Spring 2022

Smart UV-C Disinfectant Module

Nicole Baldy
The University of Akron, neb45@uakron.edu

Luke Rogers
The University of Akron, ljr47@uakron.edu

Haitham Saleh
The University of Akron, hhs10@uakron.edu

Smart UV Disinfectant

Final Design Report

Design Team 12

Nicole Baldy

Luke Rogers

Haitham Saleh

Faculty Advisor: Dr. Jobeda Khanam

04/12/2022

Smart UV Disinfectant

Nicole Baldy Contributions

Nicole's primary responsibility was the control subsystem – this includes the microcontroller and its interactions and control of all sensors and actuators. She designed and implemented the primary state machine which ensured that sensors were checked and the data processed correctly according to the overall device state. She also implemented the software interface between the microcontroller and the Grid-EYE sensor which was used to accurately detect the presence of a human from their body temperature. Nicole also served a secondary role as an "integrations specialist"; thanks to her co-op experience in robotics, she provided expertise in planning the subsystems around integration. For example, user input was first simulated using a button on the microcontroller, which was later replaced by a signal from the phone application.

Smart UV Disinfectant

Luke Rogers Contributions

Luke's primary responsibility was the user interface subsystem – this includes the UART communication between the microcontroller and the Bluetooth board, the Android phone application, and the data sent between the microcontroller and the subsystem. He designed and implemented the phone application and provided the software interface to the Bluetooth chip for the microcontroller, which made integration with the user interface subsystem extremely simple. Luke also served a secondary role as the "mechanical specialist"; thanks to his experience in mechanical systems from the Formula Combustion Design Team, Luke planned and provided the locker support beams and motor mount among other small mechanical needs.

Table of Contents

Table of Figures

Table of Tables

Abstract

Ultraviolet (UV) light is frequently used to quickly disinfect surfaces, particularly those which cannot be easily washed. However, the more effective disinfecting wavelengths are hazardous to eyes and skin, potentially causing severe sunburn and retinal damage. This paper proposes an enclosed disinfection device which will disinfect objects such as backpacks, allowing one user to control the device at a time through their phone. It will fully disinfect objects in 5 minutes or less and provide many safety features to prevent it from turning on when a person or animal would be exposed to the UV light. The proposed device will be very easy to use; it can be plugged in to a standard wall outlet and controlled wirelessly via a phone application.

1. Problem Statement

1.1. Need

COVID-19 has made the world much more complicated – especially the use of public spaces, where many people touch the same surfaces and breathe the same air. The FDA states that a type of ultraviolet (UV) light, UV-C radiation (wavelength of 190–290 nm), "is a known disinfectant for air, water, and nonporous surfaces." While the effectiveness of UV-C against COVID-19 is still under study, it has been proven effective for other Coronaviruses, such as the SARS-Coronavirus (Kitagawa 299–301). Specifically, UV-C light is the most effective light of the UV spectrum at killing pathogens, but it can also be damaging to human eyes and skin due to its high energy. According to the Skin Cancer Foundation "About 90 percent of nonmelanoma skin cancers are associated with exposure to ultraviolet (UV) radiation from the sun". UV-C radiation from germicidal lamps is similar to sunlight and is therefore can be a health hazard if too much exposure occurs. There is a need for UV-C disinfectants which can be used to disinfect public areas but with safety features that prevent unnecessary human exposure to dangerous portions of the UV spectrum.

(NB, LR, HS, KL)

1.2. Objective

The objective of this project is to create a device that can disinfect an object by emitting UV-C while limiting human exposure. The light intensity should be high enough to sanitize, and it should have safety features which prevent exposure to the LED. This device will be a touchless device which allows an object to be placed inside and sanitizes

the object, while ensuring that there is not a pet or hand exposed to the light. During each light cycle, the entire surface of the object should be exposed to UV-C light. The device should also have a user interface which provides remote access to operate the light and provide notifications if any maintenance on the system is required. (NB, LR, HS)

1.3. Background

Ultraviolet germicidal irradiation (UVGI) is the use of ultraviolet (UV) light to disinfect air or surfaces. Ultraviolet light is defined as having a wavelength between 10nm and 380nm; wavelengths of 190 – 290nm are generally referred to as "UV-C" and are most often used in UVGI systems (Kesavan and Sagripanti). The effectiveness of UV light on pathogens depends primarily on the light intensity, wavelength, and exposure time. A study by Kim and Kang used a UV LED (Light Emitting Diode) array, which produced wavelengths of 200 – 400nm, to eliminate pathogens from the air. It observed a 5-log reduction (0.001% of the initial amount) of all 3 viruses being tested after 10 minutes of exposure to the array. Several bacteria and fungi were also tested, with results varying from a 2.5-log reduction to a 4-log reduction (0.32% - 0.01% of the initial amount). Another study evaluated the effect of UVGI on multiple surfaces after 10 minutes of exposure. It found at least a 3.8-log reduction (0.015% of the initial amount) in the two tested antibiotic-resistant strains of bacteria if exposed to at least 9 mJ/cm^2 of UV light (Jelden, Katelyn C., et al. 6). These two studies show that UGVI can effectively kill pathogens both in the air and on surfaces. Due to the current pandemic, many studies are being conducted on UVGI for its effect on COVID-19, as UVGI is the primary technology for eliminating pathogens from the air, according to Kesavan and Sagripanti. A study by Buonanno et al. specifically examines the use of 222-nm light for deactivating COVID-19 and shows that doses of less

than 2 mJ/cm^2 deactivated 90% of the virus within 8 minutes, and 99.9% within 25 minutes ("Far-UVC Light" 1). From these results, it can be seen that UVGI technology eliminates a significant number of pathogens, including COVID-19, from both the air and exposed surfaces in a relatively short period of time, and makes this a promising technology for use in disinfecting public areas. (NB)

The drawback to this technology is that UV-C wavelengths are considered hazardous, according to the Handbook of Occupational Safety and Health. UV-C light can cause eye and skin damage such as sunburn of the cornea (also known as photokeratis), cataract formation, or skin pathology (Haes and Galanek). According to the International Commission on Non-Ionizing Radiation Protection, the exposure limit to UV wavelengths between 180 and 400nm is 30 J/m^2 per 8 hours to the eyes or skin without any UV protection (174). However, a 2017 study by Buonanno et al. showed no statistical difference in the amount of skin damage from exposure to 222-nm light from the control ("Germicidal efficacy and mammalian skin safety" 493). Using this wavelength while adhering to the exposure limit of 30 J/m^2 in a sanitizing device should provide the greatest amount of safety possible while still acting as an efficient sanitizer. To ensure that a device emitting UV light adheres to the exposure limit, the smallest effective dose should be used (as seen previously, this is 2 J/m^2) and it should stop emitting light if a person is detected nearby. (NB)

Currently, there are several types of devices which use UGVI to disinfect rooms. The simplest application is a UV-C light bulb which can be put any light fixture which meets the power requirements. This is problematic, however, as there are no protections to using these bulbs – a person may be exposed to a large amount of UV-C light, especially

if they are unaware that the light is not a standard lightbulb or of the hazards of UV-C. There are also handheld UV wands meant to sterilize a small area, but these have the same safety hazards as a pure lightbulb if used improperly and need require manual use – a person must move the light around to cover an area. UVGI is also used in many air purification systems where air is pumped through a device containing UV light as a disinfectant, but no light reaches outside of the device. Such devices cannot clean surfaces and are designed to sterilize only the air pumped through the device. This is useful, but sterilization of both surfaces and air is ideal in a public environment. There are also automated applications which are designed to sit in a room or are attached to a mobile robotic platform to cover an even larger area, in a growing area known as robot-assisted UV disinfection (McGinn 2). However, these more autonomous applications are very expensive and primarily marketed to large hospitals. This proposal focuses on a design with many of the benefits of one of these high-end robotic products – person-sensing ability, object disinfection, usage data collection, maintenance notifications, and some movement ability data– but at the cost closer to a consumer-grade UV air purification system. This allows the device to be used in a variety of high-traffic areas such as public airports, such that a person could put a bag inside to be sanitized. (NB)

1.3.1. Existing Designs

Several patents and patent applications have a similar design which use UVGI combined with a sensor system to maximize user safety and disinfection efficiency. One patent application, “Device for lighting ultraviolet sterilization lamp without detection of human sensor and by lights-out of lighting device” is a UV light which was meant for a kitchen or bathroom and detected if a person was nearby. It also measured the humidity in the room and contained an “odor sensor [sic]” and would activate if humidity was high

enough or the “odor sensor [sic]” was activated, but a person was not detected. It would also sound an alarm before activation and used a controller which a person could use to activate the light. This application was rejected in 2017 (Nakao). The UV-C light module being proposed in this paper is similar to Nakao’s patent application in that it has human-sensing ability and will accept user control, but does not have a complex mold-sensing ability, as it is more focused on destroying pathogens than suppressing mold. Another patent application from 2014, “Automatic sterilization device for public toilet”, used a UV band, heat sensors, infrared sensors, and a “gate detection board” which detects whether the stall door is open. These sensors allowed the device to detect if a person is within the stall, which deactivated the light. This application was withdrawn in 2015 (Chen). The proposed module is very similar to Chen’s patent application, with some improvements. First, the use of the Grid-Eye sensor discussed previously will provide very accurate data on whether a person is present nearby. Second, rather than a UV band, the proposed design will use a 222nm light or similar wavelength which will minimize risk to eyes and skin while still being an effective disinfectant. There are also approved patents which are very similar to the proposed device. “Portable light source including white and ultraviolet light sources” is a patent granted in 2007 which uses UV and white light emitted from pivoting arm to disinfect a large area and is powered by a rechargeable battery (Hopkins). Rather than having the UV light source on the end of an arm like Hopkin’s patent, the proposed module will have static light sources and rotate the platform holding object within the sanitizer. This allows us to more easily enclose our module to increase safety. With these modifications, the proposed device will be as safe as possible while also being able to disinfect an object in a relatively short time. (NB)

1.4. Marketing Requirements

The design described in this report should meet a number of marketing requirements to ensure that it is a functional product, capable of filling the identified market need. These marketing requirements are as follows:

Table 1: Marketing Requirements Table

1) The device should kill a significant number of pathogens in a short time.
2) The device should be able to sanitize backpacks and similarly sized objects.
3) The device should interface with user's phones and remain locked until the user is ready to access their belongings.
4) The device should be ready to use shortly after being plugged in.
5) The device should have numerous safety measures and precautions to prevent injury to users or animals.

(NB, LR, HS, HSA)

2. Engineering Analysis

There are several aspects of the design which require further analysis to ensure that the requirements can be achieved; these are explored in more detail below. The device can be broken down into 3 major subsystems: the user interface, which consists of the android application and wireless transmitter; the control system, which consists of the microcontroller and signal connection to the other electronics; and the electrical system, which shall consist of all electrical components to be used.

2.1. Background Research on Circuit Components to Fulfill Requirements

Additional background has been explored to narrow down components for the project based on these requirements.

