humidity sensor is put in place to make sure the plant is in a proper humidity level for their respective level. This humidity sensor is used to avoid an undesirable humidity level occurring and instead help stimulate favorable conditions for the end-user. The humidity sensor is used as an alert to the end-user. When choosing a humidity sensor, the engineer prioritized the following factors: the size, operating voltage, price, reliability and compatibility of the unit. There were a handful of units that read both humidity and temperature in the same sensor. This is favorable for the Smart Garden as it increases the unit's overall portability. The DHT11 by AITRIP Temperature Humidity sensor module fulfilled all major requirements for the Smart Garden system. This unit operates through 3.3 to 5 volts and is priced at 7 dollars for a set of two modules. This unit is described to have a plus or minus 5 percent accuracy range for humidity. However, since humidity is not one of the two most vital data readings this proved to be acceptable. This sensor would pair well with Arduino units as well as a handful of other units on the market.

However, the DHT22 Digital Temperature and Humidity Sensor by Aideepen met all requirements with a more accurate sensor. It had a plus or minus 0.5-degree error for temperature sensor. These specs are better than any other temperature and humidity sensor in this price range of 7 to 10 dollars. Therefore, the DHT22 Digital Temperature and Humidity Sensor will be implemented in this project.

3.1.4 Temperature Sensor

The temperature sensor, while not as important as some of the other sensors, still is used as a valuable tool for the Smart Garden system. Dangerously low or extremely high temperatures can be fatal for any plant, specifically smart vegetables and herbs. When looking at temperature sensors the engineers favored units that were reliable, compatible and used in the same unit as the humidity sensor. The ability to combine these two systems into a singular unit would prove to be very important for the Smart Garden system as it increases the overall portability of the Smart Garden system. The DHT11 by AITRIP Temperature Humidity Sensor module proved to fit the mold for both the Humidity and the temperature sensor requirements. The two-piece set is only about 7 dollars and keeps a low profile. It also has an operating range of 3.3 to 5 volts as many of the other sensor modules that are implemented in the Smart Garden. This specific unit is known for its accurate and reliable temperature readings. There is room for error in this unit as described in the description of plus or minus 2 percent, but since temperature and humidity are less of a vital metric this was proven as acceptable. The temperature unit of this module can read up to 50 degrees Celsius, far higher than the Smart Garden will reach.

However, the DHT22 Digital Temperature and Humidity Sensor by Aideepen met all requirements with a more accurate sensor. This sensor has an up to 2 percent error on the relative humidity sensor.

3.1.5 Displays

The Smart Garden system will implement an interactive display into the unit for increased user experience. The Liquid Crystal Display or LCD screen is a key component in the Smart Garden system. This LCD screen will house all the important information for the user to read. This includes the flora diagnostics, such as the humidity, temperature, soil moisture levels and the amount of UV the plant is receiving. It will also be the home to selecting which of the predetermined plants the end-user is choosing to maintain. Based off this selection, the system will cater to the programmed inputs for the respective plant.

OLED displays, or Organic Light Emitting Diodes, is another typical screen technology used throughout the world. This technology is an advancement in display technology and boasts deeper colors than other display technologies. This type of display is desirable, but more expensive than your basic LCD displays. The cost versus gain analysis for what type of display to use was a challenging process for the Smart Garden system.

There are a handful of factors to consider when choosing the proper LCD screen for the Smart Garden system. Factors such as the price, size, resolution and overall quality are all important to the innovators of the Smart Garden. The initial price range was anywhere from about 7 dollars to well over 60 dollars. Due to the aim of keeping the Smart Garden budget friendly, the engineers allowed a budget up to thirty dollars for the screen. Below are some of the different LCD screens that were researched and screened to see if that LCD screen made the cut.

**SunFounde LCD display – 20x4**

The SunFounde LCD screen is a twenty column by two row LCD that has a working voltage of 5 volts. This is the typical I2C display that can be found just about anywhere at any time, therefore it is very accessible. This unit comes in at about 10 dollars and would be a standard and sufficient choice for displaying normal data and information over a 4 row by twenty column area. There can be a total of 2004 display characters on this display.

**Waveshare 2.23inch OLED Display**

A larger OLED display coming in at 2.33 inches is the Waveshare 2.23-inch OLED display. This display features a higher resolution and more immersive experience for the user at the cost of a more expensive unit overall. There is much more customization with. This unit is due to the OLED technology of the typical old-fashioned LCD screen above. This module boasts an SSD1305 driver on board with communication via I2c or SPI.

3.1.6 Number Pad

The number pad is a typical input device for a plethora of devices. Whether the number represents an actual value or an arbitrary command, the number pad is an efficient way to communicate an input to the microprocessor at hand. There are a number of options, but realistically most of these number pads do exactly the same function to the same accuracy and reliability, Therefore, the Smart Garden system does not use too much budget towards the actual number pad module itself.

The smart garden system will implement the number pad in a slightly different fashion. Most number pads are programmed to input a number onto a display. However, the Smart Garden system will utilize the number pad to make choices and simplify the process for both the developer and the end-user. The number pad will choose different commands and will also be implemented to choose the plant at hand in the Smart Garden system. For Instance, if you choose number “1” on the number pad under the list of plants, the first plant on the list will be selected and the Smart Garden System will maintain the plant based on that data.

The number pad could be used for a number of different features. These features can extend beyond these few basic commands. The engineers will continue to implement and innovate the number pad functionality as the Development process continues. Possible innovations include a side setting where the end-user can manually edit or override system settings such as the temperature warning.

**Adafruit 3 by 4 Number Pad**

This number pad features the digits zero through nine. It is arranged in the standard number pattern for familiarity. The price comes in at about 7 dollars which is on the lower side of cost for modules like this. It functions the same as most, where the input corresponds to the digit on the button. This module also features an asterisk and pound symbol. These additional two keys can be edited to hot keys of the engineer's choice.

**Pimoroni Pico RGB Keypad**

The Pimoroni Pico RGB keypad represents what high tier keypads are like. The key difference between a high-end unit like the Pico and the lower end unit is the build quality. However, build quality in this circumstance is something the developers were willing to sacrifice in the name of cost savings for both the developers and the end-users. The purpose of this Smart Garden system was to provide an economic solution to garden management. Using an expensive 25-dollar keypad defeats the mission of the Smart Garden system.

3.1.7 Solar Panels

Solar panels add to the technologically advanced gardening management system in many beneficial ways. Solar panels will aid the battery depletion constraint as well as make the UV light sensor more valuable to the Smart Garden system. Unlike a number pad for an Arduino unit, there is a lot of prices variation as well as performance variation. The key to a good solar panel for the Smart Garden system is finding the performance vs cost maximization. The Smart Garden does not need the highest model solar panel. However, the system needs a powerful enough unit in order to fill up the rechargeable battery to a safe operating level. Without the solar panel having enough power to supply a charge to the battery, there is no point in implementing the solar panel to start.

The main factors for choosing the appropriate solar panel array for the Smart Garden became one of the biggest research topics among all the different peripherals researched during the developmental period. The system required a solar panel that would have enough power to charge the on-device battery while also providing to be a good value. The developers wanted to maximize this relationship as much as possible. After due-diligence and research, the top two solar panel arrays were the following: *AOSHIKE 30mA mini solar panels* and the *RedTagCanada Solar Panels (25mA)*.

The AOSIKE solar panels and the RedTagCanada panels both sport 5 volts but the AOSIKE panels provide an extra 5mA of current. It is important to note that while both of these would suffice for our project, the AOSIKE maximizes the relationship mentioned earlier. For an extra dollar or two the end-user receives an extra bump in utility and power from this device. No matter which solar panel is used in the Smart Garden system, there will need to be a diode to prevent any sort of backflow into the solar panel. This diode will act only as a block for backfill and will be very low cost. See the schematic below:

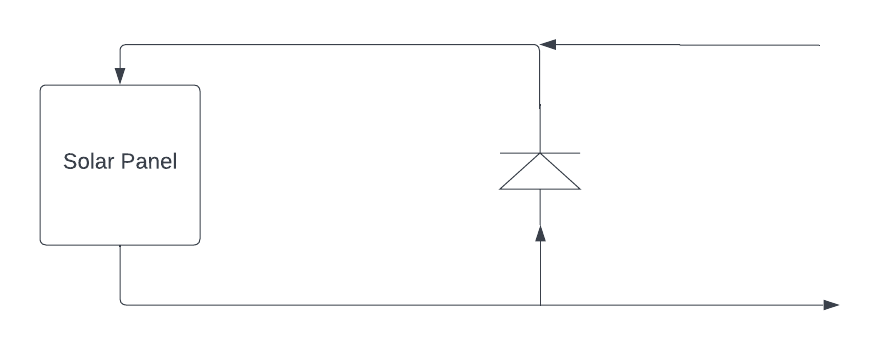


Figure 2 Basic Solar Panel Outline

3.1.8 Batteries

Batteries store chemical energy for later use as electrical energy and are typically small enough to be easily moved from place to place. This makes them a great replacement for a connection directly to the power grid where such a connection is not viable or is too impractical. Since batteries are relatively low cost, they are easily accessible to consumers when a device that requires them to run has depleted them of their charge. Batteries are available in a few varieties, as shown and compared below. All types below are available in a variety of sizes, which correspond to higher or lower voltages.

*Non-rechargeable*

Batteries that are not rechargeable are single use. Once they are depleted, they are trash to be disposed of or recycled safely. The two main types of non-rechargeable batteries are alkaline batteries and lithium batteries. Alkaline batteries tend to have a longer shelf life, but less power output otherwise. A single AA lithium battery stores 3500 mAh (milliamp hours), while an alkaline battery from the same manufacturer stores only 3000 mAh.

*Rechargeable*

If the device in use is expected to have heavy use, it will likely need its batteries replaced often. Instead of purchasing new batteries every time, consumers can instead purchase rechargeable batteries so that they can simply recharge the battery at home. There are three primary varieties of rechargeable batteries- Nickel-Cadmium (NiCd), Nickel-Metal Hydride (NiMH), and Lithium Ion (Li-ion). NiCd batteries have issues when charging them back up without first fully discharging them and have an overall lower voltage than the other types. NiCd batteries can be ‘cycled’ around 1000 times before they degrade too much to use further. NiMH batteries can also be cycled around 1000 times before needing replacement, and have much larger charge capacity, all without the charging issue faced by partially discharged NiCd batteries. The tradeoff for this is that they’re typically more expensive than NiCd batteries. Li-ion batteries solve the partial-charge issue and have a greater capacity as well.

With the format of the batteries now decided, the variety is left to be chosen. The capacity of D batteries is tempting, but they only provide about 1.5 volts per battery. Since the microcontroller we chose has a recommended operating input voltage range of 7-12 volts, we would have to use at least 5 D batteries, most likely 6 to account for imperfect batteries, to provide enough voltage to ensure the microcontroller functions as expected in all scenarios. For the prototype at least, a 9-volt battery was chosen, since it would be small enough to not cause components to be crowded, but also provides enough voltage to prevent loss of microcontroller function.

3.1.9 Waterproofing

Aside from the reservoirs that are designed to hold it, we don’t want to end up with water inside the SMART Garden. The only things we want to get wet in any capacity are the moisture sensors, the feeder systems, and the plant itself. The wiring and the PCB would likely get damaged by the presence of water. If we can be certain that there is no water damage, that would result in having one fewer issue to check when it comes to testing. With fewer points of failure, there are fewer things that could cause issues in the chain of reliance. There are several ways to prevent water from causing any problems with the function of our project, some of them are listed and discussed below.

*Circuit board waterproofing spray*

Instead of leaving it up to simply sealing any cracks and gaps between components that could let water into the device and onto the circuitry, waterproofing spray, such as acrylic conformal coating can be applied directly to the circuitry in question. This coating dries around each component, protecting the PCB from moisture, dust, and fungus.

*Waterproof Enclosures*

Rather than entrusting all the waterproofing to a coating on the circuit boards, it may be possible to prevent the water from entering the device at all. Using sealant on the gaps between the pieces of the device’s shell, we can try to create an unbreachable connection that would withstand all weather conditions that we could predict. This could be done with something like endothermic sealant, a latex-based sealant that is applied like caulk, and blocks water once dry.

*Heat Shrink Wire Tubing*

After the circuitry is coated and the gaps in the shell are sealed, the only thing left is the wires inside connecting each part together. One possible solution is heat shrink tubing that tightens around the wires, providing a watertight seal. One popular choice for this would be polyolefin heat shrink tubing. Once it is applied to the wires and heated up, it shrinks down against the wires, providing a tight seal wherever there is a rough connection between wires.

The Decision

The ideal way to prevent water damage will be to keep the water from getting in at all. The 'seams’ of the case are the least fragile part of the garden, therefore it would make sense to reinforce them as much as possible and prevent water from getting deeper at all. Utilizing all or some of the other options would still be prudent, but if all goes well with the outer case, there will be no need to rely on them. We would simply stop the water from entering in the first place. Rather than applying coatings that might damage the components or even just hinder heat dispersion, the best option would likely be to make those options unnecessary. Preventing water from entering the casing by reinforcing the seams of the case against water should be all we need.

3.1.10 Water and Fertilizer Pump Mechanism

The water and fertilizer pump mechanism are critical parts of the Smart Garden system. After the moisture sensor drops below a certain threshold for the respective plant, it sends a flag to trigger the water pump until minimum moisture levels are met. A very similar process is done with the fertilizer reservoir and pump system. On a scheduled basis that is taken from data logs, the fertilizer will dispense the proper amount. This requires the pump to be reliable and consistent with the amount of water it pulls from the reservoir.

There are a handful of moving parts that go into creating a fully functional water pump. The main components are: The pump itself, the MCU unit, the reservoir, the water, and the tubing. The pump needs the ability to accurately dispense the amount of water necessary to saturate the soil to appropriate levels for that plant. The MCU unit bridges the mechanics with the logic to have everything operate properly. The reservoirs hold both the water and the fertilizer to be dispersed into the garden itself. Finally, the tubing is the method of transportation to allow the water and fertilizer to reach the garden. The tubing will have a series of small punctions in the lining, allowing for an even pour along the Smart Garden system. The same will hold true for the fertilizer system. The fertilizer system may require more maintenance by the end-user, considering it is not as clean of a mix as the fresh water will be. However, the small amount of maintenance required far outweighs the personal management a normal plant would require.

There were *three* pumps that made it to the final round of research and review are listed below. All three of these pumps would do a serviceable job. However, in the mindset of maximizing performance while minimizing cost, only one made it above the others. See all the relevant information on these units below.

**Siphytoph DC 3-5V Micro Submersible Mini Water Pump**

This unit comes in a two pack for about 7 dollars. It operates between 3 and 5 volts and comes with some tubing. The unit is small in size, about 43 millimeters in length for reference. Included is 8.20-millimeter diameter tubing, which may suffice as the tubing used in final design. This unit has well received reviews, especially for its performance versus price comparison. It is said to function just as specified for a low price point. The liters per hour rate was not provided for this unit. However, users mentioned that the unit was constant and steady.

**Azuocon DC 3-5V Micro Submersible Mini Water Pump**

This unit is particularly similar to the Siphytoph mini pump, but with some slight advantages. This unit also has well received reviews, especially for its performance versus price comparison. It is said to function just as specified for a low price point. Moreover, this unit comes in a pack of 3 units for the price of about 8 dollars. This is a significant boost in price versus performance maximization. This unit boasts a *100 liter per hour* flow rate. This is far more than sufficient for the Smart Garden system.

**Reland Sung 3-6V Mini Micro Submersible Water Pump**

The third available final choice for the Smart Garden system was the Reland Sung 3-6 volt Mini–Micro Submersible Water Pump. This pump is the cheapest of the bunch at around 2.50 per unit but comes in a set of a single unit. This pump is 20 percent more efficient than the next best, as it pumps around *120 liters per hour.* However, there are two main drawbacks to the Reland Sung water pump. The first being delivery date. These Reland Sung units are low in stock and would take about a month to arrive. The second major fallback is the fact that the unit itself has no credibility. Although the price is the lowest of the bunch, there were no reviews to back the performance of this device. This makes it hard to confirm it would be a good fit in the Smart Garden system.

3.1.11 Water and Fertilizer Reservoirs

A secure way to hold the water and fertilizer for the Smart Garden system is key for longevity of the device. While the device will have a water-resistant aspect, the product is not advertised as a water-resistant device. There are no standards that the smart garden follows for water resistance, just the forethought of the developers to minimize water damage as much as possible.

*Option one* for the reservoir is a hard sided container with a lid and tubing that leads to the main pump for the respective liquid being used. This provides a structurally sound unit that would be difficult to spill. However, the variety can be limited. The cost of a hard-sided container is also higher than other alternatives, comparably speaking. However, the argument can be made that spending more on a hard sided reservoir is worth keeping other components safer.

Option two for the reservoir is a soft sided container also containing a sealed top leading to the tubing which connects to the main pump unit. The advantage to a soft sided unit is the versatility in size, variety, and overall ergonomics. However, the soft sided container would be more prone to spillage.

3.1.12 Relay Module

The relay module will be the source that operates like a master switch to the two reservoir pumps. The Arduino will gather data from the soil moisture sensor. When appropriate, the Arduino will raise an interrupt which then triggers the relay module and allows the pumps to activate and begin pumping the water and/or the plant food into the plant basin. During this process, the engineers determined that only a two-channel relay module would be necessary. However, if a higher multi-channel relay module was the best cost to performance ratio, then the engineers would consider it for add-ons throughout the development process. Basic features that were either required or very desired for the relay module were:

* 5 volts
* At least 2 channels
* Reliability
* LED Indication lights

Below are a series of different multi-channel relay modules that were researched and decided between.

**SunFounder 2 Channel DC 5V Relay Module**

This two-channel relay module comes in at 7 dollars for the peripheral. It operates at 5 volts and needs 15mA per channel to operate as indicated. This module has a 4.6/5.0 rating with well over 900 ratings on this particular device. Many speak of its reliability and the overall ability to function as intended. It has indication LEDs to relay quick information back to the engineer during testing and development as well as in-field operation. This unit's size is approximately 51mm by 38mm by 19 mm. Overall, this unit is exceptionally priced with high ratings and reliability. Just about everything a user would like to see out of a multi-channel relay module.

**Excelity 2 Channel DC 5V Relay Module**

The Excelity 2 Channel DC 5V relay module is another well reviewed, high end relay module for a low price. This module, like the last, is made to be operated with microprocessors like the Arduino. Many of the features are the same as the SunFounder but there are a few key differences. First, the operating switching current is 10mA, compared to the SunFounders 15mA. The other key difference between this module and the SunFounder is the reviews. This Excelity relay module has a solid 4.0/5.0 which is a respectable review pool. However, this is out of 17 total ratings compared to the SunFounders 900 plus ratings. Although the operating current may be lower, so is the number of reviews. This module is also 7 dollars and has a series of indications LE

**10Gtek 5V 1 Channel Relay Module**

Instead of a two-channel relay module, the 10Gtek is a single-channel relay module which comes in a quantity of two. For all intents and purposes, this would operate as needed just like a two-channel module would. This two, single channel module is appealing due to the ability to have the relay modules in different physical places on the SMART Garden. A two pack of the 10Gtek comes out to 5 dollars. Moreover, it has the 10mA current ratings. Unfortunately, this product has many instances of unreliability. For this reason, the 10Gtek will not be used.

**Qunqi 5V 2 Channel Relay Module**

This module boasts some very desirable features compared to other relay modules that the engineers had researched. It is 5 volts, as all the other relay modules. However, for the price of 11 dollars the consumer receives four relay modules. This along with well over 100 reviews on the positive side of things, this module has the best bang for its buck. It has indication LEDs for relaying output status to the user. This module is also compatible with the Arduino Uno R3. The biggest downfall of this unit is the fact that only 1 two channel relay module is needed for the SMART Garden, instead of the four that are provided. All in all this unit and the SunFounder are the two most desirable units out of those researched.

**The Decision**

After reviewing 4 different relay modules for the SMART Garden, the decision was narrowed down to two different modules for the final choice. This was between the Qunqi and the SunFounder. Both have very similar operations and side features like the indication LED. At the end of researching both, there were two main reasons that the *Sunfounder was the final choice* to be used with the Smart Garden System. The first reason was due to the price. Although the Qunqi came with an additional three units, the price was about 40 percent higher. Moreover, the reviews were not quite as positive or outlasting as the SunFounder. The SunFounder 2-channel relay module became the module of choice to be used for the SMART Garden system and the two reservoir pumps.

3.1.13 Wi-Fi and Bluetooth Modules

The Wi-Fi and Bluetooth modules explored were the two main ways to connect the SMART Garden to the end-user. After much research and debate on which to use, the engineers decided that the Wi-Fi module would be the best route to pursuit.

The main reasoning behind a Wi-Fi or Bluetooth module is simple. The Smart Garden system works directly with the application and needs to do such through a Bluetooth or Wi-Fi connection. Wi-Fi is a stable and reliable solution to this. With a WiFi module on board the SMART Garden, the application will have a way to both send and receive information to and from the Smart Garden system. Below are the three final options that the engineers decided between.

**Makerfocus 4pcs ESP8266 Esp-01**

This serial unit Wi-Fi unit comes in at 13 dollars for a 4 pack on Amazon.com. It features all standard Wi-Fi features and connectivity that every unit comes equipped with. Moreover, there are a series of different videos and manuals that come along side this unit for reference. This model is also Compatible with Arduino. Support 3 modes: AP, STA, AP + STA. Overall, this unit appears to be a good bang for the buck with good reviews and quick shipping. This unit is in stock and seems to have no history of stocking issues.

**KEYESTUDIO 3.3V 2Pcs ESP8266 ESP-01**

The second unit that made the top three options for the Smart Garden System is the Keyestudio ESP8266 ESP-01. This seems to be a very similar unit to the last, but come in a package of 2 for 10 dollars. Like the Makerfocus, this unit is compatible with the Arduino uno. However, this unit has slightly better reviews and comes in slightly cheaper (although more expensive per unit). The quality control of this module seems like a key point over the Makerfocus ESP01.

**WOWOONE 2PCS Wemos D1 Mini Arduino Uno WiFi Shield**

This module is far different from the first two. The WOWOONE is a shield which covers the whole top of the Arduino. This unit was built and designed specifically for use with the Arduino uno line of microprocessors. This is perfect for the Arduino Uno R3 that is implemented with the Smart Garden system. The two pack of shields come in at 12 dollars, but minimizes confusion. There are many positive reviews backing the performance of this Wi-Fi shield. There is also a Bluetooth module built into this shield as an addition to the Wi-Fi.

3.2 Microcontrollers and Microprocessors

A microcontroller or microprocessor serves as the main computing power for our project. It is a device whose purpose is to gather a desired input or inputs, processes that information, and perform a specific task as a result of the information gathered. The success of our project relies heavily on the efficiency and reliability of the MCU that is chosen. With that being said, it is crucial to choose a microprocessor that has the ability to seamlessly integrate all components of our system, with negligible loss to our systems efficiency, while still staying within our overall project budget.