One requirement for the project to operate safely is a sensor that is able to detect humans and pets. There are different types of sensors that were considered, but it

was concluded that an infrared sensor was appropriate for the project and provided the accuracy needed to keep users safe. The infrared sensor is widely used in industries for the purpose of human detection. Humans and pets radiate infrared energy which correlates to temperature, and that temperature with respect to a reference is sensed and then is interpreted as a voltage signal by the sensor. From the research paper "Detection and tracking of a human using the infrared thermopile array sensor — 'Grid-EYE'," two types of infrared sensors were considered. The passive infrared sensor (PIR) was first considered, however it is only able to detect humans that are in motion. This sensor would not be appropriate for the project as sensing of a person should not be reliant on motion. The other type of infrared sensor, Grid-EYE, is a better fit to the project since it is able to detect humans and pets both in motion and in static positions using an infrared sensor. The Grid-EYE sensor is also able to detect the direction of motion in a 2-dimensional coordinate system, though this functionality will likely be unnecessary for the proposed device. The Grid-EYE sensor covers moderate distances that are appropriate for the project in mind. According to Shetty et al., "Grid-EYE sensor has field of view of 60 degrees in both horizontal and vertical directions and maximum recommended distance used for detection is 5 meters" (1490). In a Grid-EYE sensor, thermopile heat detectors are constructed out of thermocouples in series; an 8 by 8 grid of thermopiles makes up the Grid-EYE. However, the paper also explores the use of the pyroelectric sensor, which generates a voltage signal that weakens under a constant temperature reading and requires refreshing the signal at a certain rate (1490). Therefore, if the standard Grid-EYE sensor does not function well enough to detect nearby persons, a custom Grid-EYE can be made out of pyroelectric sensors. (HS)

For the project, a motor is necessary to be able to rotate the module to cover a larger area. There are three types of stepper motors, and the one explored for this project is the hybrid synchronous motor. This motor operates using a voltage pulse train and can rotate in the range of 0.9 to 5 degrees per pulse (Fraser). This provides smooth scanning of the area of interest with a controllable angular speed. The hybrid stepper motor is more expensive than the other two types of stepper motors (variable reluctance and permanent magnet) due to a more sophisticated design. The permanent magnet stepper motor has a much larger step angle which is not what is appropriate for the project. The variable reluctance motor provides a small or medium step size but is faster than what is needed (Fraser). However, after a small design change where the object is rotated rather than the lights, it is clear that the exact angle of the object at any one time is unimportant; instead, it is the approximate speed of the rotation which allows for a guaranteed disinfection of every surface. Therefore, DC motors should also be considered as they are better for speed control; a brushed motor would likely be chosen over a brushless DC motor for cost and control simplicity. (HS)

Another design requirement for the system is a user interface. This user interface would have two parts, the connection on the physical device and an Android application on the user's device. There are several options for the physical connection such as Bluetooth Low Energy, LoRaWAN, USB, or the Wi-Fi Direct protocol. According to the IEEE Network, "the Wi-Fi Direct protocol enables two devices to establish a D2D (device-to-device) connection without the help of APs (access points)" (Shen 1). Therefore, this research suggests that the Wi-Fi Direct protocol will be the most useful for the project because there is one physical device and one user device. This allows the

physical device to create its own self-contained Wi-Fi network that would only connect with the Android application on the user's device. This also allows the device to be installed in a location that may not have a reliable network connection. However, there are some security risks with using the Wi-Fi Direct protocol including eavesdropping, impersonation, message modification, man-in-the-middle attack, and denial of service attack (Shen 1-2). These security issues would not be a major concern for this project, as no personally identifiable information will be transmitted between the devices. An effective alternative to the Wi-Fi direct protocol would be the Bluetooth Low Energy (BLE). According to the Bluetooth website, BLE is a Bluetooth signal designed for low power operation and larger flexibility, such as high accuracy indoor communication, compared to the Bluetooth Classic. BLE is used in a lot of IoT devices to manage the network connections between the devices and the users. Since the device is going to use a one-to-one connection between the user and the device, BLE would allow for an adaptable and secure connection for the user. (LR)

The second part of the user interface is the Android application on the user's device. According to Sikder, in 2019 "Android OS is the dominating smartphone OS with a 75 percent market share" (Sikder 4). Therefore, the application for the user interface will be created for Android OS rather than Apple's iOS or another less-common operating system. The project would use the Android Studio Integrated Development Environment (IDE) for the development of the application with Java as the primary programming language. The Android Studio IDE contains libraries for both the Wi-Fi Direct protocol and Bluetooth Low Energy which can be used during the development process. This Android application should include the ability to receive notifications from

the physical device such as low battery alerts and system fault alerts. The application should also store usage statistics about the UV light that was recorded from the physical device. Finally, the application should have the ability to start an UV light cycles (with the constraint that a detected person nearby overrides a user input to start a light cycle at that time) or stop the UV light cycle before the cycle is completed. (LR)

2.2. Electrical Circuits

The electrical circuit components which are necessary for this project are a UV-C light emitter and a rotation device for the sanitizing chamber platform. These components are explained in more detail and a type of electrical component chosen in each section below.

2.2.1. Electrical Subsystem

Both electrical components to be used in this project will fall under the electrical subsystem.

2.2.1.1. UV-C emitter

After a sufficient amount of research, it was decided to use a UV-C LED strip, the clean-UV UV-C LED Flex Strip. This also allows quick switching between the UV-C LEDs (for the final design) and UV-A LEDs (also known as “black light”, these are safe for human skin and will be used for the prototyping phase.

To simulate the UV-C LED circuit, a model must be designed first. Using LTSpice, a UV-C LED model is designed and then used to build the LED matrix circuit. In order to design an LED in LTSpice, the model D() function is taken advantage of by sweeping LED parameters and observing various I-V characteristic graphs using a DC

sweep. After observing the LED with the correct I-V characteristics graph, the LED is used to build the full matrix circuit. These circuit models are pictured in Figure 1 and **Error! Reference source not found.**.

Figure 1: Circuit setup for simulating I-V curves for different LED models

(HS)

2.2.2. Sanitizing Platform Rotation Device

In the background research, several types of motors and servos which could rotate a platform were explored. It was concluded that the system requires a motor that will rotate the object of interest at least once to get sufficient exposure to UV-C on all sides. The motor torque should also be high enough to withstand the weight of the items. Furthermore, no motor control is needed, only binary control will take place. The type of motor that is going to be used is a DC motor with relatively low speed (maximum 22 rpm) and high enough torque.

Figure 2: The DC motor that is going to be used (HG37-200-AB-00)

(HS)

Additionally, a platform was needed to sit upon the top of the motor where the object could rest while being sanitized. This platform needed to be large enough to fit the object to be sanitized while also providing a way for the UV-C light to pass through. The platform was made using a combination of stainless steel and chicken wire with a setscrew to keep the platform attached to the motor. (LR)

2.3. Electronics

The electronic components in the project include the lifeform sensor, the object sensor, the unlocking mechanism, and a transmitter for the user interface. The microcontroller shall be covered in the Embedded Systems section, 2.7.

2.3.1. Controls Subsystem

The controls subsystem requires input from two sensors, a lifeform sensor and an object sensor, to determine if the sanitation cycle is ready to proceed. It also controls a door servo motor which will be used to unlock the sanitizing chamber door.

2.3.1.1. Lifeform Sensor

To sense the presence of a person or animal within the confined area of the sanitation chamber, several types of sensors were explored during background research. The grid-eye infrared sensor stood out as having the ability to sense a grid of temperatures, and therefore provide enough information to indicate the approximate shape of an object of a given temperature. Therefore, this sensor will be chosen to isolate objects close to body temperature.

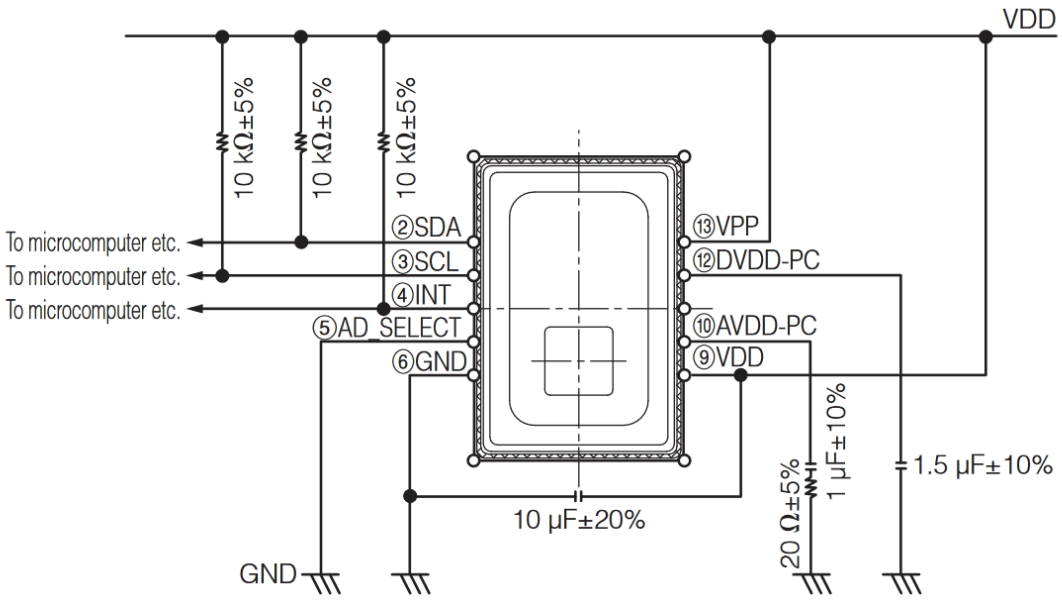


Figure 3: The Grid-Eye sensor schematic for the AMG8833, a potential Grid-Eye component, from the AMG88 datasheet ("Infrared Array Sensor", 4).

2.3.1.2. Secondary Door State Sensor

As the device should never run when the door is open and this is a key safety feature, the ultrasonic sensor will be used as a secondary door state mechanism to ensure that the gate sensor described in Section 2.4.1.3 is functioning properly. If these two sensors ever disagree, a fault will be logged to the phone application.

This ultrasonic sensor can detect the distance to the nearest object. If pointed at the door, it is easy to detect if the door is open by determining if the distance detected is greater than the known distance to the door. Otherwise, the door is likely closed. As this is a secondary sensor, complete accuracy is not required; this is primarily helpful upon startup to ensure that the gate sensor is functioning properly.

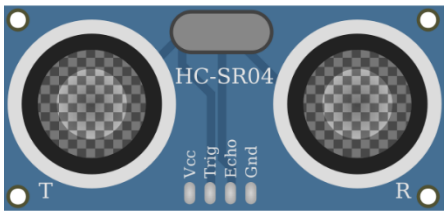


Figure 4: One potential ultrasonic sensor, the HC-SR04 (Ultrasonic Distance Measurement Model)".

2.3.1.3. Unlocking Device

The door of the module, when closed by the user, should automatically lock. It will be unlocked by a command from the microcontroller. The mechanics of such an unlocking device shall be discussed in section 2.6.2.1. This device does not need to be particularly strong, but it should have some degree of precision. Initially, a position-controlled servo motor had been identified as a good choice for this component due to its relative simplicity. However, due to the bulk of the motor and rotational motion, a solenoid has been identified to be a simpler mechanical option to lock the door. This solenoid will be attached to the bottom of the door frame. It operates on 5V; when powered on it extends outside of the frame into the inside of the door to lock it.

Figure 5: Image of Solenoid to be used in the lock

More information on how to control the solenoid shall be examined in section 2.6.2.1.

2.3.2. User Interface Subsystem

The user interface subsystem requires a Wi-Fi/Bluetooth transmission device in order to communicate between the user interface and the other subsystems. The user interface also requires the user to use an android device with an operating system of at least Android 10.

2.3.2.1. Wi-Fi/Bluetooth Transmitter

The user interface for the project will not be physically connected to the microcontroller so a wireless method of communication is necessary to send information between the two subsystems. Both Wi-Fi and Bluetooth have been identified as potential transmitting schemes. Bluetooth will be considered as the primary wireless transmission scheme due to multiple factors including implementation complexity and required range, and Wi-Fi will be considered as the alternate wireless transmission scheme. If issues arise

with the implementation of Bluetooth for the user interface, Wi-Fi will replace Bluetooth as the primary wireless transmission scheme. The wireless transmitter chosen for this project is an RN4781 Click board capable of using Bluetooth Low Energy (LE) as its transmission scheme. (LR)

Figure 6: Image of the RN4871 Click board used for the Bluetooth Low Energy transmitter.

2.3.2.2. Android Device

The user interface will be an application developed to run on android operating systems of Android 10 or newer. In order to access the user interface, the user will need to have access to an android device with this version of the android operating system. This decision is based off access to a testing device accessible during the development process of the user interface application.

2.4. Signal Processing

All three subsystems utilize signal processing; the user interface must reconstruct the user input command using the wireless transmitter, the controls subsystem must read the status of the sanitizing chamber via several sensors, and the electrical subsystem must process motor current to determine if the motor is stuck or not.

2.4.1. Controls Subsystem

There are three major signal processing components in the controls subsystem. First, the analog distance to the nearest object needs to be discretized to determine whether or not an object is present in the sanitizing chamber; the ultrasonic sensor with timer-based signal processing will be used to read the approximate distance. Second, to read the temperature throughout the sanitizing chamber will be achieved using the Grid-Eye sensor to read and discretize an array of discrete temperatures. These temperatures will then be transferred unaltered to the microcontroller. Third, the status of the door needs to be read to determine state. (NB)

2.4.1.1. Ultrasonic Signal Processing

The Ultrasonic sensor shall provide pulses, of which the length of time between is proportional to the distance sensed. That is, the microcontroller will need to poll the ultrasonic sensor line at a high enough rate to avoid missing the pulse. The distance to the sanitizing platform and therefore the neutral (no object) time between pulses can be calculated; a significant drop in the time between pulses indicate a detection.

For example, the HC-SR04, which is determined to be a reasonable ultrasonic sensor to start with due to its low cost and ample documentation and examples, will send out a pulse of the same duration as the returned sonic burst. That is, the pulse duration, $\underline{T_H}$, shall be the twice the distance detected, d, divided by the speed of sound, v_s.

$$T_H = \frac{2 \cdot d}{v_s}, \tag{1}$$

$$d = \frac{T_H \, v_s}{2}, \tag{2}$$

$$d \approx \frac{T_H \, 340 \; m/s}{2}. \tag{3}$$

(NB)

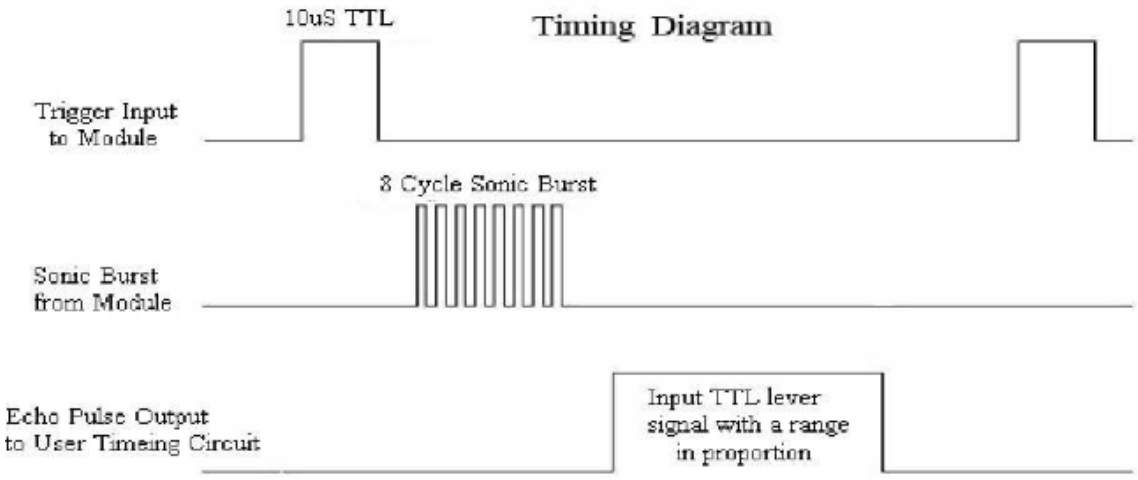


Figure 7: HC-RR04 Datasheet Signal Diagram

2.4.1.2. Grid-Eye Signal Processing

The Grid-eye shall provide an 8x8 grid of temperatures. A particular Grid-eye sensor board, AMG883, has been identified as a good starter board due to the fact that it has a provided breakout board which makes it easier to prototype with and good documentation.

To use I2C, this board requires a clock signal less than 400 kHz. Most standard I2C devices use 100 kHz, which is what will be planned to be used. This sensor will have an I2C address of either 0b1101000 (0x68) or 0b1101001 (0x69) depending on the slave-select pin. The module comes with the pin pulled up by default for the address of 0x69. The pixel registers are 0x80 through 0xFF, with each pixel having 2 registers of data. That is, to read all the grid values of the temperature registers, all 128 registers should be queried.

To query in I2C, first send the device address (0x69) and then the register address to read from and listen for the result. There should be 10 kΩ resistors pulling the I2C lines high when not in use.

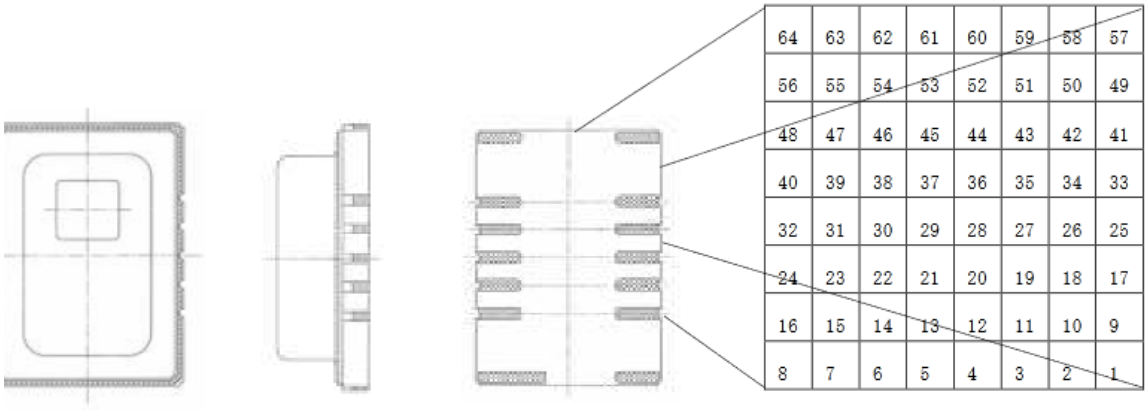


Figure 8: AMG88 Pixel Map ("Specifications for Infrared Array Sensor", 5).

Each of these temperatures is stored in [12, -2] fixed point notation; that is, the temperature (in decimal) T_{10} can be calculated using the H and L registers as in (4).

$$T_{10} = H[3](-2^9) + \sum_{i=0}^{2} H[i]i^{i+6} + \sum_{i=0}^{7} L[i]2^{i-2} \quad (4)$$

(NB)

2.4.1.3. Door Status Signal Processing

The door shall contain contacts which, when closed, bring the value of the a GPIO pin on the microcontroller to HI. This is referred to as the "gate sensor". When the door is open, the pulldown resistor shall bring the pin to a LOW voltage. Therefore, a closed door should correspond with a HI voltage on this pin.

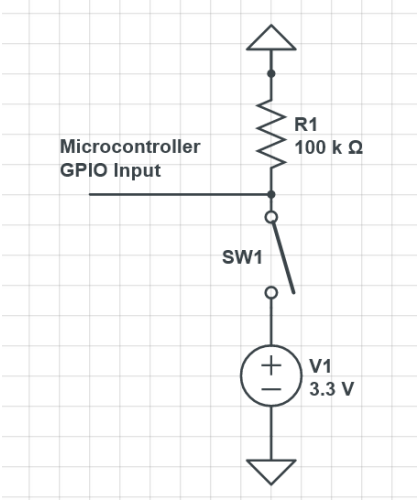


Figure 9: Door Circuit Schematic - Normally Open Switch

This output also serves as a secondary input to the LED power relay so that even if the PIC24 board crashes or fails, the power to the LEDs will be cut off when the door opens.

(NB)

2.4.2. User Interface Subsystem

The user interface subsystem has a more limited approach to signal processing compared to the controls subsystem. The input and output signals will be coming through the Bluetooth LE transmitter to be read by either the android device or the microcontroller. (LR)

2.4.2.1. Wi-Fi/Bluetooth Transmission Signal Processing

The RN4871 Click Bluetooth LE transmitter chip will be acting as the signal processor for the user interface subsystem. The chip will be placed directly into the microcontroller in a predefined position. Then the RN4871 Click board will use the UART protocol to both transmit and receive signals between the android device and the microcontroller. The chip will receive Bluetooth LE transmissions from the android device over low-power radio waves on the frequency band of 2400 to 2483.5 MHz. The

chip will also send Bluetooth transmissions back to the android device after receiving communication from the microcontroller. (LR)

2.4.3. Electrical Subsystem

Other than voltage-level switching using relays, there is no signal processing in the electrical subsystem

(HS)

2.5. Communications

The primary communication needs in this project will be the wireless Bluetooth communication between the transmitter and the android application. However, there is also a lot of digital communication needs between sensors and the PIC24FJ128GA010.

2.5.1. Controls Subsystem

The controls subsystem consists of all digital communication between the sensors and the PIC24 microcontroller. All sensors will either be simple logic/timing based or use I2C.

2.5.1.1. I2C Sensor communication.

Signals from the grid-eye sensor will be received by I2C communication – that is, the signals shall be received using 2 pins; SDA and SCL. SDA is the data line while SCL is the signal line. It supports various communication speeds, with 100kHz being one of the more common speeds. The protocol requires pull-up resistors such that the “active” state is low.

Sparkfun has an in-depth tutorial on the use of I2C (“I2C”); it includes figure Figure 10 which is included below to explain how frames are sent in I2C. Specifically, the line is pulled low to indicate that a frame will be sent, then the address of the device

to receive the frame is sent, and finally the data is sent over the SDA wire.

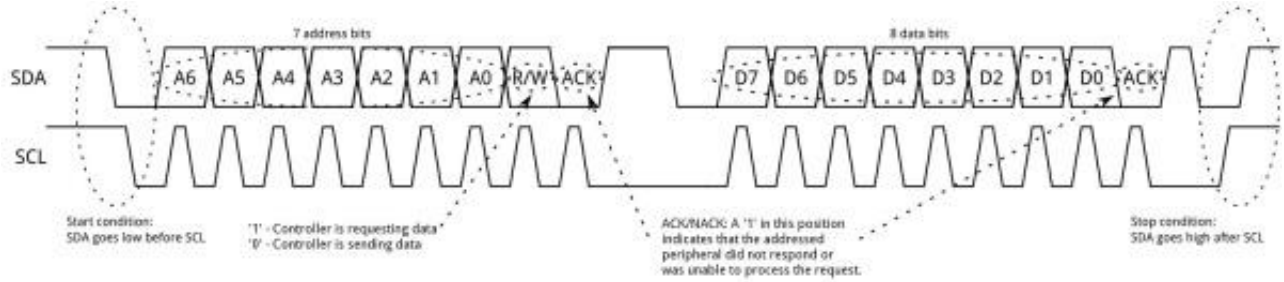


Figure 10: Sparkfun example of I2C protocol frames, ("I2C", 5).

The data should then be read from the sensors; in the case of a grid-eye sensor, for example, this is an 8x8 array of IR data (temperature), and an ultrasonic sensor will provide a distance (possibly also in a grid depending on the sensor chosen).

(NB)

2.5.1.2. Other Sensor communication.

In the case of the gate (door) sensor and estop, the input is entirely binary (open or closed, active or inactive), so a single GPIO input pin on the PIC24 will be appropriate. The ultrasonic sensor will be slightly more complicated, utilizing two pins and timing analysis as described in section 2.3.1.2.

(NB)

2.5.2. User Interface Subsystem

The user interface subsystem will be responsible for the communication between the android device and the microcontroller. To communicate between the two devices, a combination of UART and Bluetooth LE will be used to send and receive data. (LR)

2.5.2.1. Wireless Communication

The user interface will use the UART protocol for part of the wireless communication, namely the communication between the microcontroller and the RN4871 Click. The setup for I2C between the two devices will be similar to the setup described in

section 2.5.1.1 above, however the ESP chip will be considered the slave in the relationship. In addition to I2C, the ESP-WROOM-32 chip will use Bluetooth to communicate between the chip and the android device. The android device will use Bluetooth radio waves to communicate data to the chip which will then use I2C to communicate the data back to the microcontroller. This process will also work in reverse with the microcontroller sending data to the chip via I2C and then the chip sending the data to the android device via Bluetooth. (LR)

2.6. Electromechanics

The electromagnetic needs of this project shall be relatively simple – they consist of a motor system which will rotate the sanitizing platform and the target object, and a lock which will keep the door closed and locked when the sanitizing cycle is active.