In figure 2 below we see that the MCU is the core of our project's hardware set-up. It is connected not only to all four of our sensors (which are the main desirable features for our project) but also to the relay modules, LCD display, water pump, and power supply. In addition to that, it also plays a key role in how we establish wireless communication to ensure our user has access to the mobile app or website that will allow them easily to interface with the system. All aspects of our project success are dependent on this component so the decision will only be made after in-depth research and analysis.

Chart, diagram

Description automatically generated

Figure 3: Hardware Block Diagram

3.2.1 Development Boards with on-board Microcontrollers

In this section we will discuss three potential choices for our project's development board unit. After some research and discussion our group found the following to be our top three choices: Arduino Uno R3, MSP 430, and STM32F103C8T6. Below is a general overview of all three as well as a brief analysis of the advantages or disadvantages they would have if implemented in our hardware design.

**Arduino Uno R3**

The Arduino Uno is a microcontroller board that is based on the ATmega328P. It offers 32 KB of flash memory and 2 KB of ram. This microcontroller has a total of 14 digital input/output pins, 6 of which are analog input pins and 6 of which are PWM pins. The Arduino Uno supports UART, I2C, and SPI communications and features 1 KB of EEPROM (electrically erasable programmable Read-Only memory) which is memory that does not get erased when the device is turned off. The microcontroller has a required input voltage of 7-12 V and has I/O voltage of 5 V. It also comes with a barrel plug power supply connector which can be easily hooked up to a standard 9V battery. The benefits of this microcontroller are as follows: minimum power consumption, cheap price in comparison to competitors, simple and easily programmable interface, and a large user interface.

**MSP 430**

The MSP430 is a line of microcontrollers made by Texas Instruments. The specific chip we will be looking at is the MSP430G2231. This microcontroller is typical in applications that include low-cost sensor systems that take analog signals, convert those signals to digital values, and process the captured data to be displayed or transmitted to a host system. The benefit of this microcontroller would be its ability to achieve extended battery life in portable measurement applications such as the one we are trying to achieve with the SMART Garden. This device supports I2C and SPI communications and comes with the 16-bit core processor MSP430 CPU16. The 16 MHz MCU has 2 KB of flash memory and 128 B of SRAM. Lastly, the MSP430G2231 has 10 I/O pins and comes with a built-in 16-bit timer.

.

**STM32F103C8T6**

The STM32F103C8T6 is a member of the medium-density performance line of microcontrollers known as STM32F103xx. This MCU features an ARM® Cortex®-M3 32-bit RISC core which operates at a maximum frequency of 72 MHz. The STM blue pill comes with the option of either 64 or 128 KB of Flash memory and 20 KB of SRAM. The device supports up to two I2Cs and SPIs, up to 3 USARTs, a CAN interface, and a USB 2.0 full-speed interface. It also has 37 GPIO pins, 10 of which are analog, and 15 of which are PWM. The microcontroller has an input voltage of 5V, an operating voltage of 3.3 V, and a built-in micro-USB connector. The STM32 blue pill offers nearly the same capabilities as the Arduino previously discussed but can be purchased for a cheaper price. Drawbacks to the device are as follows: the 32-bit system results in slightly slower execution time, the STM blue pill is mechanically incompatible with most Arduino shields, and the device is known to have poor quality assurance resulting in a higher likelihood of receiving an unusable product.

3.2.2 Development Boards with on-board Microprocessors

In this section we will discuss three potential microprocessors that could be used in our project. After some research and discussion, our group found the following to be our top three choices: ASUS Tinker Board S, Raspberry Pi 4, and UDO Bolt v3. Below is a general overview of all three as well as a brief analysis of the advantages or disadvantages they would have if implemented in our hardware design.

**ASUS Tinker Board S**

The ASUS Tinker Board S is a microprocessor that utilizes the quad-core ARM-based processor known as the RK3288. This device offers 2 GB of LPDDR3 dual-channel memory and operates at a frequency of 1.8 GHz. The microprocessor offers storage in the form of an onboard 16 GB eMMC and utilizes a micro-SD card. The Tinker Board S has 40 GPIO pins with enhanced I2S interface, features a full-size HDMI output, and includes four USB 2.0 ports. The benefit to this board is its large number of I/O pins and onboard eMMC storage that alleviates some issues dealing with an SD card. The drawbacks to this device are the cost, smaller community, and overheating struggles

**Raspberry Pi 4 Model B**

The Raspberry Pi 4 Model B is a popular microprocessor in the Raspberry Pi range of computers. The Model B uses a 64-bit quadcore processor that operates at up to 1.5 GHz. The product includes up to 4 GB of RAM, Bluetooth 5.0, dual-band 2.4/5.0 GHz wireless LAN, two USB 3.0 ports, two USB 2.0 ports, and Gigabit Ethernet. The device comes with a standard 40 pin GPIO header (backwards compatible) and includes a micro-SD card slot for loading operating systems and data storage. The Raspberry Pi 4 offers video and sound options hosting two micro-HDMI ports, a 2-lane MIPI DSI display port, a 2-lane CSI camera port, and a 4-pole stereo audio and composite video port. Input power is 5 V DC via both the USB-C connector and the GPIO header. The benefits of the Raspberry Pi are as follows: low power consumption, offers large userbase and helpful community, compatible with most Arduino accessories, supports multiple sensors, and unlike the Arduino that only supports coding languages C and C++ the Model B supports a wide variety of coding languages such as C, C++, C#, Ruby, Java, Python, etc. One of the disadvantages of the Raspberry Pi is its lack or internal storage which results in reliance on its SD card. This increases the board’s overall boot time as well as the Raspberry Pi’s read/write speed. Another disadvantage of the Raspberry Pi is the cost. Overall, for the quality of product you are getting the price is not bad but compared to the Arduino, MSP 430, and STM32 it is substantially more costly.

**UDO Bolt v3**

The UDO Bolt V3 CPU is the AMD Ryzen Embedded V1202B Dual Core. It operates at a maximum frequency of 2.3 GHz. It has two USB 3.0 Type-A sockets and two USB 3.1 Gen2 Type-C connectors. The required power supply is 19V DC and it offers 32 GB of RAM. The device supports Linux and Windows 10 64-bit operating system. The UDO Bolt V3 has a one Gigabit Ethernet connector and a Slot M.2 Socket 1 Key E 2230 for optional Wi-Fi/Bluetooth combo. The microprocessor has 40 total I/O pins of which up to 23 are digital pins (7 PWM) and at least 12 are analog pins. The board offers multiple video interfaces and video resolution of up to 4 K. The main fields of application for this board are automation, gaming, multimedia devices, visual computing, and education. The benefits to this board are its power and wealth of features. This disadvantages to this board would be the incredibly high cost in comparison to its counterparts. With that being said, if our we were doing a project that required that level of computing power this board would be a top contender. Although unfortunately the SMART Garden has no use for the large quantity of extra features so the benefit would not be worth the cost.

**Decision**

After careful deliberation the group decided the best microcontroller for our project would be Atmega 328P-PU which is on board the Arduino Uno. The reason for this is due to the groups collective comfortability with the company and system, the simplicity of the programmable interface, the low cost, and last but not least its low power consumption. The Arduino Uno is also a widely used development board that has a large amount of open-source material as well as an extensive community of users. The easy unplug and plug into a PCB factor made this development board and microprocessor a no brainer. Having the ability to code and test on the Arduino development board and switch the Atmega onto our own printed circuit board was extremely valuable to the team. We as a group did not feel the need to spend excess money to get a high-powered microprocessor when the demands of our project do not call for or need that level of computing power.

3.2.3 Development Board Comparisons

|  |  |  |  |
| --- | --- | --- | --- |
| Microcontroller/  Microprocessor | Processor/Clock Speed | Operating Voltage | Unit Price |
| Arduino Uno R3 | ATmega328P/  16 MHz | 5 V | $27.6 |
| MSP 430 G2231 | MSP430/  16 MHz | 1.8V ~ 3.6V | $7.21 |
| STM32F103C8T6 | ARM® Cortex®-M3 32-bit RISC core/  72 MHz | 3.3 V | $7.64 |
| ASUS Tinker Board S | Rockchip RK3288/  1.8 GHz | 5 V | $149.99 |
| Raspberry Pi 4 Model B | Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit/  1.5 GHz | 5 V | $55 |
| UDO Bolt v3 | AMD Ryzen Embedded V1202B Dual Core/Four Thread/  2.3 GHz | 19 V | $74.99 |

Table 3: Microprocessor Comparisons

|  |  |  |  |
| --- | --- | --- | --- |
| Development Board/  Microprocessor | I/O Pin Count | RAM | Operating Temp. |
| Arduino Uno R3 | 14 pins | 2 KB | -40°C to 85°C |
| MSP 430 G2231 | 10 pins | 128 B (SRAM) | -40°C to 85°C |
| STM32F103C8T6 | 37 pins | 20 KB (SRAM) | -40°C to 85°C |
| ASUS Tinker Board S | 40 pins | 2 GB | 30°C - 80°C |
| Raspberry Pi 4 Model B | 40 pins | 4 GB | 0°C and 85°C |
| UDO Bolt v3 | 40 pins | 32 GB | 0°C and 70°C |

Table 4: Microprocessor Comparison - Pins, RAM and Temperature

3.2.4 Moisture Sensor Module

|  |  |  |  |
| --- | --- | --- | --- |
| Moisture Sensor Module | Reliability | Operating Voltage | Unit Price |
| HiLetgo LM393 | Average Reliability | 3.3-5.5 Volts | $ 3.20 |
| Waveshare | Very Reliable | 3.0-5.0 Volts | $ 10 |
| Gikfun | Average Reliability | 3.3-5.5 Volts | $ 5 |
| SparkFun | Very Reliable | 3.0-5.0 Volts | $ 12 |
| ACEIRMC | Above Average Reliability | 3.3-12 volts | $ 6.50 |

Table 5: Moisture Sensor Module Comparison

3.2.5 UV light Sensor Module

|  |  |  |  |
| --- | --- | --- | --- |
| UV light Sensor Module | Detectable Range of Ultraviolet | Operating Voltage | Unit Price |
| Comimark UV sensor | 200nm-350nm | 3.3-to-5.0 Volts | $ 11.39 |
| ACEIRMC UV | 200nm-320nm | 3.3-to-5.0 Volts | $ 12.15 |
| Adafruit 1918 UV light sensor | 200nm-350nm | 3.3-to-5.0 Volts | $ 10.31 |
| *ML8511* UV light sensor | 10nm-400nm | 3.3-to-5.0 Volts | $ 15.95 |

Table 6: UV Light Sensor Module Comparison

3.2.6 Humidity Sensor Module

|  |  |  |  |
| --- | --- | --- | --- |
| Humidity Sensor Module | Possible Maximum Error (+/-) | Operating Voltage | Unit Price |
| AITRIP DHT11 | 2.5 percent | 3.3-5.0 Volts | $10 |
| Aideepen DHT22 | 2 percent | 3.3-5.0 Volts | $8 |

Table 7: Humidity Sensor Module

3.2.7 Temperature Sensor Module

|  |  |  |  |
| --- | --- | --- | --- |
| Humidity Sensor Module | Possible Maximum Error (+/-) | Operating Voltage | Unit Price |
| AITRIP DHT11 | 2 -degrees Celsius | 3.3-5.0 Volts | $ 7.15 |
| Aideepen DHT22 | 0.5-degrees Celsius | 3.3-5.0 Volts | $ 10.35 |

Table 8: Temperature Sensor Module

3.2.8 On-Unit Display

|  |  |  |  |
| --- | --- | --- | --- |
| Display | Display Technology | Operating Voltage | Unit Price |
| SunFounde LCD display | Liquid Crystal Display | 3.3-5.0 Volts | $ 9.99 |
| Waveshare 2.23inch OLED | Organic Light Emitting Diode | 3.3-5.0 Volts | $ 23.59 |
| TUXGR\_16X2\_R2 LCD | Liquid Crystal Display | 3.3-5.0 Volts | $ 9.95 |

Table 9: On-Unit Display

3.2.9 Reservoir Pump

|  |  |  |  |
| --- | --- | --- | --- |
| Pump | Flow Rate | Operating Voltage | Unit Price |
| Siphytoph Mini Water Pump | N/A | 3.0-5.0 Volts | $ 3.27 |
| Azuocon Mini Water Pump | 100 Liters per hour | 3.0-5.0 Volts | $ 1.89 |
| Reland Sung Mini Micro Submersible | 120 Liters per hour | 3.0-6.0 Volts | $ 2.50 |

Table 10: Reservoir Pump

3.2.10 Reservoir

|  |  |  |  |
| --- | --- | --- | --- |
| Reservoir | Maximum Capacity | Hard Sided/Soft Sided | Unit Price |
| 12oz Tumbler | 12 Ounces | Hard Sided | $ 1.60 |
| TRIWONDER Soft Pack | 8.45 Ounces | Soft Sided | $ 7.50 |

Table 11:Reservoir

3.2.11 Multi-Channel Relay Module

|  |  |  |  |
| --- | --- | --- | --- |
| UV light Sensor Module | Number of Channels | Quantity of module | Unit Price |
| **SunFounder** | 2 | 1 | $ 7.00 |
| **Excelity** | 2 | 1 | $ 7.00 |
| **10Gtek** | 1 | 2 | $ 6.00 |
| **Qunqi** | 2 | 4 | $ 11.00 |

Table 12: Relay Module

3.2.12 Wi-Fi and Bluetooth Module

|  |  |  |
| --- | --- | --- |
| Wi-Fi Module | Independent Module or Shield | Unit Price |
| MakerFocus | Independent Module | $ 12 |
| Keyestudio | Independent Module | $ 10 |
| WoWoone | Shield | $ 12 |

Table 13: Wi-Fi and Bluetooth Module

3.3 General Positioning

Positioning the sensors and unit itself in the most functional and effective way possible is key to the most accurate and effective readings. If the Sensors are in an odd fashion in the Smart Garden environment, accurate and constructive results are not guaranteed. The set-up process should take less than 10 Minutes of time maximum for guaranteed accuracy. As listed below, there are many important reasons for the specific positioning of these different sensors. For our guaranteed results, these sensor positioning techniques are vital. For accurate soil moisture readings, humidity and temperature readings, and ultraviolet radiation readings, these positioning techniques become something the developer and end-user cannot look over. Whether it is an all-in-one unit or portable unit, the position of the sensors will be what enacts the sensors to function as they were developed and designed to do such.

In the section below the design is detailed with a comprehensive breakdown of the different sensor positions along with other key components. Every electrical component has a strategically engineered position on the Smart Garden system to prevent any reliability and useability constraints. Moreover, with the developers placing the components on the board instead of the end-user, there is little to no chance for user error. The caveat to this is there is less of a modular component to the SMART Garden. However, the engineers prioritize functionality over a modular design. High-risk components like the Arduino board have been strategically placed in the most waterproof area of the SMART Garden to minimize the risk of critical water damage as much as possible. Due to this being an indoor unit, the positioning of the sensors was also important. The engineers wanted the sensors to be in universal positions where the readings would be as accurate and precise as possible from any orientation the SMART Garden may be in at that particular moment.

3.4 Possible Designs and Related Diagrams

Below is an “artist’s rendition” of the initial intended design of the finalized “consumer model” of the SMART Garden. The angled view in figure 3 shows the water reservoir and nutrient reservoir doors, where the respective fluids will be stored until the plant needs them. It also clearly shows where on the outside of the case the informational display will be, which displays the most recent readings of the sensors

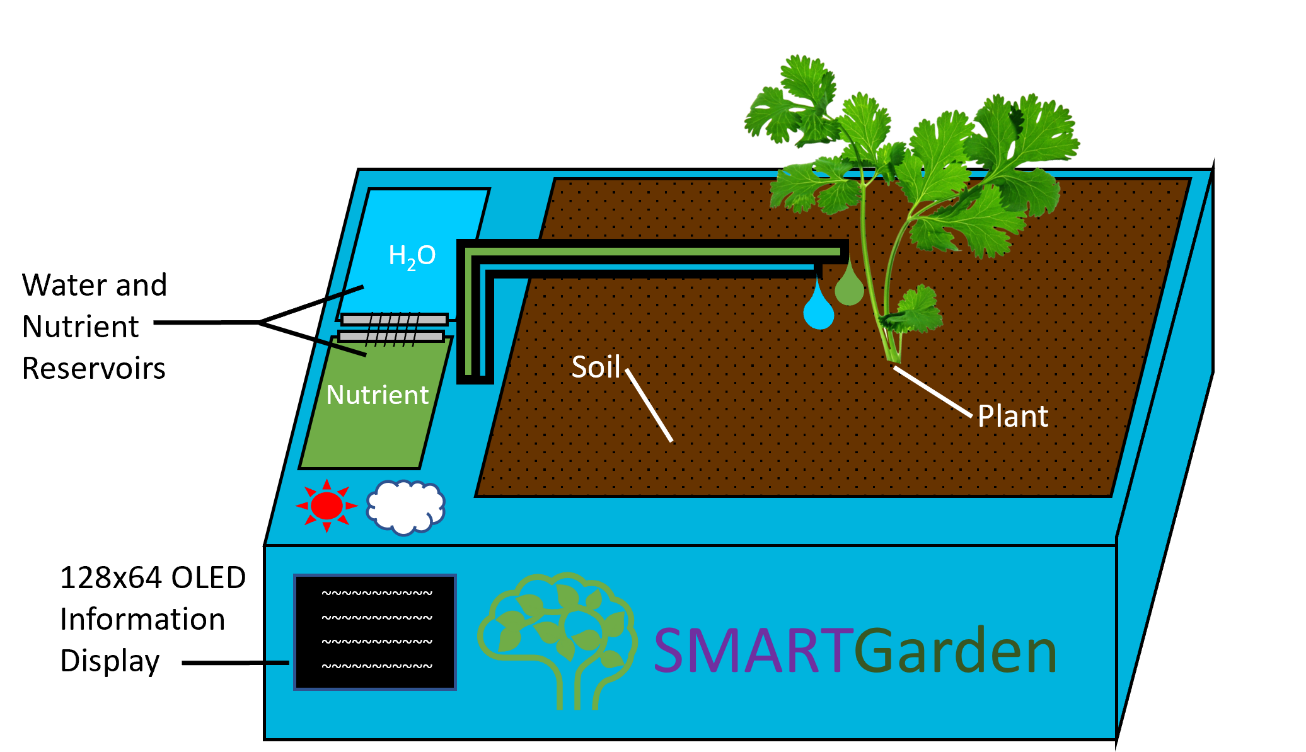
.

Figure 4: Angled view of SMART Garden

In the next image, figure 4, the top-down view is shown, with the plant and soil removed to show the location of the soil moisture sensor in a good place. If it were too far from the plant, it would not provide an accurate reading of the conditions the plant is experiencing. Additionally, the water and nutrient reservoirs are open, showing the fluid pumps inside, which will drive the water and nutrients through the water and nutrient tubes and to the plant when the garden determines it is needed. Above the location of the informational display are the UV Sensor and the Temperature and Humidity Sensors. These are less critical than the soil moisture sensor but can help the user to be aware of improper conditions and mitigate them as well as they are able. Lastly, the location of the actual Arduino Uno R3 microcontroller is shown. However, in the final iteration of this first generation prototype, the actual PCB and microprocessor is separated from the actual housing, in order to minimize the risk introduction of water to high risk components.

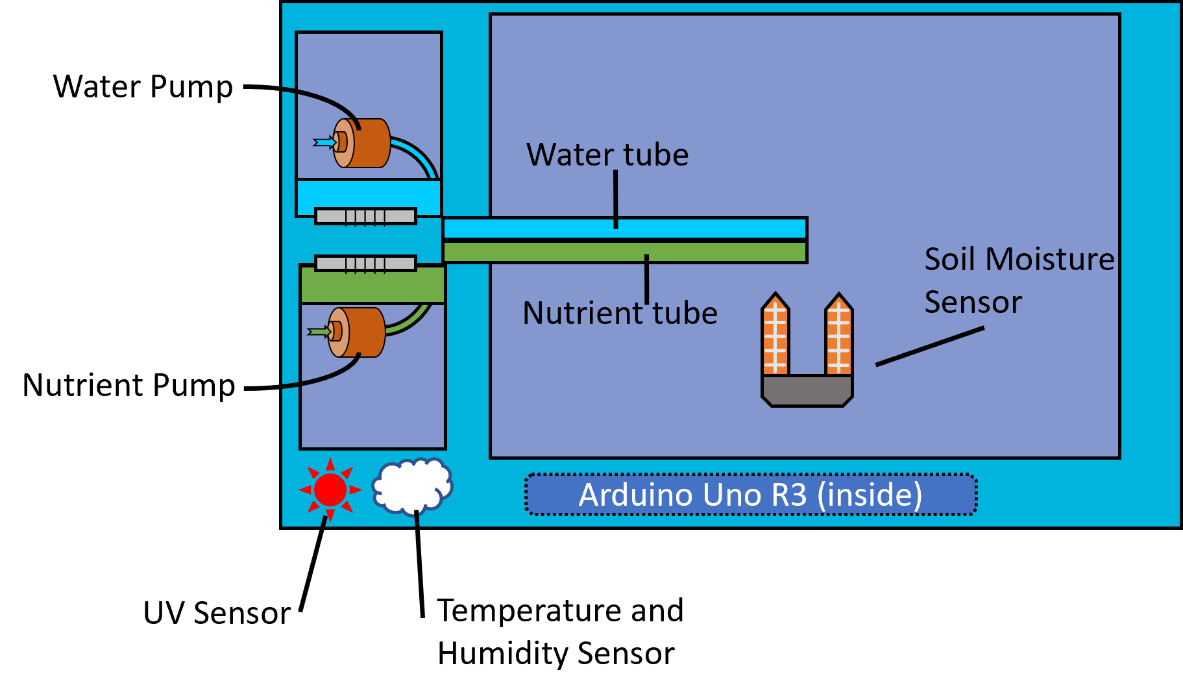


Figure 5 :Top-down view of SMART Garden

The intended final plan for the device includes the option to recharge the batteries with solar panels, but they are not necessary for the function of the garden and could even become an optional feature.