2.6.1. Electrical Subsystem

The primary electromechanical subsystem in the Electrical Subsystem is the platform motor, which must rotate the target object.

2.6.1.1. Platform Motor Torque

In order to provide the constant speed needed to sanitize equally, the necessary motor torque needed to accelerate the sanitizing platform is calculated as follows:

Approximating the platform base as a perfect circular disc with radius R and combined mass M, the moment of inertia of the disc may be written as:

$$I = \frac{1}{2}MR^2$$

Given a desirable speed ω_0, and a time interval Δt needed to reach a constant speed, the angular acceleration may be approximated as:

$$\alpha = \frac{\Delta\omega}{\Delta t} = \frac{\omega_o}{\Delta t}$$

The toque may then be calculated from the standard formula:

$$\tau = I\alpha$$

This simplifies to:

$$\tau = \frac{\omega_o MR^2}{2\Delta t}$$

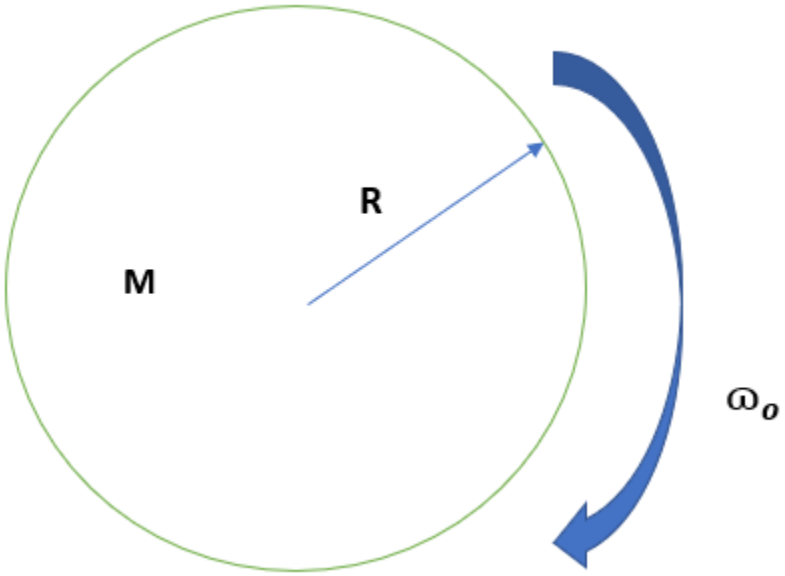


Figure 11: Disc model of the sanitizing Champer base

(HS)

2.5.1.2 DC motor simulation

By using Simulink, the DC motor proposed is simulated. Using the datasheet of the motor, the stall torque and no-load speed given can be used to modify the DC motor model in Simulink to match our motor. For the simulation setup, a linearly increasing torque ranging from zero to the stall torque in one second is applied. The resulted speed curve is then observed. To validate the results, it can be observed that the speed is zero at one second for the maximum load condition

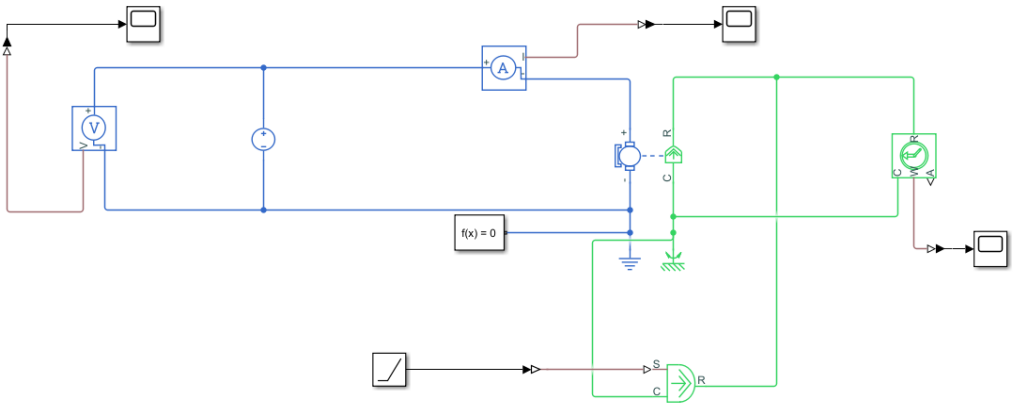


Figure 12: DC motor Simulink model (24 V excitation)

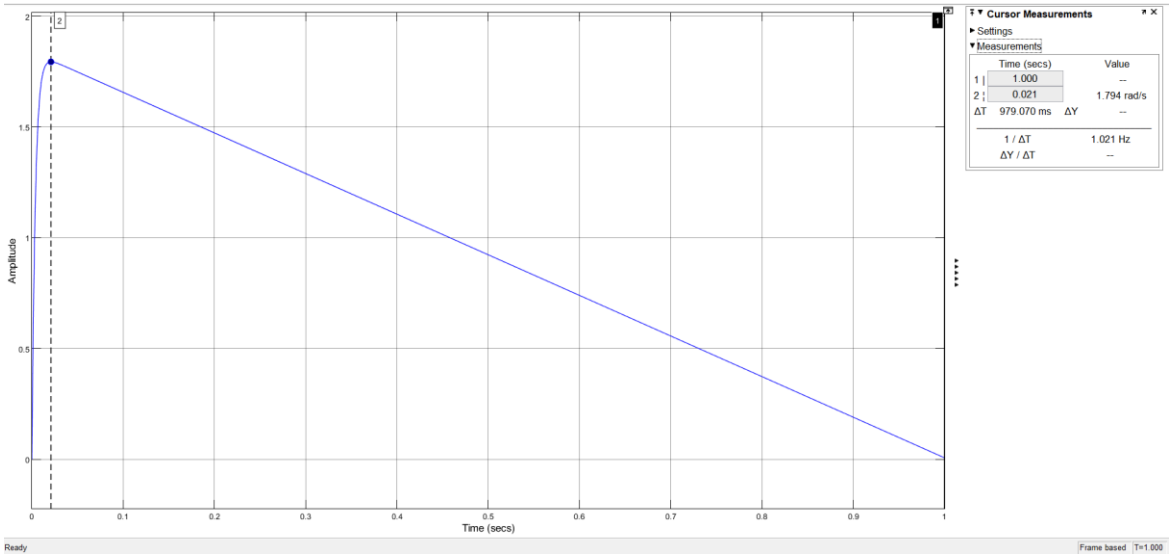


Figure 13: Resultant Speed curve from Simulink simulation

(HS)

2.6.2. Controls Subsystem

The controls subsystem is primarily digital and therefore has few electromechanical components; however, in the effort of evening work distribution, the servo door unlocking mechanism shall fall under this subsystem.

2.6.2.1. Solenoid Lock

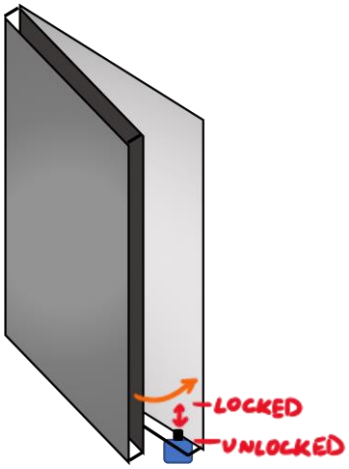


Figure 14: Mechanical Sketch of Lock System

The lock servo shall be used to release a mechanical lock; this lock shall be engaged upon door close. The gate sensor described in Section 2.4.1.3 will allow the microcontroller to detect a closed door, and before a cycle starts, the door will be locked. Commands to this lock solenoid are purely digital, transformed from 3.3V to 5V using a power relay – “HI” from the microcontroller locks the door, and “LO” unlocks the door. In the event of a power failure, the door will be unlocked, allowing any items inside to be removed.

(NB)

2.7. Embedded Systems

A single microcontroller will be used in this project and will be responsible for all control decisions regarding state of the module. The user interface will not be an embedded system because it is designed to run on an Android device, which is closer to a general computer than an embedded system. This microcontroller must be powerful and precise enough to drive multiple I2C connections, a PWM connection, and have timers which give timing precision at the microsecond level. There should also be general

purpose input and output pins for reading and writing less complex digital information, such as the status of the Estop or turning on LEDs.

Other considerations when choosing a microcontroller are the complexity of use, how much troubleshooting hardware is available, and cost. The PIC24FJ128GA010 is offered through the university on the Explorer16 development board, and therefore would be the cheapest option with a significant amount of troubleshooting hardware and documentation. The primary tradeoffs are size and power; since this module shall not be mobile and will remain plugged into a wall outlet, this tradeoff is expected to be very favorable for this specific application.

(NB)

2.8. Controls

The control subsystem will orchestrate a state machine based on user and sensor inputs. The microcontroller It shall ensure that enough time is spent sanitizing the object.

2.8.1. Prevent Human Exposure

Before running the light cycle, the grid-eye sensor should be queried; if several pixels detect a value close to human body temperature, the LEDs should not be activated, and a fault should be reported. The door should also be unlocked to release anything trapped inside.

(NB)

2.8.2. How much time to sanitize

Given that the radiated power of the UV-C LED chosen is 60 mW, an approximation of the power density on a spherical shell going through the center of the box can be calculated. The average value of the intensity can be calculated by dividing the radiated

power by the amount of steradians in a sphere. Using the fact that the power density is related to the intensity through the square of the distance from the source, the power density is obtained. Given the energy density of 10 mJ/cm^2 needed to achieve sanitizing, the time needed is obtained by dividing the energy density by the power density. The time can then be reduced through using multiple LEDs

$$Uav = \frac{Prad}{4\pi} = \frac{60\,mW}{4\pi\,sr} = 4.77\,\frac{mW}{sr}$$

$$\boldsymbol{P}_{av} = \frac{Uav}{R^2} = \frac{4.77\,\frac{mW}{sr}}{30^2 cm^2} = 0.0053\,\frac{mW}{cm^2}$$

$$t = \frac{\boldsymbol{E}}{\boldsymbol{P}_{av}} = \frac{10.6\,\frac{mJ}{cm^2}}{0.0053\,\frac{mW}{cm^2}} = 2000\,s$$

Using 10 LEDs, the time is reduced to 200 seconds, or 3 minutes and 20 seconds. To make up for the power density estimation, 5 minutes results in an overestimate ensuring proper sanitizing.

(HS)

3. Engineering Requirement Specification

The device should follow a set of engineering requirements and specifications to ensure that its design will meet the identified marketing requirements.

Table 2: Engineering Requirements Table

Market Req.	Engineering Requirement Specifications	Justification
1	An object placed inside the sanitizing chamber should be exposed to at least 10mJ/cm^2 of UV-C light on all exposed surfaces within 5 minutes.	Light intensity 10mJ/cm^2 (Heßling et al) required to kill most coronaviruses. 5 minutes is a reasonable amount of time to reach this level with 10 LEDs. (Analysis in section 2.2.1.1).
2	The sanitizing chamber should be capable of holding a target object smaller than 18”x14”x8” and 20 lbs.	Limit object size to the size of a personal item defined by American Airlines with a weight of 20 lbs., which is 10% (“Stuffed Backpacks”) of the average American adult male weight (“FastStats”).
1	The sanitizing chamber should rotate the target object at least $\pi/30$ rad/s.	Need to ensure entire object surface is exposed to LEDs, so aim for 1 full rotation/minute to allow for full coverage of object even when rotation speed is not exact.
5	The device should be able to detect the presence of an object within the sanitizing chamber and differentiate this from the presence of an animal or hands by detecting a temperature of 85-105°F of the contents.	To avoid trapping an animal inside or child inside, the module should detect a range of temperatures within approximately 10°F of human body temperature.
3	The design should include a mobile application which should interface with the device after scanning a code containing the device’s specific MAC address.	Allow the user to control the device wirelessly from their personal device while connecting as easily as possible (do not “search through Bluetooth connections by hand”)
3	The user interface should allow the user to start and stop the sanitizing cycle and unlock the device; it should contain information about the time remaining in the sanitizing cycle and any faults which are active.	The user interface should contain enough information for the user to effectively use and control the device.
3	The system should contain an Estop system which disables the UV-C LEDs and should be automatically activated when the sanitizing chamber door is open.	There should be no light allowed to escape even if the door is pried open.

4	An initialization process should begin as soon as the microcontroller is plugged in and take less than 10s to complete and enter the "ready" or "fault" state.	The user expects the system to be ready in a short time after being plugged in – 10s is a reasonable amount of time for a user to wait before using the device and allocates plenty of time for the power system to start up, and the microcontroller to check all sensor and component statuses.
4	The device should connect to a 120V AC wall outlet and draw at most 5A at any time.	The device should be capable of operating on a standard wall outlet at a reasonable current. 5A is often used in fast-chargers so this is a reasonable maximum voltage and capable of powering the UV-C LEDs.
2, 3	The device should monitor the rotation of the platform to determine if an object has become stuck or is too heavy and register this as a fault.	As the object will be rotating, there is some chance that it will fall in such a way that the motor is jammed; this case should be detected to prevent the object or module from being broken.

4. Engineering Standards Specifications

Based on the engineering analysis, several standards have been identified which the project should follow whenever possible. These are detailed in *Table 3.*

Table 3: Engineering Specifications Table

	Standard	**Use**
Safety	NASI/ISEA Z87.1 UC (glasses certification)	Safety Glasses with this certification will be worn when operating any UV-C LED.
Communication	I2C	Sensor communication with microcontroller.
	Bluetooth or IEEE 802.11	Wireless communication from module to phone application.
Data Formats	JSON	Potential standard format for data between module and phone application.
Design Methods	Finite state machines	Software will employ state machines to consolidate data and which actions need to be taken at each time.
Programming Languages	C	Code running on microcontroller.
	Java	Phone application source code.
Connector Standards	Through-hole	Solder through-hole components to the developed PCB.

5. Accepted Technical Design

5.1. Level 0 Design

Figure 15 and *Table 2* discuss the "level 0" or basic inputs and output of the UV-C Smart Light System. This system should meet the marketing requirements discussed in section 1.3.

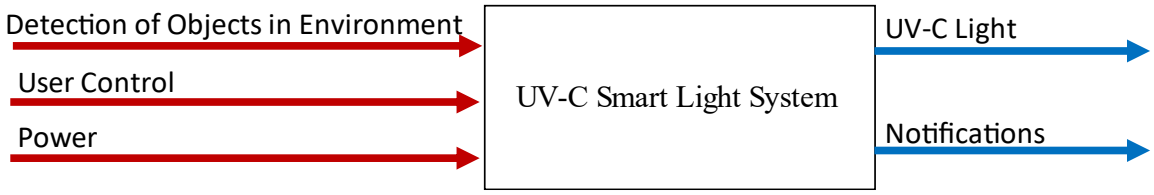


Figure 15: Level 0 Block Diagram

Table 4: Level 0 Functional Requirements Table

Module	**UV-C Smart Light System**
Designer	Team 12: Nicole, Luke and Haitham
Inputs	Motion and IR sensor: Detection of nearby motion and IR sources Power: 120 V AC standard household outlet. User interface control: Start/Stop commands.
Outputs	UV-C Light: ~222 nm with an intensity of at least 10 mJ/cm^2, measured 1 foot away from the module. Status: Visible Light indicating module status. Notifications: Usage statistics and Status
Description	Turn the UV-C light on or off based on user interface commands. The detection of a person or animal in the sanitizing chamber should overrule a command to turn on, resulting in the light remaining off.

(NB, LR, HS)

5.1.1. Hardware Theory of Operation

The electronic subsystem shall be focused on providing power to all components and ensuring that analog outputs such as light intensity or motor torque are sufficient to

meet the engineering requirements. The control subsystem will contain the digital signals which control the operation of the motor and LEDs, as well as digital communication with all sensors. The user interface subsystem contains the communication with the Bluetooth transmitter and the android user interface.

5.2. Level 1 Hardware Design

Figure 16 shows the level 1 hardware system block diagram; it shows the primary submodules which will exist within the project, such that each has a unique set of high-level inputs and outputs. This hardware will be further broken down in section 5.3.

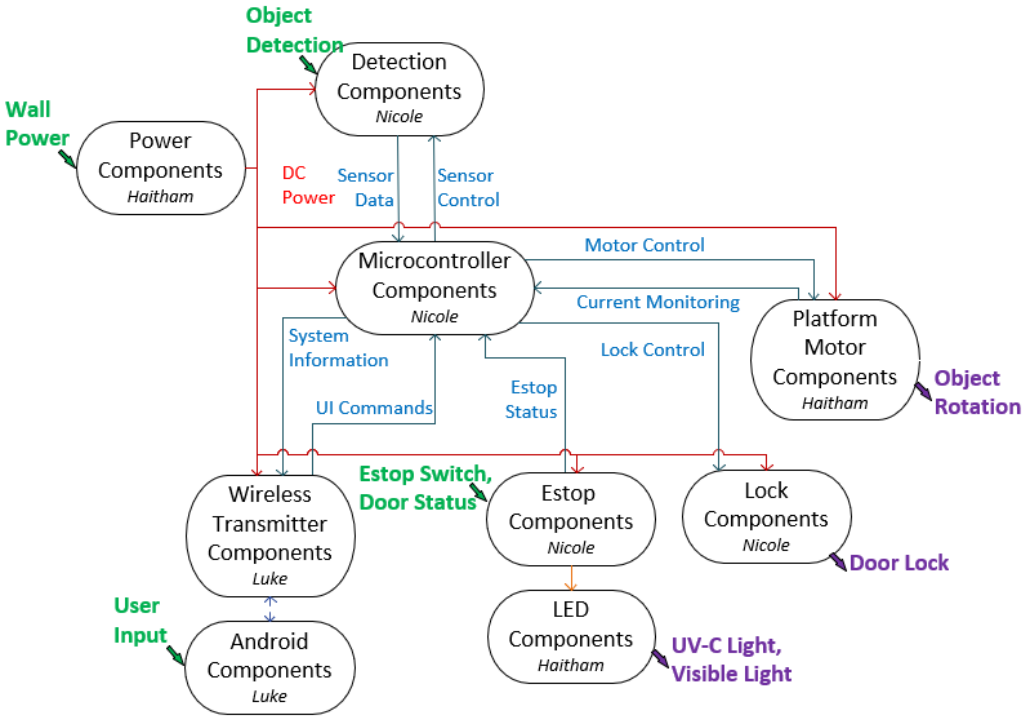


Figure 16: Hardware Level 1 Block Diagram

This block diagram shows the 9 unique subsystems within our project and the designer for that subsystem, as well as the inputs to and outputs from the module to the environment

and user. These inputs and outputs, as well as more the more detailed subsystem-to-subsystem interface, are explained in *Table* 5.

Table 5: Hardware Level 1 Functional Requirements Table

Module	**Power Components**
Designer	Haitham Saleh
Inputs	Power: 120 V AC standard household outlet.
Outputs	Multiple Regulated Voltages (per other system)
Description	Regulates wall power to provide consistent, regulated DC voltage to each subsystem.
Module	**Microcontroller Components**
Designer	Nicole Baldy
Inputs	Sensor Data: Human and Object Detection information. Estop Status: Current state of the Estop system. UI Commands: Start/Stop commands from the UI. Power: DC voltage to power microcontroller.
Outputs	System Information: System faults, sanitizing/ready status to the UI. Lock Control: Lock and unlock the module door. Motor Control: Turn off and on sanitizing platform motor. LED Control: Turn on and off the UV-C and status LEDs. Sensor Control: Turn off and on the sensors.
Description	The microcontroller contains the clock and orchestrates all the other components; timings and state.
Module	**Estop Components**
Designer	Nicole Baldy
Inputs	DC Voltage: Voltage for UV-C LEDs is passed through the Estop system. Door Status: Estop is triggered if door is open. Estop Switch: An Estop button available to the user.
Outputs	DC Voltage: Pass through or cut off voltage to the UV-C LEDs. Estop Status: Communicate status back to the microcontroller.
Description	The Estop system is entirely hardware-controlled, and when enabled cuts off all voltage to the UV-C LEDs. The microcontroller should be able to determine its state.
Module	**LED Components**
Designer	Haitham Saleh
Inputs	DC Voltage: Voltage passed through Estop for UV-C LEDs.

	LED Control: Microcontroller commands to set LED status.
Outputs	UV-C Light: Sanitizing light inside the chamber. Visible Light: Status indicators outside of the module.
Description	The LED system controls all LEDs within the module, including the UV-C sanitizing LEDs.
Module	**Wireless Transmitter Components**
Designer	Luke Rogers
Inputs	DC Voltage: Voltage for powering the Wi-Fi User Input Data: Wireless data from the Android UI application. Usage Data: Data from the microcontroller to send to the UI.
Outputs	UI Commands: Pass through the UI commands to the Microcontroller.
Description	The wireless system sends and receives data to and from the user interface application.
Module	**Android Components**
Designer	Luke Rogers
Inputs	User Input: Direct input through the interface from the user.
Outputs	UI Commands: Pass through the UI commands to the Wireless Transmitter.
Description	The Android system provides a user interface to the target user and accepts commands.
Module	**Lock Components**
Designer	Nicole Baldy
Inputs	DC Voltage: Voltage for powering the lock. Lock Control: Commands from the microcontroller to change lock status.
Outputs	Lock State: Door lock is activated or deactivated.
Description	The lock system locks and unlocks the module door at the command of the microcontroller.
Module	**Platform Motor Components**
Designer	Haitham Saleh
Inputs	DC Voltage: Voltage for powering the motor. Motor Control: Commands from the microcontroller to activate and deactivate the motor.
Outputs	Object Rotation: Motor rotates object in sanitizing chamber when activated. Current Monitoring: Report the status of the motor's current consumption.
Description	The platform motor system rotates the target object when the sanitizing cycle is active.

Module	Detection Components
Designer	Nicole Baldy
Inputs	Object status: Whether an object is present within the sanitizing chamber. Lifeform status: Whether a lifeform is present within the sanitizing chamber. DC Voltage: Power for the detection sensors.
Outputs	Sensor Data.
Description	The sensor system detects objects and lifeforms inside the sanitizing chamber.