To reduce the bulkiness of the product as a whole, we would like to make sure that there is not much wasted space inside the casing. To this end, the electronics are largely grouped near one end of the device, where we can group together small electronics that won’t have much chance of causing issues due to proximity to other devices. This includes most of the sensors and the display. Another benefit to tightly grouping the majority of the components is that much of the wiring can be bound together rather than having many wires travelling wildly throughout the inside of the device. This idea is sometimes called “cable management” and can help prevent stray wires from damaging fragile components when the device is being moved. Since the SMART Garden is designed to be portable, as in easily moved, we don’t want to design a device that is likely to damage itself. Binding up the wires inside the casing is a simple way to prevent such a thing. The PCB and microcontroller will be postioned serperate from the housing itself in order to reduce the introduction of moisture and shorts to high-risk electrical components on the board. An adequate distance was included in the layout so that if any pieces must be replaced while testing, they will likely be the cheaper pieces, and the benefit carries over to the customer as well. If any part fails, it would be a less vital component, like a sensor or the display. While sensor function is important, no single sensor is as important as the device that interprets their readings.

3.4.1 Sensor Positioning

The positioning of the different sensors proves vital to the accurate and precise measurements the SMART Garden is expected to maintain. With poor positioning, the reliability of the system would be compromised. The SMART Garden would not have any smart features if not for accurate readings from its peripherals. As discussed in section 3.4, the general design of the system is in place. Small tweaks may be done between the first prototype and the final model, but the general outline and design has been modeled. However, the actual positioning of the peripherals on the device will be much more specific than the general design outline mentioned above. Each peripheral sensor will be covered below.

**Soil Moisture Sensor Positioning**

The most important sensors, the soil moisture reading sensors, have been studied and thought of where to be best placed. In figure 4, there is a general case outline of where the soil moisture sensor would be located. This generality just shows the sensor in the soil. However, with 3 sensors being implemented throughout the whole device, there will be sensors placed about every 3 inches to always maintain a consistent soil saturation. This will provide to be key in sustainment and growth of the garden plant using the system. The soil moisture sensors are the only sensors which need to be manually set by the end-user. The set-up time will take no more than a couple of minutes. Variation of about an inch shall not skew the SMART gardens accuracy. This simplifies set up, overall set-up time, and stress relief from the end-user. These are all key goals mentioned under the SMART Gardens introductory section. This information will be provided in the user manual to ensure proper accurate and precise results from the Smart Garden system. The reliability of the system should not be affected by the proper installation of these soil moisture sensor peripherals on the SMART Garden

**UV Light Sensor Positioning**

The UV light sensor peripheral is the second of the three main sensors packed onboard the SMART Garden. While not as critical as the soil moisture sensor and its positioning, the positioning of the UV light sensor is still a key point to make the end user aware of. The positioning of the UV light sensor was strategically placed near the lower left side of the unit for several key reasons. The two most important reasons are accuracy and reliability.

For the engineers to enhance the overall accuracy of the UV light sensor, the final position of this sensor was put in the lower left corner of the unit for a few different reasons. Notice that the placement of this is as close to the left bottom edge of the unit. This was intentionally engineered this way to provide the minimization of false cast shadows from the fauna residing within the SMART Garden. If placed closer to the center and depending on the directional positioning of the unit to the sun, there would be an increased chance that the plant within the system would block the true UV light reading from the sensor. This would defeat the purpose of the sensor. Therefore, the engineers designed this sensor to be as far from the plant as possible while maintaining a universal positioning standard. This means that the Smart Garden system does not need to be intentionally placed in one direction for the sensors to take proper readings. Instead, the unit itself can be oriented in any position which increases versatility and overall useability. This again maximizes the overreaching goals of the smart garden system.

In order to enhance the overall reliability of the UV light sensor in respect to the Smart Garden system, putting the peripheral sensor near the Arduino was a large goal for the engineers. Limiting the distance of wiring will never be a downfall for any robust system. This minimizes failure on many fronts. Having these minimizations in place will help the overall reliability of the SMART Garden for the end-user. Reliability is a huge concern for smart devices like the SMART Garden, even when operated indoors. Elements like water, nutrient mix, soil, and plant matter will all be within inches of this wiring and connection points. Therefore, strategic placing of the UV light sensor was a crucial design point for the engineers.

**Temperature and Humidity Sensor Positioning**

While it may be the least crucial positioned peripheral, the temperature and humidity sensor still plays an important role in the Smart Garden system. The reason behind the lesser significance in positioning is due to the unlikely variation in temperature or humidity no matter its positioning on the system. The only main requirement to increase the temperature and humidity sensor is for the actual peripheral to be placed somewhere on the top of the SMART Garden unit. It is highly unlikely that the temperature or humidity would vary to any level of significance like soil moisture may across the unit. With the temperature and humidity sensor placed on the top of the unit, the accuracy was maximized. However, the positioning of the sensor again can play a large role in the reliability of the unit. Please refer back to figure 4 to notice that the temperature and humidity sensor was placed just to the right of the UV light sensor peripheral.

The reasoning behind positioning the temperature and humidity to the right of the UV light sensor (bottom left of the Smart Garden Unit itself) is in order to maximize its reliability. Again, to enhance the overall reliability of the temperature and humidity sensor in respect to the Smart Garden system, putting the peripheral sensor near the Arduino was a large goal for the engineers. Limiting the distance of wiring will never be a downfall for any robust system. This minimizes failure on many fronts. The engineers ensured that minimization of risks was a priority when creating the detailed feature-set and design of the Smart Garden system.

3.4.2 Strategic OLED Display Positioning

The OLED information display was placed in a similar area to the UV light as well as the soil and moisture sensor peripherals in order to maintain the same overall goals as those sensors. A large factor for a small OLED information display is readability. Since the display is not as large as many easily viewable screens, the placement of this information display was greatly thought out. The engineers pushed to maximize this readability by placing the unit on the front side of the unit, near the left edge. With the display in this orientation, the display will not have a bright glare from sunlight and overhead lights. The other main factor for the strategic placement of this OLED information display is for the SMART Garden to maximize its reliability. With the unit's placement near other peripherals and sensors, there will be minimal travel between all peripherals and the Arduino unit itself. Limiting the distance of wiring minimizes failure on many fronts. The engineers ensured that minimization of risks was a priority when creating the detailed feature-set and design of the Smart Garden system. While not as critical in positioning when compared to the soil moisture sensor, the OLED information display was designed and placed in a position which maximized the useability and reliability of peripheral.

3.4.3 Water and Fertilizer Pumps Positioning

The positioning of the water and fertilizer pumps were the least critically positioned peripherals of the design. Put simply, the two pumps and reservoirs only needed to meet a few requirements. The first is reliability implementing the closed, close to Arduino methodology explained in previous sections of 3.4. The reservoirs were placed near the left side of the unit for this reason. While near the sensors, the reservoirs will maintain a level of water resistance in order to reduce any liquid related damage from water or nutrients. Another reason for its positioning to the left is to increase maneuverability for the end user when modifying the SMART Garden for any reason. Moreover, keeping the pumps on the top of the Smart Garden unit minimizes the likelihood of foreign material reaching the internals of the two pumps.

3.5 Power Module

With a device that needs to be lasting indoors, the method used to power the SMART Garden needs to be able to withstand the elements of everyday in food use just as well as the rest of the device. With appropriate weatherproofing, batteries would be a good solution to the issue of keeping the device running at all times. The inclusion of solar panels would aim to reduce the electricity or material cost that comes with the decision to power the garden with batteries.

3.5.1 Non-rechargeable Batteries

While simple non-rechargeable batteries would be a simple solution, this approach would be insufficient. Since one aim of the project is to help people become more self-sufficient, adding one more product to purchase would be rather antithetical to our intentions. This would theoretically work, but not to a degree we considered sufficient. Basic battery holders are a simple addition to a circuit or PCB, so it would be no problem to include one, but the purchases required to replace them once they have been drained of power would be unacceptable.

3.5.2 Rechargeable Batteries

With a rechargeable battery, the issues that are involved with purchasing new batteries are severely reduced. These batteries can be placed on a charger that plugs into a wall outlet in your home to replenish their charge, thus eliminating the need to buy new batteries every time the charge is depleted. Replacing the batteries is no more difficult with rechargeable batteries than with single-use batteries. If you want to buy multiple sets of batteries, you can make sure at least one set is always charged. Doing so would also minimize the downtime of the garden, since replacing the batteries would not rely on waiting for them to recharge, only as long as it takes to swap the batteries out for a charged set.

The major downsides of this approach are the time it takes to recharge the batteries, the electricity cost of recharging the batteries from your own electricity. The garden’s downtime while the batteries recharge should be reduced as much as possible to ensure the plants are taken care of with as little time as possible spent with the plants not in ideal environmental conditions. If the user were to purchase a second set of batteries, they could keep the second set charged in case the batteries in were to die. In this way, the user would be able to replace the discharged batteries quickly, and begin charging them immediately, thus minimizing the downtime. The electricity to charge the batteries needs to come from somewhere though, and if it is charged from the wall, it would cost money, and was likely made with a polluting method.

3.5.3 Solar Panels

Solar panels, AKA photovoltaic cells, can generate electricity using the light of the sun. While this is largely conditional, with no guarantee that there will be an ideal amount of light available to generate electricity, we believe that the possible benefits of having solar panels are enough to warrant their inclusion. With solar panels offsetting the power drain of the SMART Garden on the rechargeable batteries, each battery’s charge will last even longer. Depending on the solar panel used and the amount of sunlight available, it would theoretically be possible to have a net gain on the battery’s charge. In this case, removing the batteries would be unnecessary since they would never be drained fully.

Since the device is intended to be usable indoors, the functionality of these solar panels might be even more limited, but the prototype design will have them so that they can be integrated into the design for the users that decide to use them outdoors. Even indoors, there are likely to be times of day where the sun could reach the SMART Garden through a window if the user decides to attempt to maximize the utility of the solar panels. A solution for this could be to design the final consumer version of the garden to connect to the solar panels through long cables and allow the user to place them wherever they can get the best sunlight exposure, such as directly on the window, even if the garden is resting in the shade.

3.5.4 AC to DC Outlet Power

AC to DC power via a wall outlet provides ease of use in the sense of power and always-on functionality. However, it falls short when trying to maximize portability of the SMART Garden. Since the first generation final protype is an indoor garden, using a wall outlet is acceptable. Every room has these outlets which reduces the inconvenience of having to have it plugged in at all times. Using AC to DC outlet power also maximizes the ease of use when not needed to remember to recharge or replace batteries incrementally.

The Decision

The design of the SMART Garden includes AC wall power via an AC to DC wall plug. This decision was based on having a good union of power options to maintain ideal uptime for the garden. For the beginnings of the prototype stage of the project, 9-volt rechargeable batteries were purchased, since the Arduino Uno R3 has a recommended voltage range that some lower voltage batteries would not satisfy. Some portions of the board would function, but others would not be guaranteed to. Since we haven’t assembled anything yet, we couldn’t be sure those unusable features aren’t important, so leaving everything at full functionality was decided to be the most logical option. In the future, this decision would change, since the rechargeable 9-volt battery had a notably lower capacity than the rechargeable D batteries we previously considered. Since the final design relied on voltage regulators, and the Arduino development board was ‘replaced’ by the ATMEGA328P chip inside of it, it would no longer be viable to use batteries for a consistent voltage. Additionally, the garden’s focus shifted to be operating indoors, where electrical outlets would be easily accessible.

3.6 Communication Modules

Since this SMART Garden needs to connect with the internet, it required an apparatus to achieve this. The garden needs to communicate with the website in order to be configurable from anywhere, or else it would not be as convenient and customizable as desired. Unless each garden is given its own phone plan, the viable option remaining is to communicate with a secondary device to act as a middleman. The possible methods of doing this are to communicate with this secondary device using either Wi-Fi or Bluetooth. The primary factors to consider are data transfer rate, range, power consumption, implementation, and security.

Communications must be two-way, since the garden needs to both receive instructions from the web application and report the readings of the plant or plants it is taking care of to the website. This information is not too data-intensive, since it is largely simple strings of text, and the instructions required to interpret this data will be stored locally on either the SMART Garden’s PCB or the web server. Because of this, we will not need extremely reliable *high speed* data transfer as long as the connection is stable. The data transfer speeds of both Wi-Fi and Bluetooth should be more than sufficient for our needs, so that leaves primarily range, power consumption, and security as our primary specifications to consider.

3.6.1 Range

We want the SMART Garden to fit in with the average person’s life, not force the user to accommodate the garden with a setup that is inconvenient. Since the device will be located outdoors, the option with the highest range will be ideal. Bluetooth devices are typically only reliable and effective up to around 10 meters (33 feet) and are quite susceptible to being blocked by obstacles. Wi-Fi signals typically reach at least 20 meters (66 feet). Because of the expected location of the garden, the best option would be to communicate with the garden via Wi-Fi, since it is more likely to be within range of the Wi-Fi signal. If the user decides to keep the device outdoors, however, the longer range of Wi-Fi will be more important, since they could likely find a place within range of their wireless transmitter no matter which option was decided on. If, however, the home was larger, or had walls that were likely to block such a signal, Wi-Fi would still propagate more easily.

3.6.2 Power Consumption

Since the device was originally planned to be battery powered, the option with the lower power consumption would have been more ideal. Yes, the batteries can be recharged, but having to do so too often becomes a hassle and reduces the convenience the device affords the user. Bluetooth devices typically use between 0.01 and 100 milliwatts, with higher power having better range. The kind of Wi-Fi devices that are used on a PCB can typically use between 40 and 100 mW. Because the garden was battery powered, the ideal choice regarding power consumption would be Bluetooth communication. The final decision regarding powering the device renders this criteria more or less irrelevant on the scale of power consumed. This much power would almost be a rounding error compared to the garden itself, so it’s less important to save on power usage.

3.6.3 Security

Because the device is wireless, its security is a concern no matter how likely or unlikely strangers are to want to sabotage the SMART Garden. Bluetooth devices have a few ways to prevent attacks beyond requiring a password, such as requiring either a passcode or physical access to the device. Wi-Fi devices support the Wi-Fi Protected Access security methods, which were most recently updated in 2018. All new Wi-Fi devices must support the WPA3 to bear the “Wi-Fi CERTIFIED” logo.

The Decision

Given all the factors in play to consider, Bluetooth’s largest victory would be in the power consumption category. Given that the SMART Garden is meant to be easy to use, the benefits of having a device with a long range and good security outweigh the possible drawback of having to change the batteries more often. The SMART Garden will use a Wi-Fi module.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Data Rate | Range | Power Consumption | Security |
| Wi-Fi | 8 Mb/s (Arduino Uno) | 20 m | 0.01-100 mW | Key-matching |
| Bluetooth | 24 Mb/s | 10 m | 40–100 mW | WPA3 / WEP |

Table 14: Bluetooth and Wi-Fi Module

3.7 Printed Circuit Board

The printed circuit board is a collection of all the parts our group decided to include in the final product. The printed circuit board that was designed for this project ensured that all sensors, pumps and power supplies were routed to their correct destinations, or at least to easily accessible pins. This section will outline how printed circuit boards are made, what they are made of, how to understand the schematic of a printed circuit board, and some basic design recommendations for creating one.

3.7.1 Creating Printed Circuit Board Basics and Terminology

A printed circuit board is a circuit board onto which specific components have been placed with wired connections (traces) between them. The schematic of this circuit board is designed in a program like EAGLE Autodesk and is printed out onto the circuit board with a specialized machine. Traces can easily be automatically routed around and through the board to easily and effectively utilize the space. While it is possible to choose the routes yourself, it quickly becomes tedious and troublesome when dealing with too many components. Since the autoroute feature guarantees that you get the connections you want without overlapping in physical space, there is no realistic downside to using it, at least at the student level, not dealing with cost savings from mass production.

These schematics often include labels or markings near some of the components on them with meanings that might not be obvious at first glance. Some of them are provided and described below.

SMD – Surface-mount device – an electrical component is mounted directly onto the surface of the printed circuit board.

TH – Through hole – a hole drilled though the board with no connections going through. This is often intended to be where screws are used to mount the board on another surface.

VIA – a hole drilled through the board with connections going through it. This allows lines to pass through to the backside of the board directly, often to avoid overlapping other lines.

Teardrop Shape – either indicates the end of a line or is used to reduce the possible physical stress on a line so that it doesn’t get damaged by bending.

3.7.2 Composition of a Printed Circuit Board

The final product of the printed circuit board (PCB) is an electrical circuit with mechanically printed lines connecting precisely installed components to create a circuit more precise than humans could do by hand. This circuit will have many different types of components as created in the SMART Garden schematic. PCBs have one or two layers of copper foil that act as grounds for the circuit, with vias making sufficiently large holes in order to avoid contacting these layers unintentionally. Lines are copper paths between components that are laid out in paths that are precise enough to warrant using a machine made specifically for this purpose.

*3.3.3 Design Tips and Recommendations*

Gerber Labs has provided some suggestions of how best to design a PCB. Some of them are listed below, with an explanation of why they are helpful.

*Don’t Rush to Use the Autorouter*

While the autorouter tool is useful, it doesn’t move any components around your board, and thus might be an extremely inefficient path for your lines. Try to get the components where they make sense to be before using the autorouter tool.

*Know Your Manufacturer's Specifications*

Knowing trace width, amount of board layers, and spacing before finalizing the design can prevent errors that come with poor layout, and thus having to redesign the schematic to fit within these parameters.

*Component Placement*

The location of components affects the design of the PCB, whether there are sensors to consider or large components with many pin connections. Keeping components nearby other related components and being mindful about their orientation relative to each other can help simplify the lines that the autorouter tool places, leading to less clutter and in some cases, less unnecessary heat generated.

3.8 Fertilizer Remedies

There are a handful of different fertilizer options to choose from for agricultural needs. The climate and type of plant being grown are two major factors for the choice in most circumstances. However, after research it turns out that the engineers need a particular type of fertilizer for the SMART Garden pump system. Since this Smart Garden system will only be handling small plants (mostly herbs and small vegetables), the SMART Garden does not need a very specific fertilizer. Below are some of the fertilizers which were considered during the research and development process. The engineers decided to go with Miracle-Gro due to its simplicity, reliability and overall effectiveness. Moreover, it is very reliably in stock, which cannot be said about a lot of components of the SMART Garden during the fabrication process.

**Miracle-Gro Indoor Plant Food**

This is a best seller plant fertilizer and comes in at 75 cents per ounce of plant food. This plant fertilizer is for all indoor plants, including edible plants like the vegetables the engineers will be cultivating. This particular bottle comes in at 8-ounces (about 236.59 ml) and has a 4.6/5 review with 25,000 reviews total. In the description for the product, it notes you may mix with water or add directly to the soil. The engineers decided to mix it with water to allow the pump system to properly function. The description for this product also gives a general use formula for applying once a week. Obviously with different plants this will vary. This product is well in stock, and the engineers had no issue of obtaining a bottle of such plant food. This plant food/fertilizer is the perfect all-around option for the Smart Garden system.

**Miracle-Gro Pour and Feed Plant food (liquid)**

This version of Miracle-Gro is also readily available and another fan favorite. The general overview of this product is as follows:

* 19 cents per ounce
* 32-ounce (about 946.35 ml) bottle
* No mix formula, pour and store
* The general use guide says to use it about every 1-2 weeks for most small plants.
* Does not mention whether it is edible safe
* 4.6/5.0 with over 2000 ratings

**Schultz All Purpose 10-15-10 Plant Food Plus**

This product varies from the Miracle-Gro products listed above. This plant fertilizer is recommended to be used with every watering, about 7 drops of the mixture to be exact. The issue that arises with this is the ability to get a proper dosage every watering with a large reservoir. It was for this reason the engineers moved on from this fertilizer. Other favorable features of why this was considered are listed below.

* 5 stars with over 3,300 reviews
* Easy to use dropper
* Feeds the plant the way natural environments would
* More reliable than Miracle-Gro

**Lush House Plant Fertilizer**

This product is another liquid-based formula from a small business named Lush. They are environmentally friendly and this 500mL bottle can make up to 5 gallons of mixture to be used. The environmentally positive aspect was appealing for the engineers but due to cost this one was not part of the final consideration. This would most certainly be the choice in a commercial scale usage of the Smart Garden system.

**Easy Peasy Plants All Purpose Plant Food**

This all-purpose plant food comes in at 10 dollars for an 8-ounce bottle. Off the start, this is slightly less expensive than the Miracle-Gro plant food at 62 cents per ounce of formula. It is environmentally positive and has a formula which provides desirable results. However, there is no mention of edible safe formula, which seems necessary for small herbs and vegetables.

**The Decision**

Fertilizer plant foods are a quintessential part of the SMART Garden and its success. For these reasons, a fair amount of effort was put into the research of which plant food would yield the most positive results for the Smart Garden system. In the early stages of the design and system itself, it was thought that any fertilizer would suffice. However, after research and development the choice of which plant food to use became more important.

The Miracle-Gro Indoor Plant Food is an all-around stable and environmentally neutral choice with an ideal price point. The reviews show its reliability, and the price is no higher than other similar options. It is liquified enough from the reviews and there is the option to dilute it with more water to get the formula to the desired consistency. Moreover, it is edible and safe so the user will not have to question whether the curated plant is safe to consume. It is for these reasons that the engineers chose this Miracle-Gro product during the research and development phase of the SMART Garden.

3.9 Parts Selection Overview

Table 15 displays an overview of the parts selection for the project. All small parts were excluded from the table along with any non-technical items. Details on why each part was selected can be found in their respective sections.

|  |  |  |  |
| --- | --- | --- | --- |
| Item | Part Number | Manufacturer | Cost per Part ($) |
| Microcontroller | ‎A000066 | Arduino | 27.6 |
| Soil Moisture Sensor (5-pack) | 3-01-0313-A | HiLetgo | 7.89 |
| Temperature and Humidity Sensor | DH | Aideepen | 10.35 |
| LCD Display | 128x64 Yellow Blue SSD1306 (OLED screen) | UCTRONICS | 6.99 |
|  |  | **Total Cost** | 52.83 |

Table 15: Part overview

4. Related Standards and Realistic Design Constraints

Since the SMART Garden is developed with being a consumer product in mind, it is important to make sure it follows all regulations necessary to be such a product. Design standards must be met, and further, the device must be able to be assembled and function in the real world. All the optimism in the world won’t make a device work if it wasn’t designed with the idea of working within realistic limitations in mind.

4.1 Related Standards

Incorporating widespread standards was a great boon to the development and iteration of the SMART Garden. Not only did it ensure all the systems in place, such as the Wi-Fi module or the sensors, communicate with each other seamlessly, adhering to these standards also simplified the troubleshooting of any issues that were discovered throughout the development cycle. Since we could be sure that we were using as much technology as possible to the letter of the standards, there were fewer things that could be the source of our issues. That’s just the standards of implementation though, and material standards can help protect both the environment and the garden from harm.

In addition, any standards that apply to ease of use are simply a boon to the user. A product should be designed so that it is easy to use and maintain, rather than obtuse and complex. Though we ask the user to ensure the water reservoir remains filled, we want that filling process to be straightforward when it does need to happen.

4.2 Team’s Coding Standards

It was important that the programs produced when creating the software components of the project were standardized. Each language used in different aspects of the project had their own requirements for use, such as limitations of the technology, but they also have typical “best practices” used by the majority of the users for said programming language. In the same way, our team must abide by our own rules wherever possible.