(NB, input from HS)

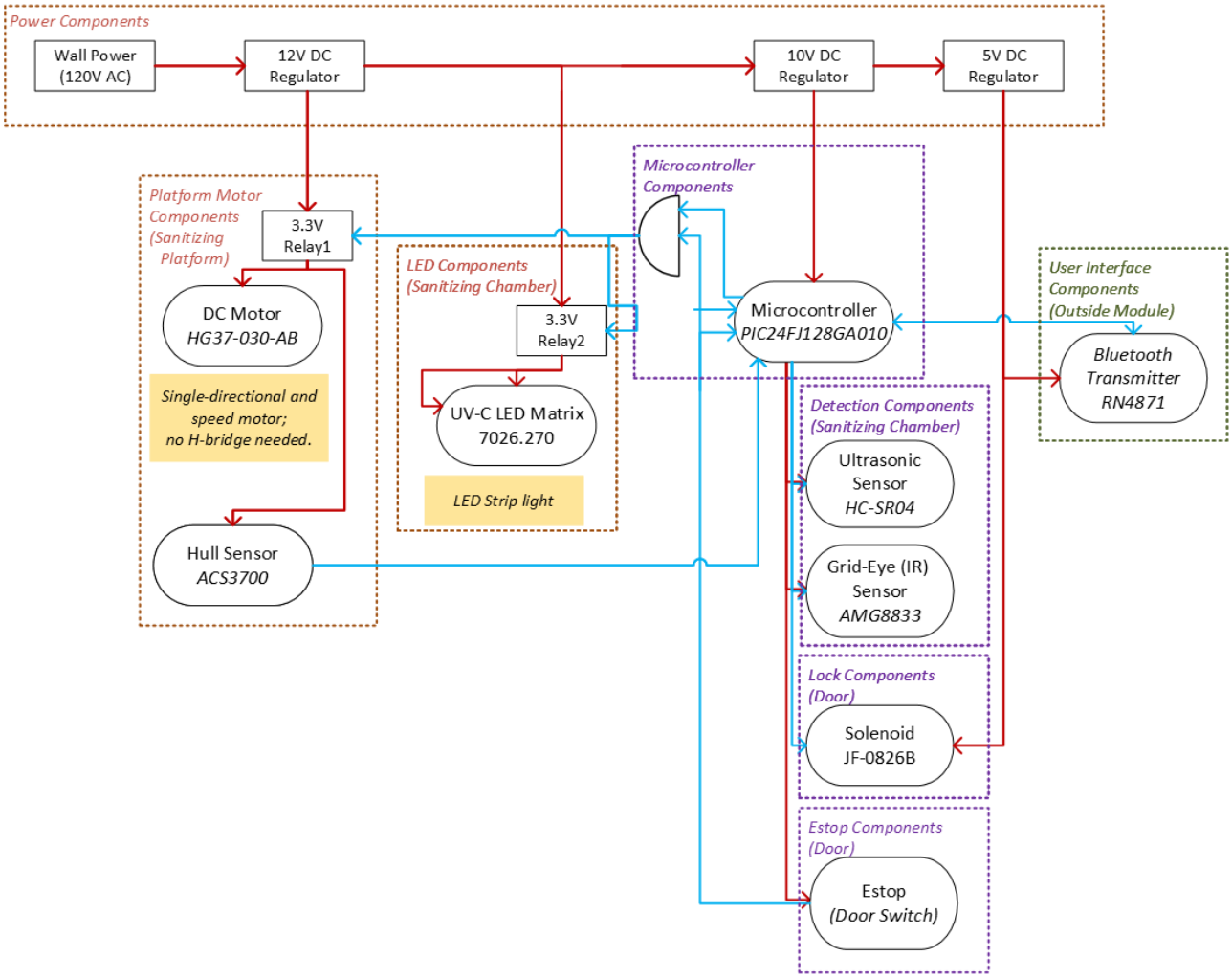
5.2.1. Hardware Theory of Operation

As noted in section 5.1.1, there are three important subsystems to be integrated within this project. The electronics subsystem will contain the power, motor, and LED components and the power and analog connections to the other subsystems; it will also contain the current sensor ability as part of the motor module. The control subsystem will contain the detection, lock, microcontroller, and Estop components and their digital connections to the other subsystems. The UI subsystem will contain the communication of data to and from the android UI application and the Bluetooth transmitter.

5.3. Level 2 Hardware Design

The system electrical components are then broken down further to include communication protocols and specific off-the shelf parts in *Figure 17*, the level 2 hardware block diagram of the system. (NB)

Figure 17: Hardware Level 2 Block Diagram



In this block diagram, I2C is the communication protocol between the microcontroller and the sensors and wireless transmitter. The only exception is the ultrasonic sensor, which uses a timing-based communication, such that a pulse duration is

proportional to distance. The specific inputs and outputs, and any associated protocols are explained in *Table 6.*

Table 6: Hardware Level 2 Functional Requirements Table

Part	**AC-DC Adapter**
Designer	Haitham Saleh
Inputs	AC 120 V wall power.
Outputs	24V D.
Description	An off-the-shelf wall wart shall be used to adapt AC wall power to the maximum voltage required by the design.
Part	**[3x] DC Voltage Regulator**
Designer	Haitham Saleh.
Inputs	DC Voltage (24V, 14V, or 9V DC)
Outputs	DC Voltage (14V, 9V, or 5V DC)
Description	3 voltage regulators, each of which will contain a Commercial Off the Shelf component as well as capacitors and resistors to properly regulate the voltage. See section 5.4 for more detailed information.
Part	**Servo**
Designer	Nicole Baldy
Inputs	PWM control (from microcontroller), 5V DC Power.
Outputs	Servo angle.
Description	A small servo will be used as an unlocking device. The lock shall engage mechanically upon the door closing; the servo will release the locking mechanism to unlock the door such that it springs open.
Part	**Estop Gate Sensor**
Designer	Nicole Baldy, Haitham Saleh
Inputs	3.3V DC Power (from microcontroller), Door state (as switch).
Outputs	Digital Signal (0V or 3.3V) indicating status (Microcontroller GPIO and digital logic).
Description	This device will be designed to detect the status of the door – this will cut power electrically to the LEDs via digital logic when the door is open as well as report status to the microcontroller.
Part	**Estop Button**

Designer	Nicole Baldy, Haitham Saleh
Inputs	3.3V DC Power (From microcontroller), Estop button state (as switch)
Outputs	Digital Signal (0V or 3.3V) indicating status (Microcontroller GPIO and digital logic).
Description	This button will be available for any user to press if the device would malfunction – this will cut power electrically to the LEDs via digital logic as well as report status to the microcontroller.
Part	**Microcontroller**
Designer	Nicole Baldy
Inputs	DC Power (9V), GPIO, (Estop Gate Sensor, Estop Button), I2C (Sensors, Transmitter).
Outputs	PWM Commands (Servo), Digital Signal (0v or 3.3V) to control door and motor.
Description	The microcontroller will have digital connections to all sensors as well as the motor and LED relays.
Part	**Grid-Eye Sensor**
Designer	Nicole Baldy
Inputs	3.3V DC Power (From Microcontroller), I2C commands (From Microcontroller).
Outputs	I2C detection data (8x8 temperature array) as I2C register contents.
Description	An 8x8 Pixel IR sensor that stores temperature, and will be read to determine lifeform presence.
Part	**Ultrasonic Sensor**
Designer	Nicole Baldy
Inputs	3.3V DC Power (From Microcontroller), Digital Pin input (From Microcontroller).
Outputs	Digital pin output (timed pulse).
Description	An ultrasonic distance sensor which will be used to determine whether there is an object inside of the sanitizing chamber or not. See section 5.6 for more information on how this determination is made.
Part	**UV-C LED Matrix**
Designer	Haitham Saleh
Inputs	14V DC Power.
Outputs	UV-C light.

Description	2 UV-C LEDs connected in series, with 5 of these groups in parallel. These are the source for sanitizing
Part	**DC Motor**
Designer	Haitham Saleh
Inputs	DC Power,
Outputs	Platform Motor Movement.
Description	A simple DC motor that will provide at least 5 revolutions per operation to sanitize the item of interest equally on all sides
Part	**Wireless Transmitter**
Designer	Luke Rogers
Inputs	5V DC Power, I2C Commands (from microcontroller).
Outputs	I2C Commands (from UI).
Description	The Wireless transmitter encodes I2C data from the microcontroller into wireless (WiFi or Bluetooth) signal, and wireless signal from the user interface application into I2C data.

(NB, HS)

5.4. Level 3 Electrical Subsystem

The three major elements of the electrical subsystem are the LED matrix circuit, the motor, and the high-current voltage regulator. The following schematic shows the appropriate connections with the various electrical components. The 24 V adapter output is connected directly across the DC motor. The output of the voltage regular is adjustable via a resistive network. The LED matrix circuit necessitates a 14 V input, which means the voltage division circuit consists of the following resistors:

$$Vout = Vnominal * \left(1 + \frac{R2}{R1}\right) + error\ term$$

$$14 = 1.25 * \left(1 + \frac{R2}{R1}\right) + zero$$

$$R2 = 10\ k\Omega\ \ R1 = 1k\Omega$$

This is assuming the error term is zero ($IADJ * R2 = 0.$) Also, the values of the resistors produce an output of 13.75 V instead of 14 V. Exact values could've been easily achieved using a potentiometer, but for the sake of simplicity, fixed standard values are used. Moreover, specific types of capacitors are connected on the input, output, and adjustments pin of the regulator. The reasoning behind the use of the capacitors is the improvement of the transient response and reducing the ripple voltage at the output. The use of the capacitors adds the need to use protective diodes to disallow the capacitors to discharge into the electronics of the regulator and possibly damage it. Note: the motor is not present on the schematic due to the symbol not being available on Mouser for the Eagle schematic.

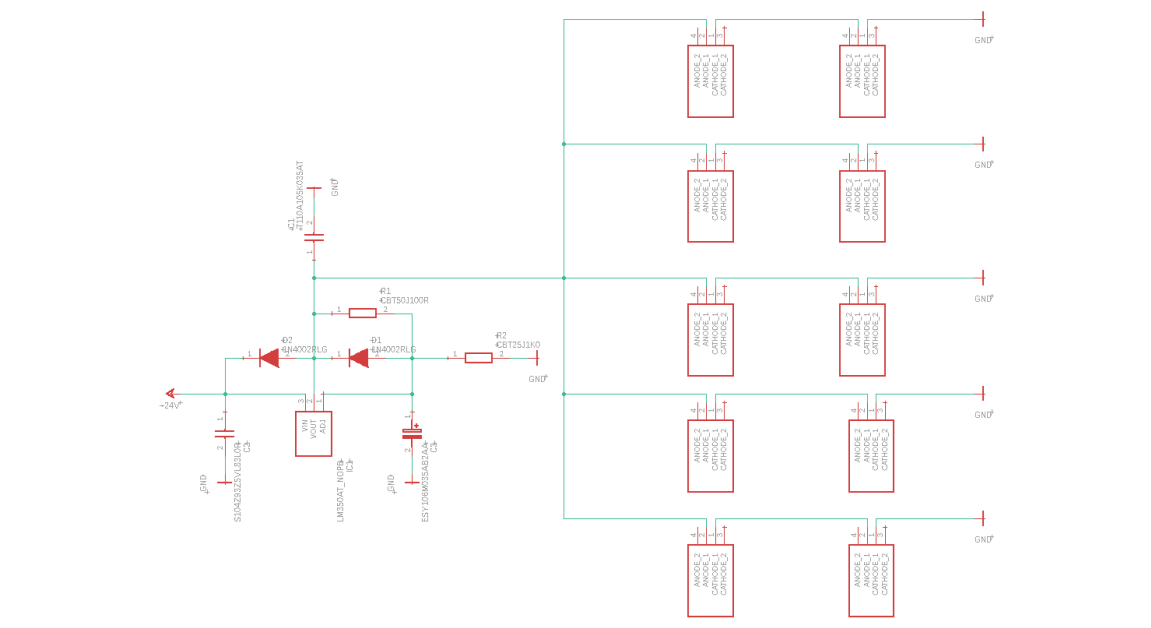


Figure 18: Schematic of the level 3 electrical subsystem

(HS)

5.5. Level 3 Control Subsystem

The control subsystem consists of all sensors (as these are used to determine the state of the system for making control decisions) as well as all control outputs on the digital level. This subsystem will not cover the power requirements or mechanical workings of the module but will focus on the digital connections necessary to enable digital communication between the PIC24FJ128GA010 microcontroller and all components of the module.

A schematic of these components has been designed in Autodesk Eagle to show the connections between components. This schematic is shown in Figure 19.

Of these components, the microcontroller connections to the AMG8833 Grid-Eye sensor, the door input pin, and the relay to LEDs and motor have been tested; see the code in Section 5.8.

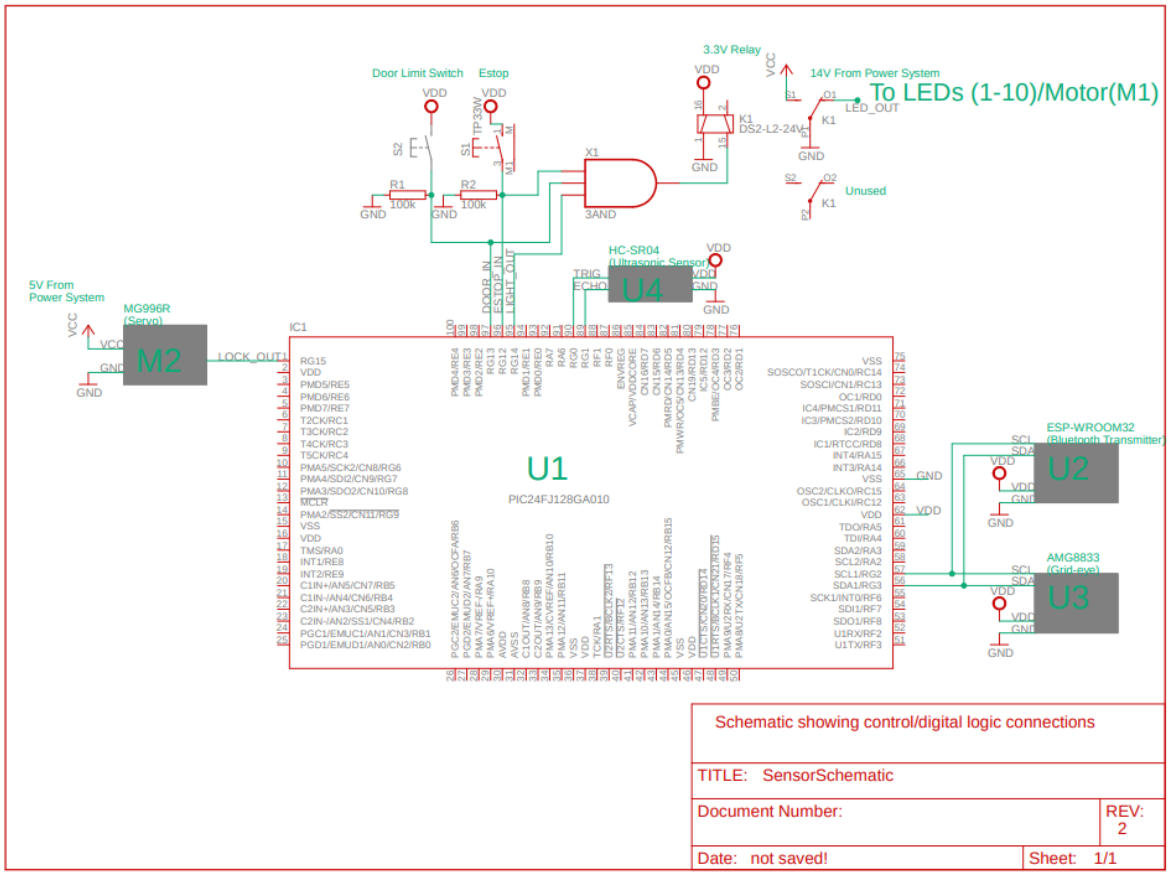


Figure 19: Schematic of control system components

Note that Figure 19 only includes components which fall under the control system, with the exception of the ESP Bluetooth Transmitter chip; this transmitter is part of the user interface subsystem, but the digital I2C communication falls partially under the controls subsystem due to the shared bus with other sensors.

(NB)

5.6. Level 1 Software Design

The level 1 software system is broken down into a single flow-control diagram shown in Figure 20, with the table of functional inputs/outputs in *Table 7*. This diagram contains functions performed by the collaboration between the microcontroller and the

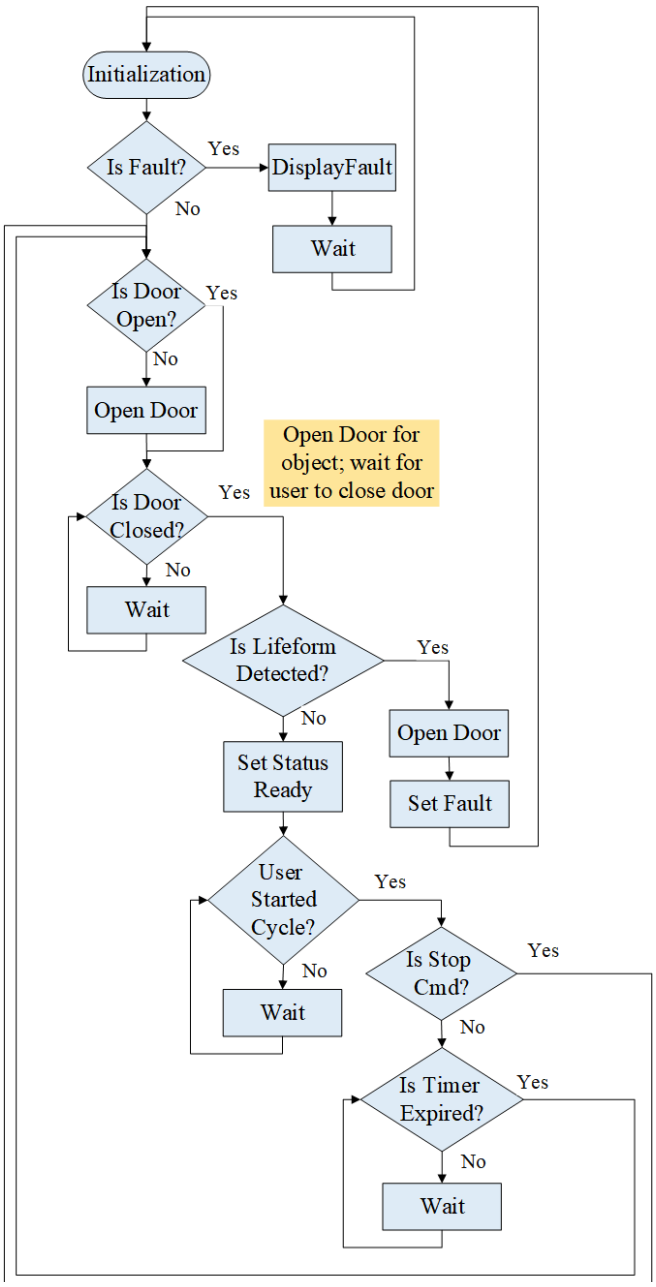


Figure 20: Software Level 1 Diagram

user interface without differentiation; these two software systems will be broken down separately in section 5.6.

Table 7: Software Level 1 Functional Requirements Table

Function	**Initialization()**
Designer	Nicole Baldy
Inputs	Sensor State Estop State Wireless Transmitter State Door State Lock State
Outputs	Fault Codes System State
Description	Queries status of electrical subsystems and sensors; sets any faults and initializes system state.
Function	**DisplayFault()**
Designer	Nicole Baldy
Inputs	Fault Code(s)
Outputs	LED Fault Code User Interface Fault
Description	Given a set of fault codes, DisplayFault parses the fault and sets all appropriate indicators to show a fault state.
Function	**Wait()**
Designer	Nicole Baldy
Inputs	Time to wait
Outputs	Interrupt set.
Description	Pauses code execution until wait time is met.
Function	**GetDoorStatus()**
Designer	Nicole Baldy
Inputs	Gate Sensor State
Outputs	Door Status
Description	GetDoorStatus checks the status of the gate sensor (part of the Estop).
Function	**OpenDoor()**
Designer	Nicole Baldy
Inputs	N/A
Outputs	Motor Command Door Status

Description	OpenDoor sends the commands to the lock servo to unlock the door such that it springs open.
Module	**IsLifeformDetected()**
Designer	Nicole Baldy
Inputs	Sensor Data
Outputs	Detection Status
Description	IsLifeformDetected reads the GridEye sensor and determines whether a lifeform is present.
Function	**QueryUserCommand()**
Designer	Nicole Baldy
Inputs	Wireless transmitter data
Outputs	Command Status
Description	QueryUserCommand checks the wireless transmitter for a pending start command.
Function	**SetStatusReady()**
Designer	Nicole Baldy
Inputs	N/A
Outputs	LED Ready Code User Interface Ready Status
Description	SetStatusReady sets the appropriate indicators to show the ready status.
Function	**SetFaultCode**
Designer	Nicole Baldy
Inputs	FaultCode
Outputs	UV-C LED off command Motor Off Command Sensors Off Command Unlock command *Calls DisplayFault()*
Description	SetFaultCode saves a fault code and (code-dependent) pulls the system into a “safe” state by turning off the UV-C LEDs, Platform Motor, and unlocking the door.

(NB, input from LR)

5.6.1. Level 1 Software Theory of Operation

The Level 1 Software consists of two subsystems: the control and user interface subsystems. The control subsystem can be thought of as a state machine, with sensors and Estop components generating “events” which change the state of the system. The user

interface consists of a Java-based android application and its connection to the microcontroller via a Bluetooth transmitter. In this way, the control subsystem and the user interface subsystem can be thought of each having a public interface which is accessible to the other, as information must be passed in both directions. Specifically, the control subsystem determines what inputs are valid at a given time and relays the current state to the user interface subsystem; the user interface subsystem displays this status, waits for a user to provide input via the application, and relays this input to the control subsystem. (NB)

5.7. Level 2 Control Software

Two different sets of software must be written for this project: the control software, which is written in C and will be running directly on the microcontroller, and the user interface software, which will be running on an Android application that will wirelessly interface with the device.

This section focuses on the controls subsystem. The microcontroller in this project can be visualized as both a flowchart and a state machine; the flow chart is shown in *Figure* 21 and explores how specific components of the software will function together. The state machine in Figure 22 focuses on the higher-level control flow, showing how the microcontroller transitions between statuses. The input and output information is located in *Table 8*.