These rules may not be specifically necessary to follow for the success of the project, but they made development easier for all involved. Each rule may or may not follow the widely accepted “best practices” for the application in question, but since the rule is deemed logical, there was no reason to deviate from the norm.

Some of these rules involved variable naming conventions, such as whether to name a variable something easy to remember, or something extremely descriptive. For example, we had to decide whether we should name the variable for the moisture sensor’s output something like ‘soilPin2’ or something like ‘HiLetgoLM393Output’. We decided to choose the first option, since it is easier to type and remember, but there are arguments to be made for choosing extremely descriptive names as well, since it is easier to know at a glance which device is the origin of the data stored by this variable.

It’s not just the content of the name that can be mistaken, we must also decide on a standard for the format of the name as well. We decided to use ‘camel case’ for variable names to denote where one word ends and another begins while keeping the name as one “word” for the programming language requirements. We also chose to have different standards for different contexts if so desired, such as using underscores in the database names to separate words while using camel case for programming the ATMEGA. Whatever decisions we made, it was important to stick to them to make sure everything worked correctly.

If the standards weren’t enforced, there would have inevitably been conflicting decisions made, leading to issues when the relevant variables are referenced incorrectly. This would also apply to “pointless” formatting, such as whitespace, and where on the line a curly bracket ( { ) falls. If the code was difficult to read quickly, the chances of a statement being placed in the wrong location are high. For example, something might be placed outside of a loop instead of inside it, leading to something that should occur regularly, such as checking a sensor value, instead never occurring, or only occurring once.

In most cases, it was best to separate the functions of the code we write as much as possible. This is referred to as modular coding, breaking up each part into the smallest viable pieces in order to reduce the chances of running into any issues with dependencies. Each function of the project should be its own section of the code. We didn’t want to run into issues with a small part of the code and end up unable to test what went wrong because that issue is causing other issues. This will also prevent making unnecessary API calls, since the applications of each part of the code are only retrieving what is needed. It wouldn’t be helpful to retrieve the data from all the sensors when we only want to check the level of humidity. While we ended up polling all the sensors at once regularly anyway, this practice separated any possible failures, preventing one failure from causing a domino effect and leading to more.

4.2.1 Coding Language requirements

While we had a certain amount of freedom when it came to deciding what kind of standards, we choose to enforce on ourselves when it comes to things like variable names and whitespace layout, there are certain things each aspect of our project required when it came to the programming language needed. Each language had different built-in standards that cannot be expanded in any way. That’s just how the program is, and that cannot be changed.

Data types have restrictions on the kind of data you can store in them, some languages have functions (in the programming sense) that other languages don’t have, and some languages even have different ways to write conditional comparisons. Adapting to writing in each of the programming languages involved in the project was a challenge but was one we were able to overcome. Expectations were occasionally subverted when we realized that what we wanted to happen will take many lines of code in the environment in question when it would be a single command in another. Adapting to these differences was a challenge we rose to, showing that we were able to do what is necessary to achieve our goals and use the tools at our disposal as necessary to do so.

4.3 Realistic Design Constraints

When you undertake a task you care greatly about, you can’t help but have high hopes for it. Sometimes those hopes may come true and sometimes they were always beyond reach. The ideal outcome in any scenario is often born from unrealistic expectations of what might happen. The same is true for this SMART Garden. We wanted it to be as good as it could be, but the final product didn’t look like it did in our heads the day we agreed to undertake this challenge as our senior design project.

That’s not a failure though, sometimes the way things work out, those dreams can’t come true for a very good reason. Whether those limitations are because it is impossible, unsafe, or just too impractical, there was a reason all along. Not being able to implement your exact idea can also inspire you to achieve the closest equivalent to it from another method, leading to creativity through limitations!

Other times, however, these constraints lead to only frustration and stress from failure or inability to overcome them. If you can’t afford something crucial to a design and definitely end up with a shoddy product because of it, there is no way to overcome that constraint without having the privilege to not have to worry about the cost of things. Project constraints can be a double-edged sword.

4.3.1 Economic Constraints

Economic constraints- the near-universal annoyance of something just costing too much money. When measuring something with a sensor, you rarely have to be very exact, and close enough is often plenty. It’s been said that only 40 digits of pi are needed to measure the size of the observable universe, and pi is infinite. Just think how many corners are perfectly acceptable to cut here on Earth. You rarely need the perfect tool for the job, just one that fits well enough. However, sometimes, the only tool that fits well enough is itself expensive, and you then need to cut back in other ways. If an alternative can be found that functions just as well, there’s little reason not to use the cheaper version. That way, if you need to replace it for any reason, you can get its replacement for cheaper as well. This can be extended to companies mass-producing a product as well. A small saving multiple times can add up to large savings eventually. This is all to say that if we have alternatives of sufficient quality, there’s no good reason to overspend on the components that only slightly improve the performance.

On the other hand, it could theoretically be true that a more expensive option can afford you other benefits that would be considered well worth the price. For example, if a purchase makes it easier for you to perform other tasks successfully, it might help save time and effort with enough secondary benefits. At that point, you have stretched one constraint to reduce the severity of another- namely the time constraint.

For this project, we might have needed to cut back on the case for the parts if we ended up needing to pay too much for the printed circuit board, or if enough parts ended up being defective and needed replacing. Strictly speaking, the only thing that would affect the function of the SMART Garden is if the basin for the plants was too small to allow for sufficient growth. Otherwise, even if it somehow had poor water retention, the garden would simply water the plants more often to offset the loss of water. Otherwise, the placement of the screen, and to some extent, the sensors, would have been only about convenience and aesthetics. It would work almost as well as a jumble of parts sitting on the table as it would in a neat plastic box. Sometimes that’s the name of the game with economic constraints- close enough.

4.3.2 Time Constraints

When your economic constraints are dealt with and you finally have all the parts, you might realize that your lofty aspirations for the project were too great, and that actually implementing the design will be too difficult to complete in a timely manner. If you won’t be able to get the design to work how you wanted, you may need a new design. Getting something simpler to work well is much better than failing completely to get a more complex design to function. These time constraints can come from anywhere, from having to troubleshoot a simple mistake that you don’t notice for long enough to delay the progress, to the previously mentioned unrealistic expectations. One common time constraint that is faced by students in group project courses is the inherently limited schedule that they have due to being students. Many groups only have a short time per week to meet and work on the project, others have long commutes and can’t meet as easily at short notice if something needs the group’s attention. Some even have jobs that they have to go to on varying schedules, making even their limited time unreliable.

With limited time to work per week, it became important to our group to meet at the agreed upon times every week, and to have virtual meetings as needed. Being in person can’t be easily replaced, but if something needs urgent attention, sometimes a timely response is needed rather than delaying to have a perfect response. It is good to get any individual tasks done as early as possible so that no group members are stuck waiting for your contribution in order to be able to continue with their own other tasks. It’s important not to cause your own time constraints whenever they are avoidable- procrastination is a terrible habit to get used to. Working early lets you notice any issues that arise with time to spare rather than being in a state of urgency where any slight delay throws your carefully laid plans into disarray.

4.3.3 Environmental Constraints

Due to the nature of the SMART Garden, we had to consider what materials, emissions, and pollution any design we come up with could possibly produce. This is one upside to using a wall-outlet adapter rather than batteries, less waste material is being produced, at least in the form of solid physical waste. The electricity that powers the garden does have to come from somewhere, which would ideally be green energy. The materials used in its construction will also need to be environmentally friendly- it just wouldn’t do if the materials holding the soil were leeching harmful chemicals into the soil that the plants you are growing for food are themselves feeding on. This would be a very localized environmental concern, but a concern, nonetheless. The fact that the device is largely isolated goes a long way to prevent unintentionally affecting the environment on a larger scale, but the possibility of having a negative effect should never be dismissed outright. The spread of invasive plants is possible if the user plants one and places the SMART Garden outside so it can get plenty of sunlight. The plant, now outside, would be able to spread to nearby soil and then truly be in ‘the environment’ rather than in a container. Though the garden is designed to be used indoors, as long as it is powered and close enough to a Wi-Fi connection, it will work when placed outside as long as the power cord is protected from the weather. Though an IP68 rated adapter would be suitable for outdoor use, we tested the garden without an adapter with this safety rating.

4.3.4 Social and Political Constraints

This project shows support for many movements that advocate for helping the environment, since people would be able to grow a little piece of it in their own homes. They might be more willing to support saving the environment in the places once they can experience for themselves the joy of growing something for yourself. At the same time, we would be empowering individuals to more actively provide for themselves, no matter how little a difference the plants the SMART Garden can grow will make. Sometimes, a little victory like that can go a long way for a person’s self-esteem.

4.3.5 Ethical Constraints

There are no sufficiently likely cases that come to mind where this project could negatively impact somebody, something, or their livelihood. Since the goal of the garden is to improve people’s lives, the only way anything unethical could happen is if the device fails to function correctly. Though our intent is to aid those struggling to provide for themselves, if the garden fails due to a design oversight, that will be hypothetical profit for ourselves to the economic detriment of a hapless customer. It will be our ethical duty to ensure the SMART Garden works correctly so as to minimize the risk that someone who is already having issues obtaining healthy food wastes their time, money, and effort trying to grow food, only for our group to fail them.

4.3.6 Health and Safety Constraints

Because we are dealing with edible things, we need to be sure we don’t introduce some dangerous factor to the user unintentionally. For example, we need to make sure we make the water reservoir accessible and cleanable so that no mold or bacteria gets in and begins to grow. This could easily get onto the plants during the watering process and infect the person who consumes the plant. Otherwise, the materials of the device should be completely safe for both the environment and for human contact. If it degrades easily, the materials in the soil basin could be absorbed by the plants, and if eaten, those plants would be dangerous. We also made be sure that the device is physically safe. We can’t have excessively powerful water pumps or sharp edges, and we must ensure there is no possibility for somebody to receive an electric shock from the device’s power source or circuitry.

4.3.7 Manufacturability and Sustainability constraints

The final version of the SMART Garden is meant to be moved with relative ease, so there will only be a moderate amount of setup once the hypothetical customer acquires their garden, and none of it should be assembly. This setup would involve powering on the device, filling the water and plant food reservoirs, filling the main basin with soil, planting the variety of plant of your choice, and configuring the device via our website. If they desire to, they can create a custom plant entry as well. With the customer’s ‘manufacture’ steps minimized, the manufacturability issues fall to us assembling the prototype for the first time. The problems we could expect involve the many interconnecting devices within the garden. A problem with the water pump could mean the pump is failing, the signal to water isn’t getting sent by the code, or the signal was disrupted along the way by a faulty wire. This kind of issue could occur with any part involved, including the PCB. Even though you can test it, freak accidents (or even nothing expected) can cause a device to fail.

Getting these systems to interact perfectly was a tall order, especially for a prototype with so many devices we had never used before. Not to mention getting the ATMEGA to communicate with the server in a state where viewing the outputs could only be done indirectly. Apart from the pumps, there are no moving parts in our project, so at the very least, we had little to worry about in the way of alignment or calibration. Once our device is assembled and in place, there should be nothing to cause it to fail, apart from random unpredictable errors like pump failure. Assuming we have waterproofed the garden sufficiently and haven’t built the PCB or circuitry such that heat will accumulate enough to cause damage, no changes to the device are expected aside from it being unplugged.

As for the sustainability there are a few things that could affect the usability of the device in ways other than mechanical or electrical failure. This is focused on SMART Gardens that are located outdoors, but could theoretically apply to those inside as well. Since the garden is designed to specifically support plants, and not manufacture entire environments to fully sustain them, local climate could prove problematic if the temperature or other conditions are extreme enough.

We aim to solve any lack of water by automatically watering the plants when needed, but if the temperature is dangerous to the plant, the current iteration of the SMART Garden has no way to combat that, unless watering happens to be enough to (extremely locally) lower the temperature, but that seems unlikely. If the rain in an area is sufficiently high, the plants might have problems as well. Some plants can have problems if they get too much water, not just too little. Since the garden doesn’t have a way to actively reduce the amount of water in the soil, that would be a problem for the user to solve or prevent. The user should be able to rely only on the garden to water their plants. If the garden is functioning correctly, any additional water added by the user to the soil would be overwatering.

4.3.8 Problem Solving

As has been stated in a few ways there are often problems with no perfect solution. No material is without some cost, and no design change will eliminate all relevant problems without causing more in one way or another. That said, it would be foolish not to consider any solutions and simply become complacent with the issues that are present with our project. There is no guarantee that any of these solutions will be accepted as viable additions to the design of the project, but considering the benefits and drawbacks of some ideas is a crucial part of the design process. In the table below, some problems are presented that had a chance of causing either us, the developers and manufacturers of the product, or the user some amount of annoyance.

|  |  |  |  |
| --- | --- | --- | --- |
| Issue that can be mitigated | Possible Solutions | Pros | Cons |
| Expensive parts | Create the parts by hand for the prototype. | Cheaper parts that will likely work just as well. | Limited to only a few of the parts, such as the basin the plant will sit in, or the case that holds the sensors. |
| Bugs | Mesh covering around the garden | Mesh keeps bugs away from the plants in the garden. | Mesh might reduce the amount of sunlight the plant receives to unhealthy levels. |
| Animals/pets | Plastic bars surrounding the plants | Would keep some animals away from the plants. | Would not keep bugs away. Would not be very sturdy if a larger animal wanted to break in. |

Table 16: Problem Mitigation and Solution

|  |  |  |  |
| --- | --- | --- | --- |
| Issue that can be mitigated | Possible Solutions | Pros | Cons |
| Need food-safe materials for the soil and liquids. | Make sure to buy plastics that don’t degrade via UV, or use another material altogether. | Ensures that the plants grown in the SMART Garden are food safe by default, and will remain durable and functional for longer. | More expensive materials. |
| Possibility of invasive species on plants grown outside of natural habitat. | Mesh covering around the garden | Mesh would prevent larger seeds from spreading and would keep bugs away from the plant as well (pollination). | Mesh might reduce the amount of sunlight the plant receives to unhealthy levels. |
| Physical degradation of the device’s materials | Make sure to use materials that won’t degrade via UV exposure. | This material would ensure higher durability and longevity for the product as a whole | Some materials don’t have viable alternatives. Any that do are likely more expensive. |

Table 17: Problem Mitigation and Solution Continued

4.4 Arduino Communication Protocols

On the Arduino development board, there are three main communication protocols. The first is the Serial Peripheral Interface, or SPI as it is commonly referred to as. The next is the Inter-Integrated Communication, or I2C for short. Finally, the last communication protocol for the Arduino microprocessor is UART which is the Universal asynchronous receiver-transmitter. All three of these communication protocols are common among most modern microprocessors. However, the implementation in Arduino microprocessors is much more effective and straight forward. This provides maximization in efficiency and allows effort to be targeted towards bigger issues topics. These three communication protocols will be detailed in the following three sub-sections below. The engineers have detailed the Serial Peripheral Interface, the Inter-Integrated Communication, and the Universal Asynchronous Receiver-Transmitter below.

When choosing a communication protocol, there are a number of factors to account for. Whether the protocol is synchronous, how many wires are involved in the communication and how the pins are set. The detailed communication protocols below will focus on their specific features and how they act with the peripherals connected. SPI, I2C and UART all have their respective advantages and disadvantages, especially when comparing them to be used in the SMART Garden.

4.4.1 Serial Peripheral Interface (SPI)

The Serial Peripheral Interface, or SPI, is a very common interface used between peripheral circuitry and the microprocessor communicating to it. Forms of peripheral circuitry can include units such as displays and sensors, both of which are included in the Smart Garden system.Serial Peripheral Interfaces are synchronous compared to some other interface systems. Moreover, they inherit a master-slave based full duplex interface. Incoming Data from the master or slave can be synchronized by way of the falling or rising edge of a clock.

Typically, SPI is a four-wire design, especially in today's industry. However, SPI can also be a 3-wire design. For the Smart Garden system, a four-wire design would be used. In this design, there are four different signals to distinguish. The first signal is the clock (SCLK, SPI, CLK). The second is Chip Select (CS). The final two are MOSI (Master out, Slave in) and MISO (Master in, Slave out). The Arduino Uno R3 documentation refers to MISO as CIPO (Controller in, Peripheral Out) and MOSI and (Controller Out, Peripheral In). The Smart Garden system will inherit this verbiage to baseline with the microprocessor. COPI is the controller line, which sends data to the peripherals. On the other hand, CIPO is the peripheral line, which sends data to the controller respectively. The Serial Clock, or SCK, is the clock pulses which synchronize with the data transmission generated by the Arduino. Moreover, the Chip Select pin, or CS, is on every device. The CS can use two modes, enable and disable, to avoid faulty transmissions occurring due to line noise.

SPI is not as clear cut of a standard as other communication interfaces. However, the Arduino has all SPI settings determined by a register on board named the SPI control register (SPCR). The SPCR is a data register referred to as the SPDR. Data registers are present to hold different Bytes of information. The SPDR holds the byte which is about to be shifted from the COPI line, or when the data had just been shifted into the CIPO line. It is also worth noting that in SPI communication interfaces, there may only be a single controller device connected to other peripheral devices.

The other two types of registers are Control and Status registers. Control Registers simply have code controls which help perform different Arduino functionalities. In the Arduino, each bit in the byte helps function a particular setting. The other type of register, the status register, converts from one state to another based off different microprocessor conditions taking place at that time. The SPCR, the control register on the Arduino has 1 byte (8 bits), each which controls a different and specific Serial Peripheral Interface setting.

The 8 bits in the SPCR are mapped to the following:

* + Bit 0 - SPR0
  + Bit 1 - SPR1
  + Bit 2 - CPHA
  + Bit 3 - CPOL
  + Bit 4 - MSTR
  + Bit 5 - DORD
  + Bit 6 - SPE
  + Bit 7 – SPIE

|  |  |
| --- | --- |
| SPR0 | Sets speed for the SPI – 00 is the fastest at around 4 MHz |
| SPR1 | Sets speed for the SPI – 11 is the slowest at around 250 KHz |
| CPHA | Samples the data on falling edge at 1, and leading edge at 0 |
| CPOL | Sets clock to idle when high at 1 and idle when low at 0 |
| MSTR | Sets to controller mode at 1, and to peripheral mode at 0 |
| DORD | Sends data LSB first when at 1, and MSB first when at 0 |
| SPE | Bit which enables the SPI when at 1 |
| SPIE | Bit which enables the SPI interrupt when at 1 |

This requires the engineer to be sure of what settings they wish to take advantage of. The engineer needs to set aspects like the data shifts and clock idle to the correct settings to obtain the proper functionality of the microprocessor. After the SPI control register is properly set to the users' expectations, the developer must figure out how long they need to idle between different instructions.

4.4.2 Inter-Integrated Circuit (I2C) - 2 Wire System

The inter-Integrated Circuit protocol, I2C, uses two lines to send and receive data while communicating. This is a huge reason why I2C is so widely used and popular in the industry. On the Arduino itself, there are two different connection pinouts for I2C communications. This makes the I2C favorable off the bat for the engineers due to the expansion factor alone. Below is a detailed breakdown of I2C with the Arduino microprocessor.

There is a serial data pin (SDA) and a serial clock pin (SCL). The serial clock pin is a clock which sends a pulse on a predetermined schedule. The serial data pin is the pin which actually sends the data back and forth over the two devices. When the clock moves from rising edge and falling edge, a bit of information will be transferred from the microprocessor to the I2C device connected over the serial data path (SDA). This data can also be sent back to the microprocessor if required for such activities.

The I2C 2 wire system is much more standardized when comparing it to other interfaces such as the Serial Peripheral Interface. Another very advantageous part of the I2C interface is the minimization of wiring. In theory, since I2C allows for every device connected to have its own address and the controller and peripheral take reciprocating turns with using the SDA – the engineer could have a single Arduino connect to *n* peripherals while only using two pins of the Arduino, where *n* is the number of peripherals connected to the Arduino microprocessor. The number of devices is limited to a maximum of 127 devices total, since the I2C addresses are 7-bit number, and the eighth bit is used to see if the signal is being sent by the controller or the peripheral.

Other factors that contribute to I2C’s wide range of popularity is its synchronous, multi-peripheral layout. Like SPI, the I2C communication interface is a synchronous communication. This is different from the universal asynchronous receiver transmitter, which as described in the name is not synchronous. Additionally, engineers have the ability to buy I2C hubs, where you connect upwards of 6 peripherals to a single up which daisy chains to one of the two SPI pinouts. This extends the reach of devices with I2C, even though this may not necessarily be advantageous for the Smart Garden system.

To begin writing to an I2C bus, there are some steps to set up first. To begin, the developer must drop the Serial Data line to zero volts. At this point, the developer must then make sure this is the case when the SCL voltage remains high. The first set of data can then be sent through the bus. At the end of this data, there is a bit that lets the device know whether the data sent is a read or write. That being said, between every 8-bit transmission of data, there is a pulse that goes through that acknowledges a pause on the serial data line called ‘ack’. That is considering that the controller is sending to the peripheral. A similar event occurs when a peripheral is sent to the controller. When this happens, there's another acknowledgment pause, called ‘nak’. This lets the peripheral end the transmission. There are many similarities between the Inter-Integrated Circuit and the Serial Peripheral Interface, as the reader can see. They both provide communications between the peripheral device and the controller. Although, the wire layout and setup are different, the similarities in communications between the controller and peripherals remain. However, the I2C commination interface allows for a multi-controller system. While this provides many advantages at times, the SMART Garden will only have a single main controller – the Arduino Uno R3. As read above, the SPI communication interface only allows for a single main controller, which in the case of the Smart Garden system is adequate.

4.4.3 Universal Asynchronous Receiver-Transmitter (UART)

The last communication protocol that the Arduino Uno R3 has on chip is the Universal Asynchronous Receiver-Transmitter, or UART for short. The UART is not synchronous, like the other two communication protocols that are on board the Arduino. UART is similar to that of I2C in the way of its Standardization. Before the UART can send data to another device, the transmitting controller much convert whatever data that needs to be sent from bytes to bits. After this conversion has taken place, the UART controller may then split these bits into separate packets of data for transmission to the other device. At the beginning of each packet of data, there is a start bit, up to two stop bits, the data frame, and possibly parity bits. Please see the table below for a visual representation of this packet found when transmitting data over the Universal Asynchronous Receiver-Transmitter.

**Packet of Data**

|  |  |  |  |
| --- | --- | --- | --- |
| Start bit  1 bit | DATA bits  5 to 9 bits | Parity bits  0 or 1 bits | Stop bits  1 or 2 bits |

Table 18 **The Data Frame**

There are a handful of parameters that the engineer must take into account when using the UART communication protocol. Parameters such as the Baud Rate, Data length, parity bit, number of stop bits, and control flow must all be considered when setting up the Arduino with a UART interface. Below every parameter is broken out into detail on what they respectively effect in the system.

**Baud Rate**

Simply put, Baud rate is the number of bits per second, or bps, a UART device can receive or transmit. It is essential to have two devices to have the same baud rate, or no more than a 10 percent difference. If they are not the same rate by more than 10 percent, this data may be lost and thus never received.

**Parity Bit**

The parity bit is the bit in the table above that is added to the data packet which tells the receiver if the number of ones in the data transmitted is odd or even. There are two setting states the parity bit may be. If the parity bit happens to be **zero**, then this means the number of *ones in the data frame were even*. Its inverse also applies. If the parity bit is **one**, *this means that the number of ones in the data frame were odd.*

**Number of Stop Bits**

There are three possible options for the number of stop bits in the packet. The Arduino UART can have zero, one, or two stop bits. The stop bits mark the end of a data packet that was transmitted through the UART bus.

**Data Length**

This simply is the number of bits per byte of fata being transmitted or received by the UART.

**Flow Control**

The Flow Control is how the UART prevents data loss during transmission of data of the Arduino UART.

On the Arduino microprocessor, there are a few different pins we will want to keep track of when talking about the *Universal Asynchronous Receiver-Transmitter*. There are two datalines used to communicate with each other which include:

* *TX* – This is used for transmitting information and data
* *RX* – This is used for receiving data and information

There are a handful of advantages to using the Universal Asynchronous Receiver-Transmitter on the Arduino. Some of the advantages include:

* The UART is very versatile on boards, and can be found on most microprocessors, especially Arduino boards
* Standardized process and pinouts
* Cheaper component than other communication interfaces
* Simple to operate and use with the Arduino – there are lots of examples that help engineers and developers work through many different problems they may encounter.
* No clock is needed due to its asynchronous manner

There are also a few key disadvantages for using the Universal Asynchronous Receiver-Transmitter on the Arduino. Some of these disadvantages include the following:

* Much lower speed than other interfaces (i.e. SPI and I2C)
* Cannot achieve a multi-Controller system like I2C has the ability of
* Baud rates affect data loss if not carefully set. Both Baud rates must lie under 11% of a difference or there may be data loss

4.4.4 Decision on the Communication Protocol

While each of the mentioned communication protocols and interfaces all have their advantages and disadvantages for the Smart Garden system being constructed, one proves to be more advantageous than the rest. The I2C protocol and SPI interface fit the SMART Garden the best, and UART falls slightly behind. The simplicity of the UART was appealing to the developers but in the end the UART communication protocol was not able to keep up with the others. While a multi-controller system is not required for this system, the ability to quickly get accurate results was. This made the engineers stay away from the UART for the SMART Garden.

The two that this SMART Garden came down to be the Serial Peripheral Interface and the Inter-Integrated Communication protocols. The engineers had put a lot of thought before deciding that I2C is the main communication protocol moving forward with the Smart Garden System. While the SPI interface had many advantages, the I2C protocol proved more reliable, simplistic and standardized. These three key factors are what allowed the engineers to confidently chose I2C as the main form of communication protocol for this Smart Garden system. This is not to say a situation may arise where SPI is taken advantage of, but the engineers will focus on applying I2C where applicable.

5 Hardware Design

5.1 PCB Design

In this section we will discuss the PCB design for the project. First, we will go over the software used to create the PCB followed by an overview of our PCB design. In summary, in this section we will discuss and analyze our PCB design layout as well as go over all manufacturing options to include a final summary of their use.

5.1.1 Software Used

In order to design the PCB that was required for our project we had to choose a PCB designing software. In this section, we will be discussing several different software options that were available to us, as well as which software(s) were utilized during our actual design and build. The top four software programs we considered for our project were Autodesk Eagle version 9.6.2, Autodesk Fusion 360, Fritzing, and EasyEDA.

**Autodesk Eagle 9.6.2**

Autodesk Eagle 9.6.2 is an Electronic Structural Automation (EDA) software. This software allows developers quick access to schematic editing tools and a wealth of library content. In addition to that, Autodesk Eagle 9.6.2 comes equipped with a SPICE simulator allowing the user to test ideas and measure circuit performance. Other notable features this program offers are real-time design synchronization, electronic rule checking, 3D PCB models, and modular design blocks. Eagle software comes with one of the most robust libraries of any PCB designing software and is compatible with Linux, Windows, and Mac operating systems.

**Autodesk Fusion 360**

### Autodesk Fusion 360 offers the user access to more than just schematic editing tools. This program connects all of your product development processes into one singular cloud-based software. Fusion 360 combines CAD, CAM, CAE, and PCB designing software tools. This product is not free to the general population although free access is offered to students attending university through an educational license. Notable features of this software include but are not limited to public/private design sharing, 3D design and modeling platform, PCB design integration, comprehensive library content, SPICE simulation, design rendering, and design animation. This software is compatible with Windows 8.1, Windows 10, and select Mac operating systems.

### **Fritzing**

Fritzing is a low-cost opensource electronic design software tool made to help designers translate prototypes into real products. The software offers three different circuit building options: Breadboard view, Schematic view, and Printed Circuit Board view. The latter two are offered in most other software programs, although what makes Fritzing unique is its addition of the breadboard view. The breadboard view shows you what the physical components look like and allows the user to construct circuits similar to how they would in real life, from there the software will convert the breadboard view into a printed circuit board. The drawbacks to this software are it does not offer as many readily available libraries as other competing products such as Autodesk Eagle and Fusion 360. In addition to that, Fritzing does not offer the ability to simulate and test circuit performance. This software is compatible with Windows XP and higher, Mac OSX 10.7 and higher, as well as Linux.

**EasyEDA**

EasyEDA is a free, zero-installation Electronic Structural Automation (EDA) software. The software gives developers access to more than 1,000,000 free libraries, offers short tutorials, and has an integrated LCSC component catalog giving the user pricing and other details for more than 200,000 components. The program allows you to work on library design and management, PCB design, schematic capture, project management and team collaboration. EasyEDA has a simple and functional user interface making it easy for beginners or those with little experience. Similar to some other PCB design software's mentioned, it offers SPICE simulation as well as auto routing options. The software can be used online as well as on a PC and is compatible with Mac, Windows, and Linux operating systems.

**Decision**

After reviewing the features and drawbacks of each PCB designing software our group made the decision to utilize a combination of Autodesk Eagle and Autodesk Fusion 360. The reason for this decision is multifaceted. For starters, Autodesk products are widely used and offer a large community of users. In addition to that, these software's offered some of the most extensive library selections available, along with a wealth of excess features that we used in the implementation and building phase of our project design. Lastly, the main reason Autodesk Eagle and Fusion 360 were chosen is due to the fact every member within our group has prior experience working with the product. This increased level of comfortability helped to ensure the hardware design could be created or altered in the most efficient manner possible.

5.1.2 Schematic

The purpose of this section is to discuss and analyze our initial PCB board design schematic. This schematic contains the wired connections for all on-board components to the microcontroller and voltage regulator circuits.

The following components being utilized in the schematic below: LM2576XX voltage regulator, ESP 8266 Wi-Fi module, ATMega328p microcontroller,100uH inductor,100uF capacitor,1000uF capacitor, 22pF capacitor, 100nF capacitor, 1MOhm resistor, RASM722X power jack, and the SSQ-118-03-F-S 18 socket pin header. All other components, to include the soil moisture sensors, the temperature and humidity sensor, and the 2-channel relay are connected to the PCB board via the on-board pin headers.

Diagram, schematic

Description automatically generated

Figure 6: PCB Schematic

In this portion of the schematic section, we will go over the thought process behind all pin connections between the microcontroller and auxiliary components.

**LM2576XX**

The LM2576SX-3.3/NOPB and LM2576SX-5/NOPB voltage regulators were used in our circuit design. These voltage regulators do not require a heatsink and generate a regulated output voltage as well as a guaranteed 3A output current. Each voltage regulator is accompanied by a 100uH inductor rated for 8.8A current, a diode rated at 4A current for 40V, and 2 capacitors. The first capacitor, valued at 100uF, is connected to the input pin on the voltage regulator and grounded at its negative terminal. The second capacitor, valued at 1000uF, is connected to the output pin on the voltage regulator and grounded at its negative terminal. The output of each circuit was routed to individual pins located on the pin headers to be used to power components that require an external connection. The only direct on-board connections made from the output of the LM2576 voltage regulator circuits were those used to power the Wi-Fi module and the ATMega328p microcontroller.

**ATMEGA328p**

The ATMega328p is composed of 28 pins. The pin mapping relationship to the Arduino Uno R3 can be found in Figure 7. Pin 7 on the ATMega328p is the VCC input pin and is routed directly to the output of the LM2576 5V regulator circuit. The reset pin on the ATMega328p is connected to one of the pins on the on-board PCB pin headers. In addition, a 10k pull-up resistor is attached starting at the wire trace exiting the reset pin and connects to the wire trace that is routed from pin 7 to the output of the 5V regulator circuit. In addition, there is a 100nF decoupling capacitor routed from AVCC pin 20 and VCC pin 7 to GND pin 8 and AGND pin 22 for noise reduction associated with the input voltage and other surrounding components. Pins 9 and 10 on the ATMega328p are reserved solely to establish a connection to the external 16Mhz crystal oscillator. The 16Mhz crystal oscillator utilizes two 22pF bypass capacitors, both of which are grounded. Lastly, there is a 100nF capacitor connected to the wire trace coming out of AREF pin 21 which attaches to ground. All remaining pins on the ATMega328p are routed to an open slot on one of the two 18 socket on-board pin headers so they can be accessed for use by components not local to the PCB board.

A picture containing diagram

Description automatically generated

Figure 7: ATMega328p Pin Mapping

**Soil Moisture Sensors**

The first pin of each soil moisture sensor is the VCC pin. This pin supplies power to the sensor and is connected to one of the pins on the SSQ-118-03-F-S 18 socket pin header. This +5V pin receives power directly from the output of the LM2576SX-5.0/NOPB voltage regulator circuit. The second pin on the sensor is the GND pin and is connected to one of the designated GND pins on the 18-socket pin header. The digital pin on the soil moisture sensor is not used. The reasoning for the lack of connection is to reduce corrosion, which was a common complaint found by other engineers who have utilized this same component. Lastly, the fourth pin on each soil moisture sensor, which is the analog I/O pin, connects directly to pins 23, 24, and 25 on the ATMega328p which are designated as ADC pins on the ATMega328p datasheet. All connections between the soil moisture sensors and the PCB board are made using jumper cables and the on-board pin headers.

**Temperature and Humidity sensor**

The first pin on the DHT22 is the VCC pin that supplies the device with an input voltage of 5V. It is connected to one of the pins on the 18-socket pin header that is routed to the output of the 5V regulator circuit. It is important to note that technically this sensor can operate in between the supply voltages of 3.3 V to 5.5 V, although 3.3 V is not recommended. The reason for this is with the 3.3V supply you are limited to a cable length of no greater than 1 meter (if greater than 1 meter there will be errors in measurement due to the line voltage drop). On the other hand, if a 5 V supply voltage is used the sensor can be kept as long as 20 meters. The second pin on the DHT22 is connected to an on-board PCB pin header socket that is routed to pin 11 on the ATMega328p. This connection serves as the data bus for the transfer of information between the MCU and the sensor. Lastly, pin 3 is connected to a ground pin.

**Wi-Fi Module**

The ESP 8266 Wi-Fi module is mounted directly on the PCB board. The 3.3V and the EN (CH\_PD) pin on the module are connected via net traces and routed to the output of the LM2576 3.3V regulator circuit. The TXD and RXD pins on the Wi-Fi module are connected directly to pins 16 and 17 on the ATMega328p. The GND pin on the ESP8266 is individually grounded. The remainder of the pins on the module are left un-connected.

**2-Channel Relay**

The Sunfounder 2-Channel Relay requires a 5V input voltage in order to efficiently run. The VCC pin on the relay is connected to one of the on-board PCB pins that is routed to the output of the LM2576 5V regulator circuit. The IN1 and IN2 pins on the relay are connected to the on-board PCB pins which are routed to pins 4 and 6 on the ATMega328p. The GND pin on the relay is connected to one of the designated GND pins on the SSQ-118-03-F-S.

5.1.3 Layout

The purpose of this section is to discuss the current layout of our PCB board design and comment on all changes made during the design process.

In Figure 8, shown below, is our initial 2-layer PCB design. The footprint of the board is 6000 × 6000mm (6 inch x 6 inch). Eagle and Fusion 360 software were used in the creation of the schematic, and Fusion 360 was used to create the final PCB board layout. The design depicted contains two 18 position pin headers which were going to be used to easily and efficiently connect all external sensors as well as our 2-channel relay module to our PCB board. Mounted on-board are the 3.3V and 5V regulator circuits, our ATMega328pu micro-controller and external 16Mhz crystal, as well as the ESP8266 Wi-Fi module. The board was planned to be powered by a 9V 2A AC/DC converter through an on-board DC power jack.

Diagram, schematic

Description automatically generated

Figure 8: Fusion 360 Initial PCB Design

Our initial PCB board design was ordered and manufactured by JLCPCB and came out to a cost of about $80 including shipping and a stencil. Unfortunately, after acquiring the PCB board we realized that there were significant issues with the product. These issues had nothing to do with manufacturing errors, but instead stemmed from issues with the original schematic. The issues included forgetting to ground both voltage regulator circuits, routing the VCC and AVCC pins on the ATMega328p incorrectly, failing to add all necessary decoupling capacitors to the pins of the ATMega328p, and failing to implement a pull-up resistor on the reset pin of the ATMega328p. In addition to this, after initial testing of this iteration of our PCB board design, when the planned 9V 2A input was utilized to power the board our 3.3V regulator went into thermal shutdown and our inductor overheated.

After spending a significant amount of time troubleshooting our original PCB design, we decided that given our tools and experience level, the board was not able to be fixed. After the decision was made to scrap the initial PCB design, a new schematic (shown in Figure 6) was created, and all issues stated above were addressed and fixed. As a result of that, our finalized 2-layer PCB design, depicted in Figure 9, was created. The footprint of the board is 6000 × 6000mm (6 inch x 6 inch). Fusion 360 software was used in the creation of the schematic, and the final PCB board layout. Unfortunately, due to time constraints when designing the PCB, we were unable to add some of the silk screen text that ideally would have been desired for the final build.

Diagram

Description automatically generated

Figure 9: Fusion 360 Finalized PCB Design

Similar to our initial PCB design the finalized version also contains two 18 socket pin headers which are routed to individual pins on the ATMega328p and serve to connect all external sensors as well as our 2-channel relay module to our PCB board. Mounted on-board are the 3.3V and 5V regulator circuits, the ATMega328pu micro-controller and all necessary decoupling capacitors, the external 16Mhz crystal with two bypass capacitors, and the ESP8266 Wi-Fi module. Due to issues with the input current when testing the first iteration of the PCB board, the board was planned to be powered by a 9V 1A AC/DC converter through an on-board DC power jack.

Unfortunately, the finalized custom PCB board design depicted in Figure 9 was not able to be shipped in enough time to be utilized in our final project presentation and demo. As a result of this, two PCB development boards were soldered and used in place of the custom manufactured PCB board design.

.

A picture containing electronics, circuit

Description automatically generated

*Figure 10: PCB Development Board*

Figure 10 shows our soldered PCB Development Board. The far left development board contains the ATMega328p to include its external 16Mhz crystal, the 10k pull-up resistor, and all necessary bypass and decoupling capacitors. In addition, as desired from our custom manufactured PCB board, every pin on the ATMega328p with the exception of pins 7, 8, 9, 10, 20, and 22 are individually connected to a socket on one of the pin headers closest to its location. The line of pin headers on the far right, far left, and top of the development board as shown in the picture above were made into external GND, 5V, and 3.3V connections respectively. A very similar concept was going to be utilized in the manufactured PCB board design to ensure there were enough 5V and 3.3V connections for the user to power all components for our design at once, while also leaving room for the addition of extra sensors if they so desired. The Development board in the center of the figure above contains both our 3.3V and 5V regulator circuits. The output of these circuits were wired to the 5V and 3.3V powerlines created on the top and far right of the first development board. Both regulator circuits are connected to and grounded via a female power jack. The female power jack is 2.1mm x 5.5mm and is used to connect our 9V 1A AC/DC wall adapter.

5.1.4 Manufacturing

The final step in the process of designing and creating the PCB required for our project was manufacturing. Once our initial PCB design was finalized, we had to send the information to a third-party vendor. This was done by downloading all the boards specifications in what is known as a gerber file. This file along with the board dimensions was uploaded to the vendors website. Any and all other specifications of the purchase, such as the number of PCBs being bought, the inclusion of a stencil, and the utilization of the vendor for physical soldering of components were then decided. Based on these decisions, the company generated a general quote for a price before shipping costs.

Originally, when beginning the process of choosing a manufacturer, our group heavily discussed whether we wanted to use a company local to the United States or expand our options internationally. We originally felt that due to the continued effects of the pandemic, along with supply chain issues and growing international political conflict, that the smartest decision when choosing a manufacturer would be to choose a company based in the United States. Although, after beginning the process of searching for companies to develop our PCB board we found that most international companies provided a high-quality product in a quicker turnaround time. The end goal for our PCB board was to get the most reliable product for the cheapest overall price, and while our second PCB board was not able to be ordered and shipped in time for our final presentation, our first iteration PCB did in fact meet our goal in regard to quality and price.

5.2 PCB Manufacturing Vendors

In this section we will be discussing all the vendors that our group looked into utilizing to manufacture our PCB. The top four vendors that were reviewed by our group prior to purchase of the first iteration of our PCB were RUSH PCB, JLCPCB, Advanced Circuits, and OSH PARK.

**RUSH PCB**

This manufacturing company is located in San Jose, California in the United States and was established in 1997. The company is deals in assembly, fabrication, design, and manufacturing of different printed circuit boards. Some of their assembly services include surface mount, mixed, and thru-hole with full turn-key in addition to testing processes. They provide fabrication and manufacturing of HDI, rigid, flexible, and rigid-flex PCBs. Their company is multi-certified as a PCB manufacturer and they pride themselves in their state-of-the-art equipment and highly-trained staff.

**JLCPCB**

JLCPCB was founded in 2006 and is slowly becoming a leading global PCB manufacturer. It is a company with over 300 employees and 800,000 plus customers. JLCPAB handles over 20,000 orders a day and produces upwards of 6 million PCBs per year. Their on-time delivery rate is over 99.97% and they have a quality complaint rate of less than 0.23%. The company promises a PCB with a quality level suitable for military, aerospace, industrial, and medical applications coupled with a lower cost than most of their competitors.

**Advanced Circuits**

Advanced Circuits, established in 1996 and located in Aurora, Colorado, is North Americas third largest PCB manufacturer. This company offers a wide range of services including but not limited to the design, stencil, assembly and manufacture of a variety of diverse PCBs. ACT offers a wide range of capabilities when it comes to circuit board production with one example being their conformal coating capabilities of which include parylene coating. The manufacturer also has a history of quick turn-around times for both large and small orders. Similar to RUSH PCB the company is a multi-certified manufacturer with approximately 45 percent of their production throughput being military products. They house their equipment in a 35,000 square foot facility that includes a 14,000 square foot temperature and humidity controlled SMT production area with anti-static flooring.

**OSH PARK**

OSH Park is a PCB manufacturing company local to the United States and located in Lake Oswego, Oregon. They claim to offer high precision boards suitable for industrial, aerospace, and medical applications. The ordering options from OSH Park are limited, but based on numerous reviews the overall process is said to be simple, and the customer service is praised.

**Final Decision**

Our group utilized multiple companies throughout the duration of our project. For the first iteration of our PCB board design our group ordered through JLCPCB. The ordering process was straightforward, and the manufacturing costs were extremely low for the production of 5 boards. In addition, the stencil for the board was included at no extra cost. Shipping was not overly expensive given the company is not local to the United States. The second board that was ordered, which was not able to be shipped in time for our final presentation, was ordered through OSH Park. The reason this company was chosen was due to their ability to offer one-day shipping. Due to the fact this company was local to the United States, it was assumed the overall turnaround time would be significantly faster than competing overseas companies. An order of only three PCBs from OSH Park ended up being three times as expensive as JLCPCB. In addition to this, the stencil had to be bought separately, and was more expensive than our entire first purchase from JLCPCB. The manufacturing time was also around three times longer for OSH Park than JLCPCB. Once the OSH Park PCBs were received, even though it was after our final presentation was due, the boards were inspected, and their quality was significantly lower than their JLCPCB counterpart. Overall, based on our experiences ordering PCB designs, if we were to order again we would use and recommend JLCPCB.

6 Software Design

Microcontrollers, sensors, and communication devices can be quite precise and complex technologies. Since it would be practically impossible to make these things work together in any way with simple hardware, the only logical choice is to use software to get the individual parts of this project to come together and create a functional system.

6.1 Arduino IDE

Microcontrollers themselves are many individual parts that make a system, and even though they are part of the same Arduino Uno R3 board, they still need some instructions on how to work as a team. Things get even more complicated when other devices like sensors are connected, since they may produce any kind of data based on their own hardware.

The Arduino IDE is a development environment built by Arduino to help users create programs to give Arduino products instructions using C++ as a programming language basis, but with some minor differences. Using an IDE (Integrated Development Environment), we were able to create and edit code to tell the Arduino microcontroller how to use the data it receives, and what functions to perform in certain conditions. As for what conditions, that would be all the conditions that we measure to keep the plants healthy, such as soil moisture.

Each sensor provides its measurements in some form, but not necessarily following a specific standard. In order to use those measurements effectively, we need to tell the ATMEGA what parameters to use to determine when to trigger certain functions of the SMART Garden. The soil moisture sensors provide a range of values between 0 and 1023, but we are the ones who know that a lower value means the soil moisture is high, and we created the code accordingly. Once the code, called a ‘sketch’ in the Arduino IDE 2.0, is created, then it must be compiled, and the resulting files must be loaded onto the ATMEGA processor. After that, the microcontroller is able to function as expected without further guidance. Though we used the Arduino Uno R3 for testing, it is important to note that in the final project, the ATMEGA328P processor was to be removed from the board and inserted into the IC socket on the PCB. The Arduino IDE was simply used to load programs into the ATMEGA328P’s onboard memory without using complicated circuitry setups. This allowed for easily swapping the processor between the Arduino board and the actual project to quickly alter code when needed for testing.

6.1.1 Libraries

Sometimes, it isn’t as difficult as it sounds to interact with a device like a sensor or a Wi-Fi module. If the device is widely used or standardized enough, there might be a library you can use. A library is a file that can be included in the program you write for the ATMEGA to help it interpret your code correctly for specific scenarios. For example, if there is a screen connected, you might be able to use a command called something like drawCircle(), instead of learning how to individually command each pixel of the screen and manually code in a way to create a circle. This would be extremely tedious and impractical, so libraries exist to simplify the creation of your programs by using functions and constants (variables) that have already been created. These libraries can vary in function greatly, from drawing shapes on displays to simply lighting up LEDs when instructed.

Every library isn’t constantly present in a sketch. There are a lot of libraries available in the world, so loading each and every one onto your microcontroller is not possible. Instead, you need to know what you are going to be using and actively instruct the IDE to include it in the sketch with special formatting for this purpose. Instructing your IDE to link a library to the sketch can increase the size of the final file. This means that if you are making a huge project, or working with significantly limited space, it might be worth it to find alternate ways to include a function than to include its library, especially if that library is exceptionally large. Additionally, a library may not prioritize using small variables, and trying to use them could result in running out of dynamic memory during runtime, thus causing issues ‘unrelated’ code. Removing the library from the sketch is as simple as deleting the relevant code from the program. Once it is no longer being instructed to include the library when compiling, it no longer will be used.

6.1.2 Bootloader

Arduino microcontrollers come by default with a bootloader installed already. The bootloader’s function is to be the first program that runs when a microcontroller (or whatever system it is installed in) is powered on or reset. The bootloader then performs the necessary steps to load more useful programs. This is because the main instructions of a program loaded into the ATMEGA are stored in non-volatile memory. The bootloader places these instructions into volatile memory as needed so that it can access them easily and effectively.

It is also possible to remove or replace the bootloader with a different application, allowing this application to start immediately when the Arduino is powered on, rather than the possibility of waiting a mandatory 10 seconds (depending on the bootloader version installed). If the device is receiving information constantly, the bootloader might think this is important information and delay finalizing startup, possibly indefinitely.

Bypassing this step can force the device to ignore any possible software updates in favor of starting the program without delay. Since our SMART Garden was originally designed to be constantly be receiving data from the sensors, this is an option that we considered looking further into. Because there are specific onboard functions that are able to alter whether the signals from the sensors are interpreted as real, meaningful data, they didn’t end up delaying the microprocessor fully booting into our program that monitors and maintains the plants in the garden. Needless to say, getting stuck in the bootloader would have gone against many of our objectives for this project.

If the bootloader is removed or becomes corrupted somehow, it is possible to reinstall the original bootloader, or install a new, more applicable bootloader. For example, if quick response time is important, you would likely want to remove the bootloader that takes 10 seconds every time it turns on to see if there is any new data to integrate from an official Arduino program, such as a flashed sketch. If having a bootloader at all is still too slow, loading your own program is likely the best option. Rather than waiting for any Arduino software or updates to go through, the bootloader will immediately get to your program.

It is possible to program the ATMEGA microprocessor without a bootloader, by burning sketches to the chip directly, but there is no functional benefit in most cases. A primary reason to do so is if you are trying to conserve space on the device, since the bootloader is not taking up any space that could be used instead for more burned sketches. This method requires deleting the bootloader from the Arduino and using a separate programmer than the Arduino IDE. The Arduino website suggests doing this with an AVR-ASP, STK500, or a parallel programmer.

If you decide to return the microcontroller to standard functionality, it is possible to rewrite the bootloader to the microcontroller from within the Arduino IDE.

6.2 Software Communication integration

The software integration element of this project involves programing a website element that allows the user of the SMART garden to access the application, and by extension the device itself, remotely.

The way the information is communicated from the SMART garden to the other devices will be through the use of an remote database that will be updated with the ATMEGA via a Wi-Fi module. There was some deliberation between using MySQL and MongoDB. The remote database MySQL is a table-based system. The table-based architecture utilizes a data query structure. Each entry is saved as a row in MySQL. MySQL is also a relational database meaning that it stores its data in different tables as opposed to one large storeroom. We can then create a set of rules to dictate the relation between those tables. It was a reasonable consideration given that it is also open source.

The second remote database in consideration was MongoDB. MongoDB is a document-based, non-relational database management system or object-based system. MongoDB keeps every record in separate documents. Documents in a specific group are kept in a collection.

Given the database for our project has reason to access data by multiple avenues, the ease of utilizing the MySQL database, and my personal prior experience using the software tool, the remote database to be used will be MySQL. MySQL was selected due to it being highly available and scalable as well as easy to program.

A remote API must also be utilized. The one we decided to work with is php, as the server we host the website on allows for easily utilizing the LAMP software stack (Linux, Apace, MySQL, and Php). The API is the functional back-end of the web application framework. Php files receive requests and take actions to alter the database and/or return data. For example, a php file will receive the most recent readings from the sensors in the garden and update the database with the readings by sending a MySQL ‘update’ command. They can also return requested data, such as a list of all the preset plants included in the database that will be presented to the user for selection. To act as a middleman/connector between the html of the website and the php of the backend, Javascript was utilized. This can be used to both alter the content of html elements and to send requests to specific php files. This allows it to retrieve ask a php file for specific data from the database, and use that data to update the webpage the user sees accordingly.

While running, the ATMEGA will collect the relevant information from the sensors in the project i.e. the moisture levels, temperature, and humidity, and update it to the MySQL database. This information is sent to the php files by providing the data, in the correct format, via the attached Wi-Fi module. Since a relatively simple Wi-Fi module was used, rather than one that has more onboard functions, the form of the command, as well as the data structure, was handled by the code in the sketch. Sending specific ‘AT commands’, or attention commands, to the Wi-Fi module provided it instructions on how to do many of its functions. The setup part of the sketch runs once, connecting the module to the nearby Wi-Fi network, and the loop part runs continuously. The primary functions of the garden are in the loop. With delays included, which ensure the Wi-Fi commands are sent/received completely each loop completes in roughly 17 seconds. Each loop both updates the database with the most recent readings, and reads the user’s currently selected preferences from the website. All actions of the garden itself are taken automatically, with no input required from the user beyond ensuring the water reservoirs remain filled. Once the correct plant is selected from the website, the ATMEGA will receive all required information for operation via Wi-Fi.

Even though this one is simplistic, the utilization of activity diagrams was useful in helping track what process runs to where.

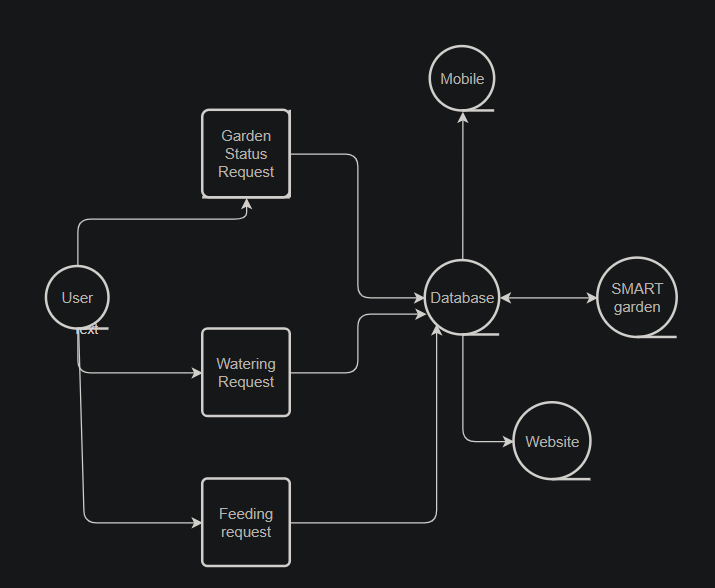


Figure 11: Activity Diagram

6.3 Website development

Our website was hosted on DigitalOcean, and implemented a LAMP program stack as previously mentioned. The page was coded in HTML and referred to Javascript code to keep the content displayed up to date. The site was coded in Visual Studio Code, and tested by either deploying the code to our server, or testing it in a locally hosted server. Layout could largely be tested with the local server, but some things, like the Javascript, were coded to be retrieved by URL, so were often easier to test by fully deploying to the Apache server. This was done by committing any changes to our Github repository and replacing the live server’s files with those on github. Replacing these files was done with only a few simple lines of code executed in the Linux environment of the server via an SSH client called PuTTY. PuTTY was also used to go one level deeper on the server and access the MySQL database. Here, three tables were created. One was called ‘plantprefs’ and held the names and ideal conditions of the different plants we found. Additionally, it contained the custom entry that could be altered as the user desired, with a custom name and condition parameters (moisture, humidity, and temperature). This table was used as the template for another table called ‘currentplant’. The ‘currentplant’ table had a nearly identical structure to the ‘plantprefs’ table, and was updated to reflect the user’s selection of plant. When the garden sends an http GET request to getPlant.php, the values in ‘currentplant’ are returned. When it sends an http PUT request to updateReadings.php, the input values are used to update the last table in our database, ‘currentreadings’. This table also has a similar structure to the others, and is overwritten with any values sent into the updateReadings php. These values are retrieved by the website to display to the user the most recent readings by the garden.

While the website was still in development, the endpoints needed to be tested to make sure they worked. One way to make sure each part is working is to eliminate as many variables as possible. In this case, that meant removing as many layers with possible errors as possible. If clicking a button on the website wasn’t working, that could be the Javascript, the HTML, the php, or the server not successfully updating any of those. HTML and Javascript were easier to test locally, but phps on the server were tested with a REST client (Representational State Transfer). In our case, the tool of choice was Insomnia, which is very user friendly, with little coding needed to create valid http requests. After entering the php’s URL, selecting the HTML method (PUT or GET in our case) and entering the data in JSON format, clicking ‘send’ would trigger the php to do as it should, or would display any errors returned which allows for easier debugging. As the php files all send MySQL commands, these could simply be tested from within the MySQL interface of the server to verify syntax and effect, removing yet another layer of possible failure.

7 Hardware and Prototype Testing

7.1 Hardware Testing Environment

In this section we will perform a comprehensive overview of the hardware testing environments that were used for all components of the SMART Garden project. The equipment needed in order to test all hardware components were a the Arduino Uno, OLED display, soil moisture sensors, temperature and humidity sensor, water pumps, relay, multimeter, breadboard, 220-ohm resister, and jumper cables.

7.1.1 Sensor Testing

In this section we will be going over the testing process for the two sensors our group utilized for our project. This testing was done by creating individual circuits centered around determining the performance of each individual sensor. The sensors were all connected to the Arduino Uno and code was run to verify all aspects of the component were not only working, but also providing accurate readings and outputs.

**Soil Moisture Sensor Testing**

The soil moisture sensor is one of the most important components of our project when it comes to preventing over or underwatering of our plants, it also plays an important role in gathering data used to optimize the automated watering system our group is planning to implement. In this section we will go over the testing process for the soil moisture sensor as well as walk through the thought process behind deciding threshold values for the system.

The soil moisture sensor uses a dual-pronged probe with two exposed conductors which essentially act as a variable resistor. These two probes allow for electric current to pass through the soil and measure the moisture level by analyzing its resistance. In order to set threshold values for our system we first had to research how soil moisture effected conductivity and resistance in the soil. The conclusion we reached is as follows:

* + Higher water content in soil increases conductivity and lowers resistance
  + Lower water content in soil decreases conductivity and increases resistance

In order to find our threshold values the first thing we did was create a small circuit using the soil moisture sensor and Arduino Uno R3 as depicted in Figure 10. We connected the analog data pin to the A0 pin on the Arduino, the VCC pin on the soil moisture sensor to the +5V pin on the Arduino, and the GND pin on the Arduino to the GND pin on the soil moisture sensor. Lastly, we connected the positive and negative pins of the soil moisture sensor to the dual-pronged probe.

After the circuit is set-up, the next step was to either write or find an open-source code which will output the numerical analog resistance values read by our sensor. Once this code was written the final step was to begin testing soil moisture values. For the purpose of this experiment, we tested five different soil samples in order to calculate our threshold values. The first soil sample we tested was completely dry (sensor will be dried and cleaned prior to use so as to not skew values). The purpose of this is to find the approximate max resistance value in our range of values. Once this value had been recorded, we repeated the process with each other prepared soil sample, which are differently moistened in order to find the approximate resistance value for our range of values. The most saturated soil sample had a resistance nearly identical to plain water, and this output value was used to represent the maximum moisture value, 100%, relating to the minimum resistance likely to occur in the garden. We also tested completely dry soil, which returned the same output value as the sensor did in open air. This value represented 0% moisture. Based on the ratio of water to soil used to create the 100% moisture soil, we created soils of different levels of moisture. For example, half as much water was added to the same volume of soil to create soil that would be calculated as being at 50% moisture. In the end, soil samples of 0%, 25%, 50%, 75%, and 100% were tested.

Once we had recorded our soil moisture sensor output values, we then used them to find an equation that related soil moisture percentages to soil moisture sensor readings. This equation would be used to calculate the current soil moisture level in the garden. Our database contains a list of plants with their ideal conditions, including the optimal soil moisture level. Since each plant has different ideal conditions, it is important to get as close as we can to the correct moisture for the user’s specific plant. The equation discovered (through an online calculator based on our readings) is as follows, with *S* representing the soil moisture sensor’s reading as seen by the Arduino/ATMEGA:

Since the equation isn’t linear, we are unable to use a simple ratio between highest resistance and lowest resistance. In our readings, 100% moisture corresponds to a reading of around 150, and 0% corresponds to around 1050. If the moisture level in the garden falls below a plant’s ideal moisture, it will be watered to make sure the plant isn’t too dry. If the moisture is above this value, the garden waits for the soil to dry naturally, waiting for the moment the moisture value is once again low enough to water the plant. The values below show the moisture levels of the test values and some plants’ ideal values and their equivalent analog representation.

|  |  |  |
| --- | --- | --- |
| Soil moisture value | Soil Moisture Content:  Analog Representation | Soil Moisture Content:  Percentage |
| Soil sample (100%) | ~150 | 100% |
| Soil sample (75%) | ~280 | 75% |
| Soil sample (50%) | ~467 | 50% |
| Soil sample (25%) | ~640 | 25% |
| Soil sample (0%) | ~1050 | 0% |
| Basil ideal moisture | ~619 | 27% |
| White Onion ideal moisture | ~429 | 51% |

Table 19: Soil Moisture Content Representation

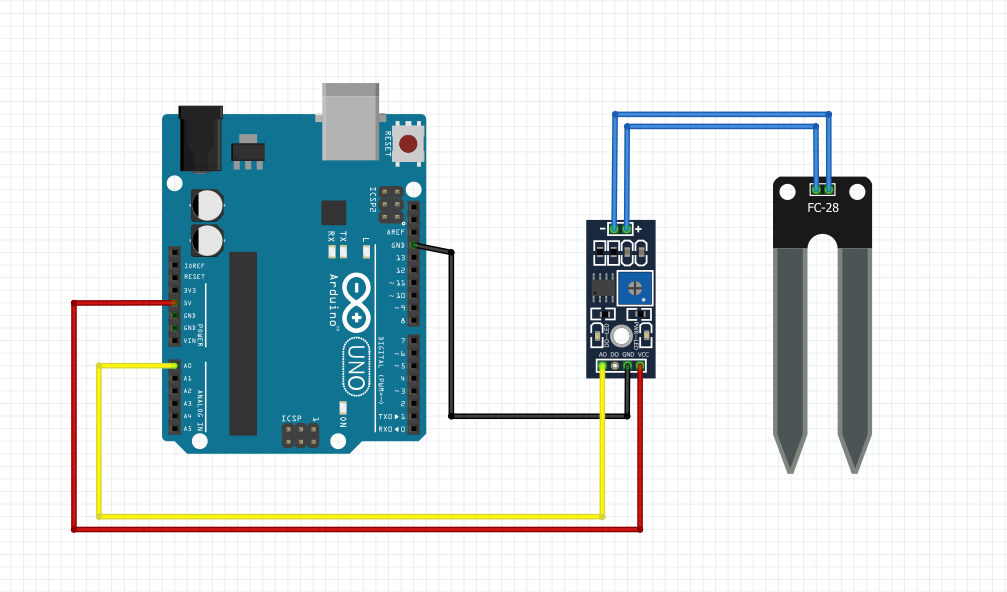


Figure 12: Soil Moisture Sensor Testing Circuit

**Temperature and Humidity Sensor Testing**

The Temperature and Humidity sensor our group used for our project is the DHT22. In this section we will review the testing procedure for our chosen sensor. The purpose of this was to ensure the device we purchased had no defective or faulty components and to verify it was able to give us an accurate reading of both the temperature and humidity of our surrounding environment. We were able to see the output of data using Arduino IDE by utilizing the serial monitor function available within the software.

The first thing we did was create our circuit. For this sensor test we used the Arduino Uno microcontroller, a breadboard, our OLED display, jumper cables, and one 220-ohm pull-up resistor. We started by making a horizontal node at the top of our breadboard which we utilized as a shared 5V power line. Following suit, we made a node located at the bottom of our breadboard directly beneath the blue horizontal line (as depicted in Figure 13) our common ground. The pin and breadboard connections we used for the part of our circuit containing the DHT22 are as follows. To start off, the left most pin, as depicted in Figure 11, on the DHT22 is considered pin 1. When reading the diagram from left to right the pin numbers increase from 1 to 4. Pin 1 on the DHT22 is also known as VCC and was connected to the shared 5V power line. The second pin on the sensor, known as the data pin, had one jumper cable connection extending from the node on the breadboard to digital pin D6 on the Arduino Uno. Directly below that connection but still on the same node as Pin 2 was where we placed the 220-ohm resistor. On the opposite end of that resistor a jumper cable that connected back to the shared 5V power line was used. Due to the fact the DHT22 sensor we bought came on a breakout board Pin 3 was not available for use. Lastly, Pin 4 or GND was connected to the shared ground.

Background pattern

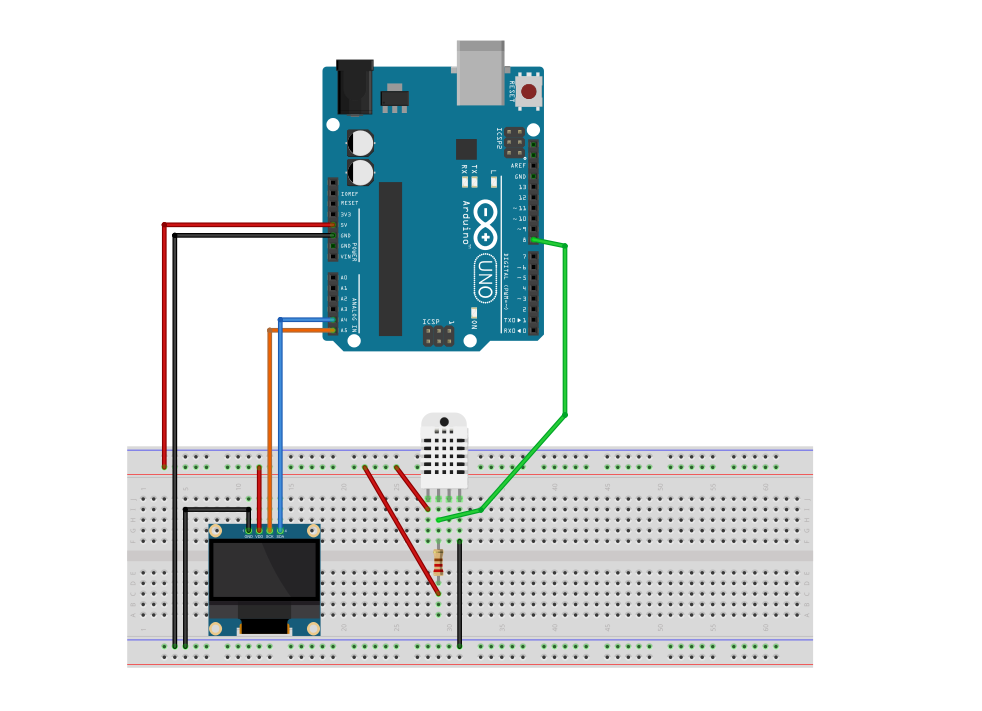
Description automatically generated with medium confidence

Figure 13: Humidity and Temperature Sensor Testing Circuit

Once the circuit was created, the next step was to run a code via the Arduino IDE. When testing this circuit, we read the sensor data values by utilizing the serial monitor within the Arduino IDE software. Similar to the OLED display we utilized the large database that is offered through Arduino. The two libraries that we included in our code in order to output values for the DHT22 sensor were the Adafruit DHT sensor library and the Adafruit Unified Sensor library. In addition, both libraries that were used in the OLED display testing procedure were included within this sketch.

Once the code was completed and ready to be run, we moved our sensor into multiple different environments with varying temperatures and humidity levels. The purpose of this was to make sure our sensor could accurately detect changes to its surrounding environment in real time. The first place we tested the sensor was inside where we had access to the thermostat and could easily check the temperature to cross reference that the value matched the data output of the sensor. During testing, the sensor correctly outputted the temperature (within a small degree of error) that we read on the thermostat inside.

The second test that we completed involved the humidity portion of the sensor. Due to the fact we had no way of accurately knowing what the humidity was inside, we tested this sensor outdoors. We used a weather app to determine the humidity level and cross referenced the value with our data output to ensure that it matched our sensor. Just as with the temperature component of the sensor, the device outputted a value (within a small degree of error) that the group deemed to be precise enough to meet the requirements for use in our project.

7.1.2 OLED Display Testing

The OLED Display chosen by our group to be used in our project was the 128x64 OLED dual-color display with an SSD1306 controller. While this component did not end up being implemented in our final design, testing procedures were still utilized upon the original purchase of the product. In this section we will discuss the testing process we used to check the display. The purpose of this testing process was to ensure that the component came with no defects. In addition, before this device was removed from our design, we were testing to make sure the board had the capability to output graphics that matched the level of complexity we wanted our display to have in order to maximize the end-user experience.

The first thing we did was connect our OLED display to our Arduino Uno. The pin connections we used during this test are as follows. We connected the GND pin on the OLED display to the GND pin on the Arduino Uno, the VDD pin on the OLED Display to 5V pin on the Arduino, the SCK pin on the OLED Display to the A5 analog pin on the Arduino Uno, and lastly we connected the SDA pin on the OLED display to the A4 analog pin on the Arduino Uno.

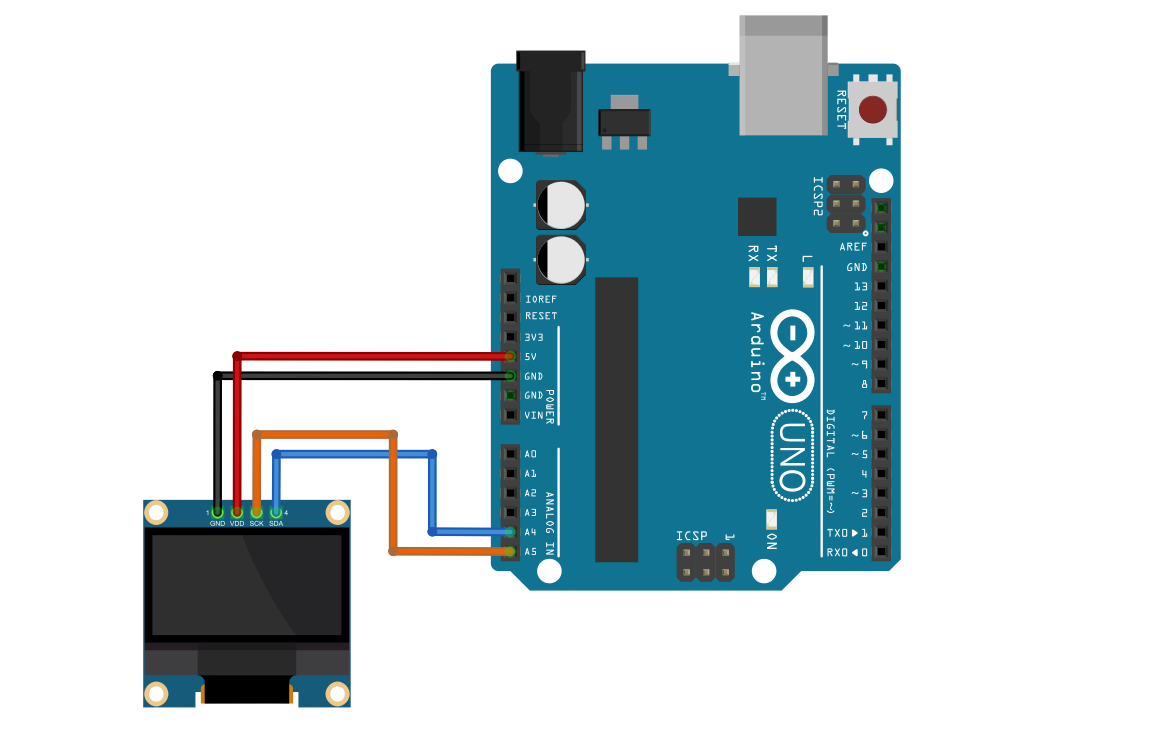
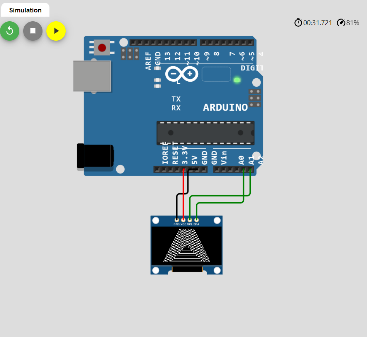
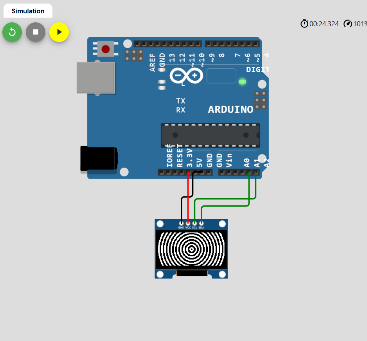
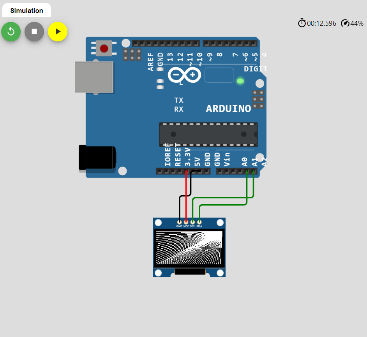


Figure 14: Arduino Connected to OLED Display

Once the circuit had been set up, the next step in the testing process was to make sure we ran a code that tested the graphics functions at a similar complexity to what we intended to use during our project. In order to do this our group utilized the large database of libraries offered through Arduino. The two libraries we downloaded are the Adafruit GFX Library and the Adafruit SSD1306 Library. Once downloaded we opened the 64-pixel display example located in the Adafruit SSD1306 dropdown in our Arduino IDE. In order for the code to execute we changed the display board address in the code from 0x3D to 0x3C. After the code began running the board was inspected for any issues and was determined to be in working condition. The types of graphics that were tested on the board can be seen in Figure 15. This contains screenshots of the virtual simulation that was done prior to physical testing to ensure we knew exactly what each output should look like so we could more accurately check for discrepancies.



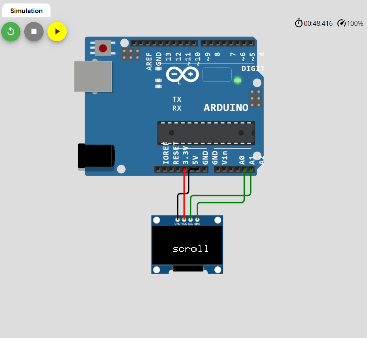
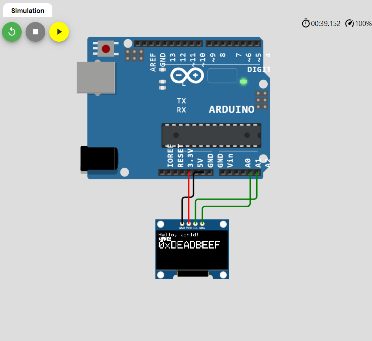
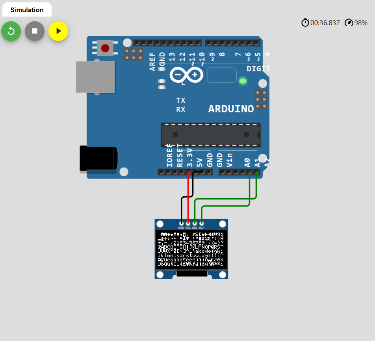


Figure 15: OLED Display Online Simulation

7.1.3 ATMega328p Testing

In this section we will go over what our testing procedures were for the ATMega328p microcontroller. In order to test all of the I/O pins on our microcontroller we performed a blink test. In this test we connected all digital and analog pins to an LED light in order to verify each individual pin was working as expected.

The steps we took in order to perform this test are as follows:

1) Installed Arduino IDE 2.0.1

2) Found open-source code to perform LED blink test

3) Created a testing circuit using ATMega328p microcontroller, capacitors, one LED, and one 220-Ohm resistor

4) Uploaded the code to ATMega328p using Arduino Uno bootloader

5) Checked each individual I/O pin and recorded results

A screenshot of a computer

Description automatically generated with low confidence

Figure 16: ATMega328p Testing Procedure Set-up

To start off, we built the ATMega328p the same as described in section 5.1.2 under the header ATMega328p. For this test, the output of our 5V LM2576 regulator circuit (not shown in Figure 1), was connected to the shared 5V power line located at the bottom of the breadboard.

Following this we grounded our LED light to the shared node located above the red horizontal line on the breadboard as depicted in Figure 16. The shorter leg of the LED, also known as the cathode (negative leg), was connected to this shared ground line. The longer leg of the LED, also known as the anode (positive leg), was connected to the same node as a singular 220-ohm resistor. This resistor is used to limit the current that flows through the LED to avoid damaging both the LED and the pin connection on the microcontroller. Using a wire, the opposite end of the resistor was connected to one of the I/O pins on the ATMega328p. Once the power supply from our voltage regulator source was connected, the LED at that pin began to blink as specified in the code. This process was repeated for all digital and analog pins, and no issues were found.

7.1.4 Power Supply Testing

The power supply unit was consistently tested throughout the duration of our project build to verify all components were working as expected. The purpose of this test was to ensure both our voltage regulator circuits were working correctly and outputting a regulated 3.3V or 5V. This circuit was set up as specified in Section 5 under the header LM2576XX. The test was completed using a multimeter as depicted in Figure 17. The dial on the multimeter was set to output voltage. The red wire labeled V-Ohm was connected to the output of the voltage regulator circuit, and the black wire labeled COM was connected to ground.

If the output voltage read within the acceptable range, as specified on the LM2576 data sheet, then the power supply circuit was deemed acceptable to use as a power source during testing. The 3.3V and 5V regulator circuits only outputted the incorrect voltage when user-error was involved, and the circuit was set up incorrectly. Overall, this power source was successful when implemented into our project both during testing and in final implementation.

Diagram, schematic

Description automatically generated

Figure 17: Multimeter Voltage Regulator Testing

7.1.5 Relay Module Testing

The Relay Module our group used for our project is the Sunfounder 5V dual-channel relay compatible with the Arduino Uno. In this section we will go over what our testing procedure was for our relay module. In order to test the performance of our relay module we wrote a simple code to trigger each relay. The purpose of this test was to make sure there were no faulty components in the relay module and that the device was working at the level of efficiency our project called for.

In order to build our circuit, we used the 5V dual-channel relay module, Arduino Uno, and jumper cables/wires. The pin on the relay module labeled VCC was connected to the shared 5V pin on the Arduino Uno. The pin labeled IN1 on the relay module was connected to the digital input pin 4 on the Arduino Uno. Following suit, the pin labeled IN2 on the relay module was connected to the digital input pin 2 on the Arduino Uno. Lastly, the GND pin found on the relay module was connected to the GND pin on the Arduino Uno. After testing, both relays triggered as specified in the code, and no other issues were found with the device.

A picture containing text, electronics, screenshot

Description automatically generated

Figure 18: Relay Module Testing Circuit

Before our group received all the materials for our project, and we were unable to do any physical tests, we decided to perform online simulations to verify our circuit set-up and code were correct. The purpose of this was to help with trouble shooting when physical tests began. We wanted to verify before tests began that the conceptual setup of our circuit and code were not an issue, so if issues with the relay did end up arising, we could be confident that the issue was in fact hardware related. In order to do this, we set up a circuit identical to the one described in Figure 18 and ran a simulation of our testing environment using the simulation platform Wokwi.

7.1.6 Water Pump Testing

# In this section we will go over what our testing procedure was for our water pumps. To begin, the water pump we chose for this project is the 5V mini submersible water pump. The purpose of testing our water pumps prior to project set-up and build was to find and identify all defective components early on so there was adequate time to order and replace them if needed.

The first thing we did was set up our testing circuit. To start off we set up the attachments between the relay module and microcontroller. The pin on the relay module labeled VCC was connected to the 5V pin on the Arduino Uno. Next, the pins labeled IN1 and IN2 on the relay module were connected to the digital input pins 4 and 2 on the Arduino Uno and the pin labeled GND on the relay module was connected to the GND on the powerline of the development board. The last step of the circuit set up was connecting the water pump to the relay module. To start off, when the relay was oriented as depicted in Figure 19 the third clamp on each relay channel was connected to the positive voltage on the power source (5V) battery pack, and the second clamp on each relay channel was connected to the red wire on the water pump. Lastly the black wire on each water pump was connected to the negative voltage on the 5V battery.

Diagram

Description automatically generated

Figure 19: Water Pump Testing Circuit

In order to run this test, we placed our water pump in a large water reservoir. We then verified that the pump was able to maintain a flow rate of 240L/H. We did this by having the water fill up a 4-liter container. The idea was if the pump was running efficiently, and maintaining the flow rate it should, then the 4-liter container would be fully filled within 1 minute. In order to test its efficiency and accuracy at longer working intervals we also let the pump run for approximately 15 minutes. We collected all water outputted during this duration of time, and 60-liters of water was collected, within a small margin of error, so we considered the water pump viable to use for the project. This entire process was repeated for each water pump we utilized in our build.

8 Software Testing Environment

This project required a great deal of programming to come to fruition, so it was just as important to know that the software is operational as it is to know each individual part works correctly. All the sensor readings in the world would mean nothing if we can’t tell the SMART Garden what they mean.

8.1 Software Development Collaboration

Since the members of our team cannot always work together in person, it will be very important to make sure we are all using the most recent code possible when we make progress in the main body of code for the ATMEGA’s behavior. To that end, we decided to maintain an online backup and source for this code on the “developer platform” called GitHub. With GitHub, we could all submit changes to the files in use for this project when we succeeded in integrating any components through the microprocessor. There are also features within GitHub to publish changes to separate sections called branches, rather than committing completely to altering the main code. These branches can be marked for review before publishing so that other team members can look over the work and see if they agree there will be no issues with the code before merging the branch with the main code.

While the GitHub website is perfectly serviceable, there are several additional applications that used to streamline this process further, such as GitHub for Desktop, and Visual Studio Code, which can cooperate to download and display code in many languages. This would include the language that the Arduino IDE is based in, C++. Though the various functions of the IDE such as compiling aren’t accessible through Visual Studio Code, it can be an easy way to view and edit code in case there is a small change that needs to be made without manually downloading and loading the newest version of the code from the GitHub website.

Since much of the format of the Arduino IDE was initially unfamiliar to us, it was extremely important to make notes of what our intent was behind each change we made to the sketch. Though the comments within the code might be plenty, it served us well to thoroughly document our changes when we pushed code changes to GitHub. This let us explain the changes in description field in less-concise language than we may want to use in the code’s comments themselves. We also won’t have to scour the code for each change like the titular Waldo in a Where’s Waldo book, instead we could refer to our diligent team member’s note about what was changed, why they used the code they did, and what the intended outcome was. Hopefully these descriptions wouldn’t include any noticed bugs, but if they do, it would be extremely helpful to know what they are in case we know how to fix them.

Working with the most recent version of the software was very important in testing whether something would work, making this much certainty necessary. One change in one section could easily render another change in another section nonfunctional. Due to these risks, instead of hoping to skirt around these conflicts we instead ensured that whatever changes we made weren’t encroaching on the changes another team member has already made. Integrating the versions without being this sure would easily lead to issues with both members being sure that their change worked when they tested it and thus overlooking the correct solution at first. Though this may be fixed quickly enough, any delays should be avoided, especially when they are this easy to prevent.

Since the code should have been created with the goal of being modular in mind, it should be relatively simple to test each feature individually. It will be important to ensure each feature is functional before expecting them to be able to work together correctly. This applies to every level of the project, from the code on the Arduino Uno R3 to the database holding the information it reads. If anything goes wrong with any of the systems, it can mean our project ceases to function. Whether our garden correctly measures the soil moisture means nothing if it cannot retrieve the target moisture from the database, just like it doesn’t matter whether it can retrieve the target moisture from the database if it fails to interpret the information from the soil moisture sensor. Each step needs to work correctly for the project to work as a whole.

8.2 Specific Service Testing

The parameters for success in each service were only slightly different, largely involving checking whether or not the tested device can affect lasting change on another system, but all involved a little trial and error. Once we were able to affect the change we desired in the method that we intended to test, we could call the test a successful demonstration. Each service would, however, involve significantly different technologies, each presenting a different set of difficulties to overcome. Any change in another aspect of the project had a chance to cause issues in another section. During development, it was a good idea to test each system below when a great enough error occurs. This narrowed down the origin of the error by making sure each system was operating correctly on its own, saving the team from searching for problems to solve where there are none

8.2.1 Wi-Fi Module Testing

Since the testing of the microcontroller itself and the involved components has already been addressed, it would be most helpful to discuss the interactions between the microcontroller and the various internet-service-compatible methods of communication we decided to involve. The Wi-Fi module was instructed by the previously mention AT commands, which prompt it to await further instructions and use them in a certain way. For example, AT+CWJAP=”wifiName”,”password” would instruct it to join a nearby Wi-Fi network with that SSID and attempt to use that password. These commands can be passed to the Wi-Fi module via code derived from the SoftwareSerial library, but there are simple ways to set up an Arduino Uno R3 board to allow a user to type these commands to the module directly. This was used to ensure the format of the commands to be coded was correct before spending the time required to code them in, and removed possible format errors from the list of possible errors to test, leaving other errors to be likely caused by Arduino IDE code, rather than AT commands.

Once we got the Wi-Fi module activated, it was relatively simple to test whether it could communicate with a specific device. Once we believed we are able to attempt to connect to a specific Wi-Fi device, like a phone’s mobile hotspot, we could view the devices that were connected to the hotspot. If the Wi-Fi module (or more specifically, the esp8266 ESP-01) is displayed, we knew we could then communicate with it via Wi-Fi. Further, we tested whether it could connect to the internet by using the AT+CIPSTART =<method>, <domain>, <port#> command. When it returned the ‘OK’ response when connecting to our website, we knew we could then attempt to send further commands instructing it to send http requests to certain phps which would be required to cause any meaningful change to the database.

8.2.2 Database Testing

This interaction was a bit more difficult than the previous one to test as it required two parameters. First, the Wi-Fi module must cause a change in the database we set up for the project, and two, the module must be able to receive and interpret a variable from the database. Once it can read the change, it can adjust its behavior accordingly- reading it in the first place is the bigger hurdle. This would logically have to come after we have proven that it can successfully communicate via Wi-Fi if we are having it get to a database via the internet.

Once we were able to instruct the Wi-Fi module where to go to access the database and change a file on it, we were able to successfully communicate with the database from the SMART Garden. We did this by sending an http PUT command to a specific file called testJSON.php so that we could test the sending of JSON-formatted data to a php. The function of this file was confirmed via Insomnia, and the Wi-Fi module’s function was confirmed by this php altering the data as expected when sent from the Wi-Fi module. Similarly, the ATMEGA (tested via the Arduino IDE’s Serial monitor to easily view String variables) was able to receive and print the responses of http GET requests. At that point, parsing the data would be all that is required to use it as a variable. Because we could actively change the database and retrieve a set of data from the database from code run from the Arduino(testing)/ATMEGA(performing), we considered this portion a success.

In the final version of the project, this is how the garden updates the database where it sends and receives data. In order to have an up-to-date source of sensor readings, the device reading the sensors must be able to connect to that source correctly. The garden also needed to receive new configurations for how to best care for the plant from the database.

To help with the testing of the server we implement unit testing. Unit testing allows us to test each of the individual units of our software. This first-level testing method is a good way to verify that things are working as they should be. We do this by writing code to run on the server then validate that the proper result are generated. While doing this we were careful to make sure that we tested for a variety of possible values, including positive and negative values. We also made sure not to ever make assumptions while testing. This method lets us test our software at an initially simplistic level. This lets us make sure the most basic functions of the software work. Once we can validate that the more critical and basic functions operate properly, we can then test the more complex and involved elements of the software. This is also convenient and effective because it is relatively automated. It can be set to run and the results can be analyzed later. This is very time efficient for the test as they can work on other aspects of the project while this test is being run.

8.2.3 Website Testing

While the website itself will require some effort to create and practice to implement, at its core it will simply be a frontend and a backend working together to send an API call to a server that the ATMEGA must also send similar requests to. With the previous section proving that this is doable from the ATMEGA, this section will focus on the webpage that the garden’s users will interact with. Testing the website was made much easier by printing certain code benchmarks at different points in the Javascript, since it is helpful to know at which point something went wrong. Functionality was added bit by bit, with some features being visually simplistic or without function, such as creating a dropdown menu with only a few options in it, and clicking a button simply printing the value of the currently selected option. As development continued, more and more things were automated, such as filling an initially-empty dropdown menu based on the value of the previously mentioned ‘plantprefs’ table. Slowly building up to things in such a manner helps isolate which component is failing if troubleshooting is needed. It is hard to select an aspect of the website to test in order to call functional besides simply “everything works”. Each component, however, can be tested to ensure it is still functioning, and that no outside changes have somehow interrupted its behavior. With much of html being visible even without meaningful content, the visual aspect of it could largely be tested by loading the files in a local server, and some functionality with the database could still be tested due to the way html files retrieve their source Javascript and php files from hard coded URLs.

Below is a table showing the various systems that need to be tested to confirm whether or not they have been implemented properly for the function of the project. Also presented is the goal of including said system and a condition that, when met, indicates the minimum result needed to verify successful operation of it.

|  |  |  |
| --- | --- | --- |
| System | Goal Desired | Success Conditions |
| Wi-Fi Module | The garden needs to be able to communicate with Wi-Fi enabled devices using a Wi-Fi module. | Sending an http request with AT commands created and sent through code. |
| Database (read) | The ATMEGA can react to changes in a database and alter its behavior as instructed. | The garden accesses the database via an http GET request and enacts a local change (LED) based on that information |
| Database (update) | The ATMEGA can change files in a database so that the user can see the latest information about the plant. | The Arduino is able to send data to a php file. If the phps can receive this data, we can control how it is used. |
| Website | The server can receive and react to API calls from an html-based interface, both providing information and changing the content of the database. | If all individual components on the website work as they are meant to, each of which are relatively simple on their own, the website will be considered functional. During testing, faulty interactions can be excused temporarily. |

Table 20: Software Test Parameters

9 Project Operation

The purpose of this section is to explain how the product is to be used. This section is intended to explain the functions this device offers and how to properly use them. Section 9.1 briefly enumerates the functions available, and section 9.2 explains how to utilize them. 9.2 is effectively the user’s manual.

9.1 General Information

Smart garden is a remote gardening-assistance system that provides the ability to:

* Measure moisture levels of the soil
* Water and feed plants remotely as needed
* Measure temperature and humidity of plant environment

9.2 Using the Device

Once the SMART garden is acquired it will need to be assembled. This involves mounting the food and water containers as well as the power supply housing unit. The power source will need to be connected to the power port through an AC/DC adapter. The water and plant food storage units will need to be filled and the microprocessor will need to be configured. After the smart garden is properly assembled, it will need to be powered on and the settings calibrated. Additionally, the main compartment must be filled with soil to hold the plant, and the plant must be planted in this soil.

The SMART Garden is always turned on while connected to the power source. Once powered on it will need to be paired with the desired Wi-Fi connection such as a router or mobile hotspot in order for it to be able to perform its functions remotely. If the system is missing any of the necessary resources to perform its functions, it will notify the user. If the water tank or plant food containers are empty, then the user will need to refill them before the functions can proceed.

Smart garden is connected wirelessly to the computer or mobile device via Wi-Fi. The application will give a visual indicator to notify the user once the connection to SMART garden is established. The user will use their computer to select the ideal values of their plant from our list of enter custom values. The ideal values will be uploaded to the remotely-hosted web server. These ideal values will be transmitted to the microcontroller which will carry out watering based on these values. The ideal values and most recent sensor readings will visible from the website. Ideal values can be changed at any time, however they may only be acquired on the following cycle of checking and maintaining.

Every ~18 seconds, which is how long it takes to ensure each and every individual step in the code has time to run and receive a response, the garden will receive the most recent ideal values for the user’s chosen plant from the web server. It uses these ideal values to know when to water the plant. If the moisture percentage is too low, it will activate the water pump for a short time. After a cycle where this happens, it will abstain from watering for a few cycles, allowing the water time to soak through the soil, which results in the soil moisture sensors receiving a value true to how much water is actually in the soil, rather than accidentally overwatering the plant due to not waiting long enough for a realistic reading.

Each cycle also includes the garden sending the most recent sensor readings to the web server. From the website, you can see the most recent sensor readings and change the current ideal readings, in case you decided to grow a different plant in the garden, or if you just want to give your plant a custom name!

10 Administrative Content

This section details some of the methods used to manage the development of this project and its members. It will list milestones, member roles, budget decisions and an over summarization of the project process. The budget section will include all the financial data related to this project. It will be updated as the process goes on. The project role section will detail what aspects of the overall project each member completed. The final part of this section will list the goals for the project, the problems encountered, and how they were resolved.

Some of the issues we faced involved reaching the required page count for the Senior Design I report. We initially divided the report into sections and had individual members pick up sections they felt knowledgeable in. They would write as much as possible for the selected section then move on to the next one. This was effective but it was still difficult at time to make the word count. To supplement this process, we started to add to sections that others had selected. This did make tracking each individual's contributions a little more complicated but overall, it increased the amount of content that was added to the paper, which was the overall goal. Even when not directly contributing to a section, team members were ready and willing to offer ideas to the member that was working on a section that they were struggling with.

Another task the presented some issues was the selection process for the parts to be used in our project. For some of the parts we collectively had little experience working with such equipment. As a result, we had to do extensive research to determine which parts would work best for us and our project. Fortunately, we had multiple similar projects to reference for our own. After doing other research and reading what other people had to say about their experiences in designing their rendition of this project, we were able to make our best-informed decisions.

A number of issues presented themselves when it came to actually assembling the prototype for the Smart Garden. While ordering the parts and writing the paper it seemed like the project would be fairly simple. As the process went along, we started to realize all the issues we didn’t consider before.

10.1 Milestone Discussion

This section will be constantly updated as the project continues.

The main communication for the team’s objectives and deadlines was done through Discord. We made specific sections the organize the different facets of project. We had channels that highlighted resources that could be useful to everyone in the group. We had a channel where we listed any ideas we had that could improve the project so that even if the idea’s lister couldn’t follow up on the execution of it, someone else might be able to. We even had a channel that helped us prevent reiteration of ideas throughout the paper by giving us a space to tell what concepts we were righting about that might be relevant to other sections in the paper.

To keep us on track we would meet two times weekly to assign members tasks and make sure everyone is keeping up to pace. The meetings would usually start with each member updating the team on any progress made since the last meeting as well as stating what they planned on getting done by the next meeting, or other date, and how they planned to go about completing that task on time. As we went through this process each member would do their best to offer ideas and help to whomever needed it. Some of this help included offering up ideas on what to write about. This included new topics that had not yet been covered that would be good to cover. This also included new aspects of topic that were already being covered that could be covered in more detail. Such as subtopics that fall under that overarching topic.

The first major milestone was completing the Divide and Conquer paper for SD1. This paper was relatively simple and as such we had no problem completing it on time. Despite our heavily conflicting schedules, we managed to find a time to meet in person and go through this paper. This is where we finalized our project selection and began doing a very rudimentary level design for the project. This involved a very basic list of objectives we wanted this project to achieve such as it being portable or easy to use. We also established what basic functions we wanted it to be able to perform. Furthermore, we were able to determine a very rough estimation of the budget for the project as well as designing some basic block diagrams for the project.

The next major milestone was having a meeting with Dr. Richie and getting our project design approved. To be thorough we had two different project ideas lined up just in case Dr. Richie thought our project needed to be more complex. Both of the ideas were laid out in a relatively basic fashion as it would not be an efficient use of time to develop an idea that we were not going to following through on. After meeting with him it was determined that our primary project idea, the SMART garden, was not only an acceptable idea but a common one as well. He offered us ideas and reference documents for inspiration and guidance. We took the advice he offered us from the meeting to aid us going forward.

The third major milestone was the second and final Divide and Conquer paper where we solidified our project idea and had a much more developed idea on how we wanted to go about creating this project. With them we were ready to really dig in and start working on bringing this project to life.

The fourth major milestone was our first major paper submission. This submission was the 30-page rough draft. While the thirty pages wasn’t too much to handle given the timeframe allotted, our time did get consumed with other schoolwork and we did have to do much of the work close to the due date. Even though this was stressful we did still meet the deadline with an adequate submission.

The next milestone was similar to the last one just with additional pages. Unfortunately, similar to the previous deadline, other work occupied our time and we did, once again, have to put in an extensive amount of work over a short period of time, but such as was the case with the previous deadline, we did meet our deadline with sufficient work.

The one-hundred-and-twenty-page deadline seemed to be a little more challenging than the previous deadlines. This was due to a combination of needing a substantial number of pages and still being relatively early in the senior design process. Given that, many of the relevant topics we intent to write about haven’t happened yet. This means that many of the topics we will be writing about will need to be filled in once those topic occur. This includes things such as the assembly process and the complications that arise from that. This did however force us to think about the topics that were available more intensely.

For the “Begin part acquisition” deadline we did work ahead. The team did look at parts they felt would be adequate whilst writing the previous papers. When we finally met to buy the parts there wasn’t much searching to be done. We already had options lined up for what we wanted. It was just a matter of selecting which options we preferred and making sure they would be compatible with each other. There were some part options that we did not fully finalize yet as there are still some functionalities that we are still not completely sure how we are going to implement. In addition to that we also have some functionalities that we are not completely certain we want to implement as they may not be necessary.

The next major milestone is actually assembling the system and making sure it’s physically functional. This includes wiring all the components and programming the microprocessor to perform all the necessary operations, as a prototype.

10.1.1 Senior Design 1 Milestones

In this section we will be going over the key milestones we hit during the Fall 2022 semester in Senior Design 1, as well as detail out any remaining tasks that still need to be completed. This chart will be referenced throughout the semester and updated as tasks are completed. The distribution of tasks throughout the semester will be reflected on at the end of Senior Design 1, and an improved planning strategy will be put into place during Senior Design 2. The purpose of this is to learn from the mistakes we made this semester, and ensure we allot adequate time for the completion of both individual and group tasks when moving forward into our project build phase.

|  |  |  |  |
| --- | --- | --- | --- |
| **Milestone** | **Task** | **Due Date** | **Status** |
| 1 | Divide and Conquer 1.0 | 9/9/22 | Completed |
| 2 | Topic approved by advisor | 9/21/22 | Completed |

Table 21: Senior Design 1 Milestones

|  |  |  |  |
| --- | --- | --- | --- |
| **Milestone** | **Task** | **Due Date** | **Status** |
| 3 | Divide and Conquer 2.0 | 10/7/22 | Completed |
| 4 | 30-page draft complete | 10/21/22 | Completed |
| 5 | 60-page draft submission | 11/4/22 | Completed |
| 6 | 80-page draft complete | 11/11/22 | Completed |
| 7 | 100-page draft submission | 11/18/22 | Completed |
| 8 | Begin part acquisition | 11/17/22 | Completed |
| 9 | 120-page requirement met | 12/2/22 | Completed |
| 10 | Final paper formatting and editing complete | 12/5/22 | Completed |
| 11 | Final Report submission | 12/6/22 | Completed |

Table 22: Senior Design 1 Milestones continued

10.1.2 Senior Design 2 Milestones

In this section we will be going over the key mile markers have hit or plan to hit during the Spring 2022 semester in Senior Design 2. Similarly, to Senior Design 1, this chart will be referenced throughout the semester and updated as tasks are completed. Bi-weekly group check-ins will be utilized to make sure tasks are being met and completed in a timely manner.

|  |  |  |  |
| --- | --- | --- | --- |
| **Milestone** | **Task** | **Due Date** | **Status** |
| 1 | Finalize ordering parts | 12/16/22 | Completed |
| 2 | Hardware Testing for  Individual Components | 1/19/23 | Completed |
| 3 | Begin physical project build | 1/24/23 | Completed |
| 4 | Edit and update SD2 document with all changes made to project | 1/26/23 | Completed |
| 5 | Software Testing | 1/31/23 | Completed |
| 6 | Solidify hardware and software connection | 2/9/23 | Completed |
| 7 | Have fully functional web application | 3/2/23 | Completed |
| 8 | Product build and design completed | 3/9/23 | Completed |
| 9 | Edit and update SD2 document with all changes made to project | End of SD2 | Completed |
| 10 | Begin building our Senior Design Group Website | End of SD2 | Completed |
| 11 | Final Product Testing | End of SD2 | Completed |
| 12 | Prepare Final Presentation | End of SD2 | Completed |
| 13 | Final SD2 document submission | End of SD2 | Completed |
| 14 | Final presentation slides/video submission | End of SD2 | Completed |
| 15 | Final Presentation | End of SD2 | Completed |

Table 23: Senior Design 2 Milestones

10.2 Final Budget and Finance

The SMART Garden project was fully financed by our four group members. The goal of our group was to not exceed a total expenditure of $400. When choosing parts for our project we looked to find the best fit for our project for the cheapest overall price. In this section, there is an in-depth overview of all components and parts we plan to buy or have already bought for the SMART Garden system. These prices, quantities, and items are subject to change and will be updated as necessary throughout the duration of Senior Design 2.

|  |  |  |  |
| --- | --- | --- | --- |
| **Item** | **Price** | **Quantity** | **Total** |
| Arduino Uno | $28.50 | 1 | $28.50 |
| PCB Board (estimated) | $100 | 1 | $100 |
| Temperature and Humidity Sensor | $10.35 | 1 | $10.35 |
| Soil Moisture sensor | $7.89 | 5 | $7.89 |
| UV Sensor | $3.00 | 1 | $3.00 |
| OLED Display | $6.99 | 1 | $6.99 |
| Wi-Fi Module | $7.50 | 1 | $7.50 |
| 12V Water Pump | $15.99 | 2 | $15.99 |
| Water Pump tubing (10 ft) | $11.99 | 1 | $11.99 |

Table 24: Final Budget

|  |  |  |  |
| --- | --- | --- | --- |
| **Item** | **Price** | **Quantity** | **Total** |
| 5V Dual Channel Relay Module | 6.79 | 1 | $6.79 |
| 9V Rechargeable Batteries | $17.99 | 2 | $17.99 |
| 9V Battery Holder | $5.99 | 2 | $5.99 |
| Solar Panel | $24.49 | 1 | $24.49 |
| Keypad | $3.95 | 1 | $3.95 |
| Breadboard (already owned) | $0.00 | 1 | $0.00 |
| LED lights | $5.99 | 100 | $5.99 |
| 220-ohm resistors | $5.49 | 100 | $5.49 |
| Wires/Jumper Cables (already owned) | $0.00 | 120 | $0.00 |
| Multimeter (UCF provided) | $0.00 | 1 | $0.00 |
| Water Reservoir Container (already owned) | $0.00 | 1 | $0.00 |
| Miracle-Gro Indoor Plant Food | $6.29 | 1 | $6.29 |
| Soil (4 quart) | $14.99 | 1 | $14.99 |
| Herbs (Cilantro, Basil, Oregano) | $9.99 | 1 | $9.99 |
|  | | **Final Cost** | **$294.17** |

Table 25: Final Budget Continued

10.3 Bill of Materials (BOM)

In this section we will be going over our Bill of Materials (BOM). This table depicts the components already purchased by the members of our group. The BOM serves the purpose of keeping track of overall expenditures to make sure we stay within our desired budget.

|  |  |  |
| --- | --- | --- |
| **Number Label** | **Part Name** | **Price** |
| 1 | Arduino UNO REV3 [A000066] | $28.50 |
| 2 | DHT22 Digital Temperature and Humidity Sensor | $10.35 |
| 3 | HiLetgo 5pcs LM393 3.3V-5V Soil Moisture Detect Sensor | $7.89 |
| 4 | DC 12V Mini Submersible Water Pump | $15.99 |
| 5 | UCTRONICS 0.96 Inch OLED Module 12864 128x64 Yellow Blue SSD1306 | $6.99 |
| 6 | SunFounder 2 Channel DC 5V Relay Module | $6.79 |
| 7 | LAMPVPATH 9v Battery Holder | $5.99 |
|  | **Total Cost** | **$82.50** |

Table 26: Bill of Materials

10.4 Project Roles

In this section we will be going over the main roles of each member of the group within Senior Design 1. It is important to note, that there is some overlap within group members roles. All members of the group worked together to complete this project and made it a point to help out their group members whenever needed. This list is not fully comprehensive but serves as a general overview of workload distribution.

Jonathan Wallhauser

* Sensors
  + Soil Moisture
  + UV Light
  + Temperature
  + Humidity
* Relay Module
  + Multiple Single Channel
  + Single Multiple Channel
* LCD Display
  + OLED
  + LCD
  + Capacity
  + On-Board Controller
  + Pinout
  + Communication
    - UART
    - SPI
    - I2C
* Solar
  + Schematic
  + PV

Kevin Lorch

* Communication Devices
  + Wi-Fi
  + Bluetooth
* Software Development
  + Software Testing
    - Testing per service
  + Version Control
* Power Supply
  + Battery
  + Solar Panels
* Realistic Design Constraints
  + Economic Constraints
  + Time Constraints
  + Environmental and Health Constraints
  + Possible Solutions
* Coding Standards
  + Variable Names
  + Formatting

Ryan Hassan

* Software Design
  + Website Structure
  + Remote Server
  + Front-end/Back-end Communication
* Mobile App Development
  + OS selection
* Software integration
  + Front-end/Back-end Communication
  + Server selection

Lauren Melancon

* PCB
  + Board Design
  + Manufacturing
* Microcontroller
* Hardware Testing
  + Microcontroller
  + Sensor Testing
    - Soil Moisture
    - UV Light
    - Temperature and Humidity
  + OLED Display
  + Power Supply
  + Relay Module
  + Water Pump

Everyone

* Documentation
  + Section Breakdown
  + Cross Referencing
  + Peer Review
* Part Acquisition
  + AdaFruit
  + Amazon
  + SparkFun
  + Arduino Store
* Communication During Research and Development Phases
  + Discord
  + In-person
  + Tag-ups
  + Scrums
  + Sprints
* Overall Project Design
  + Goals
  + Engineering Specifications
  + Engineering Requirements
  + SMART Garden Design
  + SMART Garden Feature Set

10.5 Content Distribution

* Part Acquisition
* Overall Project Design

10.5.1 Organization

The team had a handful of different techniques for keeping their AGILE workflow methodology in a clear and concise manner. This included the main form of inter-team communication, Discord. Discord is an all-in-one communication application that is widely used in the world of engineering academia at the University of Central Florida. All of the engineers on the team were familiar with Discord. This made it clear that Discord would be advantageous to use over other applications like Slack or Microsoft Teams.

**Text Channels**

The first main section in the Senior Design group 34 design documentation is the text channels. In the engineers’ Discord, there were 15 main channels. These channels were broken down into the following for clearest levels of communication:

* General Text Channels
* Specific Resources
* Citable Sources

With these three main channel groups, the engineers were able to relay information clearly under the general text channels. Information like ideas, basic resources, meetings, and others were found under this channel group. These channels were open to all of the engineers, and everyone contributed to communication in these tabs. The complete channel breakdown for the General Text Channels is shown in figure 20.

In the ‘Resources’ group, an engineer could post links that would help when working on specific aspects of the project. Webpages about html documents Javascript syntax would be posted in the ‘website-programming’ channel, for instance. The ‘general-resources’ channel held information that every team member could utilize, such as the password to the web server’s Linux interface, as well as the codes to run if we needed to replace the current website files with the ones in our Github repository.

The last channel group, Citable Sources, was more of an individual source. Each engineer had their own citable sources channel. The engineers added any sources they used during the research phase in their channel, respectively. The complete channel breakdown for the Citable Sources channel group is shown in figure 20.

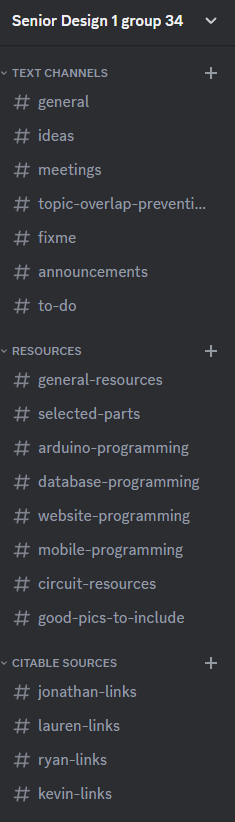


Figure 20: Organization of Discord text channels

Additionally, Discord was used to make sure every engineer was using their time effectively. The ‘topic overlap prevention’ text channel was used to make sure nobody was working on a section that had too much similarity to another section. When selecting topics for each team member to take on, this was used to ensure that everybody was on the same page about who was working on each section. These conversations also happened during the face to face meetings, but for asynchronous collaboration, such methods are often necessary. A section of this channel has been included below as figure 21 to illustrate the use case for this channel and illustrate how it can help distribute the workload.

Text

Description automatically generated

Figure 21: Example of the 'topic overlap prevention' channel

10.5.2 Virtual Communication

The engineers of SMART Garden had bi-weekly meetings, and one of the two meetings each week was conducted virtually. This optimized the workflow of the team, allowing for more leniency and greater volume of Scrums and working sessions. The engineers used Discord for this form of communication as well. However, there was no need for more than a single voice channel for 4 team members. Therefore, we only had a single voice channel named “General” as our channel that the engineers used for virtual voice communication for the biweekly scrums, quick tag ups, and other necessary calls. The Voice channel looked like the following:

Graphical user interface, text, application, chat or text message

Description automatically generated

Figure 22: Virtual Communication - Voice Channel

11 Components Not Used in Final Product

Some of our aspirations for the SMART Garden never came true for one reason or another. This section will discuss those reasons for each component that was omitted from the final design.

11.1 Batteries and Solar Panels

The SMART Garden was designed for use by people who already don’t have the space and/or free time to tend a regular garden, so these two options were among the first to go. If a user is busy most of the day, they likely wouldn’t be around to replace the batteries if it is needed. Additionally, if the garden was left outside and inclement weather began, a busy user wouldn’t be able to take time out of their day to move the garden indoors, and checking the weather for the day to accommodate a SMART Garden would just be another task that could fall by the wayside.

With the decision to focus the function of the design to be usable primarily indoors, the decision was made to instead power the garden with a 9V AC/DC adapter so that the user could provide constant power to the garden without interrupting their day or risking that the plant go without water. Since the garden would now remain indoors and have constant power, solar panels no longer served a purpose in this project. There would be no downtime to save the garden from, nor any battery life to supplement. Thus, solar panels were removed from the project, and didn’t need replacing.

11.2 UV Sensor

For the same reason as the solar panels above, the UV sensor was stricken from the design of the SMART Garden. Additionally, the options we found for this sensor in the level of accuracy we desired were quite expensive, and that much cost for so little benefit wasn’t something we wanted to inflict on the users of the garden. We removed the UV sensor, but the code still had traces of it. Anywhere we store ideal UV values, those are used for the low end of the ideal temperature range. Sensor readings are discarded on the server side of things, and the ATMEGA sends ‘sensor readings’ set to a default value.

11.3 OLED Display

Our original plan was to have a display so that the user could see the readings on the display rather than having to view the website to receive the latest sensor readings. Unfortunately, we ran into a roadblock during implementation. The local variables in the sketch the garden runs on were beginning to take up too much dynamic memory, and the variable required to run the screen took up nearly one third of the dynamic memory on the ATMEGA. Without enough space to store variables, the data of the Strings used to send data to the website didn’t have enough space to exist. We prioritized the main function of the garden over user experience, even though the individual pieces worked on their own. If we were to redo this project, one thing to change would be to use a processor with more dynamic memory so that we could avoid this kind of issue.

12 Appendix

12.1 References

Gerber Labs <https://www.gerberlabs.com/10-tips-and-tricks-for-printed-circuit-board-design/>

Bluetooth SIG <https://www.bluetooth.com/learn-about-bluetooth/tech-overview/>

Wi-Fi Alliance <https://www.wi-fi.org/certification>

Karl Söderby <https://docs.arduino.cc/software/ide-v2/tutorials/getting-started-ide-v2?queryID=6224c3a7cad5ed4b836eb73698938a3b>

Karl Söderby <https://docs.arduino.cc/software/ide-v2/tutorials/ide-v2-installing-a-library>

Arduino <https://docs.arduino.cc/hardware/uno-rev3>

Arduino <https://docs.arduino.cc/hacking/software/Bootloader?queryID=7f177702bc2f6afe8cf66c3f967efc24>

Arduino <https://docs.arduino.cc/hacking/software/Programmer>

Arduino <https://docs.arduino.cc/retired/getting-started-guides/ArduinoUnoWiFi>

Circuit Basics [How to Set Up UART Communication on the Arduino - Circuit Basics](https://www.circuitbasics.com/how-to-set-up-uart-communication-for-arduino/)

Dejan, SPI [How I2C Communication Works & How To Use It with Arduino (howtomechatronics.com)](https://howtomechatronics.com/tutorials/arduino/how-i2c-communication-works-and-how-to-use-it-with-arduino/)

Jeff Paredes [How I2C Communication Works & How To Use It with Arduino (howtomechatronics.com)](https://howtomechatronics.com/tutorials/arduino/how-i2c-communication-works-and-how-to-use-it-with-arduino/)

Project Hub [Add WiFi to Arduino UNO - Arduino Project Hub](https://create.arduino.cc/projecthub/imjeffparedes/add-wifi-to-arduino-uno-663b9e)

Microcontroller [MSP430](https://www.ti.com/product/MSP430G2231?utm_source=google&utm_medium=cpc&utm_campaign=epd-null-null-GPN_EN-cpc-pf-google-wwe&utm_content=MSP430G2231&ds_k=%7b_dssearchterm%7d&DCM=yes&gclid=CjwKCAjwh4ObBhAzEiwAHzZYU6lz6RFpqKZE0MKd2_sQYYwKNV5yFHbmek61vhtrvGLef_RXK014rRoCDDQQAvD_BwE&gclsrc=aw.ds)

DHT22 [Interfacing DHT11 and DHT22 Sensors with Arduino](https://lastminuteengineers.com/dht11-dht22-arduino-tutorial/)

Water Pump [Control a water pump by Arduino](https://www.robotique.tech/robotics/control-a-water-pump-by-arduino/)

PCB [10 Top-Rated US PCB Manufacturers](https://www.raypcb.com/us-pcb-manufacturers/)

STM32 [Blue Pill Development Board](https://components101.com/microcontrollers/stm32f103c8t8-blue-pill-development-board#:~:text=37%20General%2Dpurpose%20I%2FO%20pins.)

UDO Bolt [Datasheet](https://www.seco.com/Datasheet/SBC/UDOO%20BOLT_datasheet.pdf)

Raspberry Pi 4 [Datasheet](https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-4-Product-Brief.pdf)

PCB [Design Software](https://fixthephoto.com/best-free-pcb-design-software.html)

Design Software [Fusion 360](https://www.autodesk.com/products/fusion-360/features)

UV Sensor [Interfacing ML8511 UV Sensor Module with Arduino](https://learn.sparkfun.com/tutorials/ml8511-uv-sensor-hookup-guide/all?print=1)

Arduino [Arduino Pinout](https://www.circuito.io/blog/arduino-uno-pinout/)