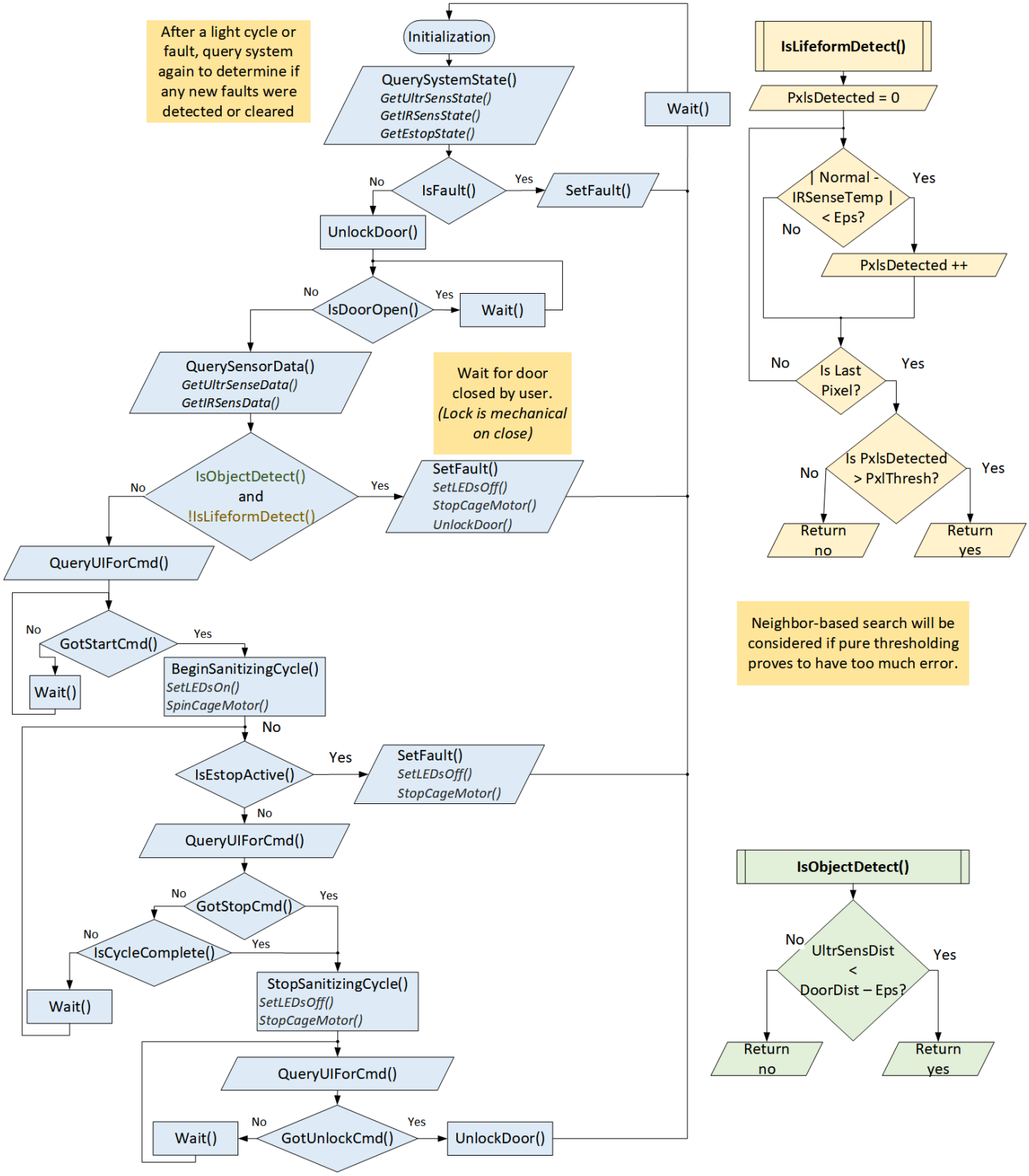


Figure 21: Microcontroller Level 2 Software Flowchart

Table 8: Software (Microcontroller) Level 2 Functional Requirements Table

Function	**Initialization()**
Designer	Nicole Baldy
Inputs	N/A.
Outputs	Reset all statuses.
Description	A short function to reset any outdated status and begin the querying process.
Function	**GetUltrSensState()**
Designer	Nicole Baldy
Inputs	N/A.
Outputs	Ultrasonic Status.
Description	Attempt to read distance from the ultrasonic sensor. If no response or distance does not make sense (too large/small), this is a fault.
Function	**GetIRSensState()**
Designer	Nicole Baldy
Inputs	N/A.
Outputs	IR Status.
Description	Attempt to read pixels from the ultrasonic sensor. If no response or result does not make sense (temperatures outside of reasonable threshold), this is a fault.
Function	**GetEstopState()**
Designer	Nicole Baldy
Inputs	N/A.
Outputs	Gate Sensor State. Estop Button State.
Description	Read both the GPIO status of the Gate Sensor and the Estop Button.
Function	**IsFault()**
Designer	Nicole Baldy
Inputs	Existing Fault Codes, Sensor Data, Estop status.
Outputs	Fault Status (binary).
Description	Determine whether existing fault codes are still active or if any new faults are detected based on sensor input; if so, then return true (fault active).
Function	**Wait()**
Designer	Nicole Baldy
Inputs	Time to wait
Outputs	Interrupt set.

Description	Pauses code execution until wait time is met.
Function	**UnlockDoor()**
Designer	Nicole Baldy
Inputs	N/A.
Outputs	Servo movement.
Description	Commands the servo to unlock the door.
Function	**IsDoorOpen()**
Designer	Nicole Baldy
Inputs	Gate Sensor Data
Outputs	Door Status (Boolean).
Description	Returns true (open) if gate sensor circuit is detected as open.
Function	**SetFault()**
Designer	Nicole Baldy
Inputs	IR Sensor Data.
Outputs	Lifeform Detection (Boolean).
Description	Iterates through the I2C detection grid, determines if the pixel falls within the lifeform range and thresholds the number/cluster of potential detections to determine overall detection status.
Function	**GetUltrSenseData()**
Designer	Nicole Baldy
Inputs	N/A.
Outputs	Ultrasonic Sensor Data (Distance).
Description	Read distance from the Ultrasonic sensor and save this data.
Function	**GetIRSenseData()**
Designer	Nicole Baldy
Inputs	N/A.
Outputs	IR Sensor Data (8x8 temperature grid).
Description	Read temperatures from the IR sensor and save this data.
Function	**IsObjectDetect()**
Designer	Nicole Baldy
Inputs	Ultrasonic Sensor Data.
Outputs	Object Detection (Boolean).
Description	Returns true if the detected distance to a surface is less than (thresholded) the distance to the sanitizing chamber wall.
Function	**IsLifeformDetect()**
Designer	Nicole Baldy
Inputs	IR Sensor Data.
Outputs	Lifeform Detection (Boolean).

Description	Iterates through the IR detection grid, determines if the pixel falls within the lifeform range and thresholds the number/cluster of potential detections to determine overall detection status.
Function	**QueryUIForCommand()**
Designer	Nicole Baldy
Inputs	Transmitter Data.
Outputs	Last Transmitter Command.
Description	Updates the last UI command received.
Function	**GotStartCmd()**
Designer	Nicole Baldy
Inputs	Last Transmitter Command.
Outputs	Start Cmd (Boolean).
Description	Returns true if the last transmitter command was “Start”.
Function	**GotStopCmd()**
Designer	Nicole Baldy
Inputs	Last Transmitter Command.
Outputs	Stop Cmd (Boolean).
Description	Returns true if the last transmitter command was “Stop”.
Function	**IsEstopActive()**
Designer	Nicole Baldy
Inputs	N/A.
Outputs	Estop (Boolean).
Description	Returns true if the Estop (both button and door) are queried and either are found to be activated.
Function	**SetFault()**
Designer	Nicole Baldy
Inputs	Fault Code
Outputs	LED command Platform Motor Command Unlock command
Description	Saves the Fault code, stops the UV-C LEDs, Platform Motor, and unlocks the door.
Function	**SetLEDsOn()**
Designer	Nicole Baldy
Inputs	N/A.
Outputs	LED Command.
Description	Commands LED GPIO to turn UV-C LEDs on.
Function	**SetLEDsOff()**
Designer	Nicole Baldy

Inputs	N/A.
Outputs	LED Command.
Description	Commands LED GPIO to turn UV-C LEDs off.
Function	**SpinPlatformMotor()**
Designer	Nicole Baldy
Inputs	N/A.
Outputs	Platform Motor Command.
Description	Commands platform motor to begin to spin.
Function	**StopPlatformMotor()**
Designer	Nicole Baldy
Inputs	N/A.
Outputs	Platform Motor Command.
Description	Commands Platform motor to stop spinning.
Function	**IsCycleComplete()**
Designer	Nicole Baldy
Inputs	Time Since Cycle Start. Minimum Cycle Time.
Outputs	Cycle Complete (Boolean).
Description	Determines if enough time has passed since the sanitizing cycle started to end the cycle (5 min).

To make the control flow expected on the microcontroller simpler, it can be visualized as a high-level state machine. This state machine can be visualized in *Figure* 22; this state machine can be thought of as a simplified, state-based version of *Figure* 21, and thus shall not have its own table of functional components.

This state machine and many sensor connections have some level of completed implementation; all code for this subsystem has been uploaded to GitHub and will continue to be updated as improvements are made. This is further documented in section 5.8, the Level 3 software. (NB)

5.8. Level 3 Control System Software

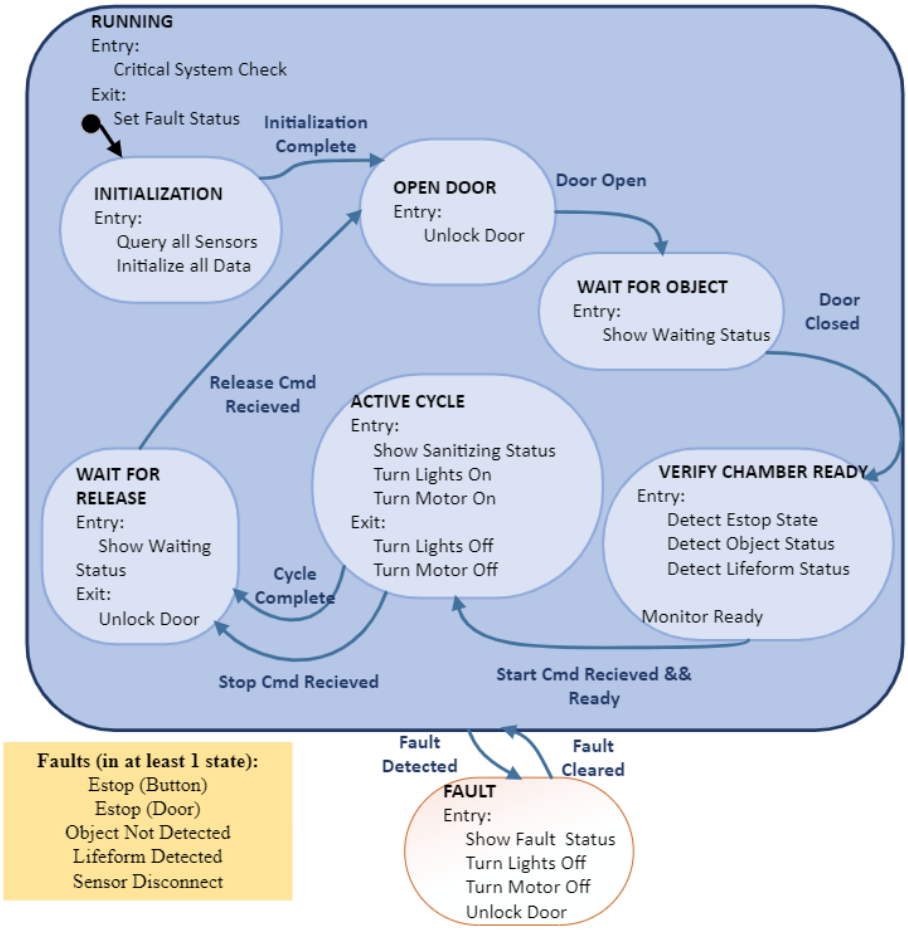


Figure 22: Microcontroller State machine diagram

Pseudocode for the control subsystem can be divided into several levels: the state machine, which runs at the highest level and uses lower-level modules; the light control peripherals, which read the status of the estop system and turn the light and motor on and off; the AMG8833 communication code, which forms a library interface for use of the grid-eye human-detection sensor; the HC-SR04 communication code, which forms a library interface for use of the ultrasonic object sensor, and the MG996R communication code, which controls the PWM connection to the lock servo. Of these libraries, a proof of concept for the first three systems (state machine, light control, and AMG8833) has

already been achieved and therefore both pseudocode and actual code will be provided. The other libraries only consist of pseudocode at this time.

5.8.1. State Machine Framework

The state machine framework pseudocode is shown in Figure 23; this is just a general framework showing the code is expected to function. Figure 24 shows the existing implementation of this state machine. Code relevant to section 5.8.1 is shown in **dark green**, and code relevant to section 5.8.2 is shown in **dark purple**.

State Machine Pseudocode [Page 1]

```
MAIN:
state = InitState
forever
{
  state = processState(state)
  waitForStateTimer()
}

HELPERS:
StopActiveCycle()
{
  turnLEDsOff()
  turnMotorOff()
}

RunningActions
{
  IF(EstopDetected())
    StopActiveCycle()
    GOTO FaultState
}

STATES:
InitState
{
  statusLEDs(INIT)

  IF(IsAMG8833Ready() AND
   IsHCSR04Ready())
    openDoor()
    GOTO WaitForObjState
  IF(InitTimeout())
    GOTO FaultState
}
```

```
WaitForObjState
{
  RunningActions()
  statusLEDs(READY_FOR_OBJ)

  IF(IsDoorClosed() AND
   IsWaitTimerDone())
    GOTO VerifyReadyState
  ELSE
    ResetWaitTimer()
}

VerifyReadyState
{
  RunningActions()
  statusLEDs(CHECK_READY)

  IF(IsDoorOpen)
    GOTO WaitForObjState

  IF (IsLifeformDetected())
    GOTO FaultState
  IF (IsObjectDetected() AND
   IsObjTimerDone())
    StartTimeoutTimer()
    GOTO WaitForStartState
  ELSE
    ResetObjTimer()
}
```

State Machine Pseudocode [Page 2]

```
ActiveCycleState
{
  RunningActions()
  statusLEDs(ACTIVE)

  turnLEDsOn()
  turnMotorOn()

  IF(GotCancelCommand() OR
   CycleTimerDone())
    StopActiveCycle()
    StartTimeoutTimer()
    GOTO WaitForReleaseState
}
```

```
WaitForReleaseState
{
  RunningActions()
  statusLEDs(STOPPING)

  IF(GotReleaseCommand() OR
   TimeoutTimerExpired())
    openDoor()
    GOTO WaitForObjState
}

FaultState
{
  statusLEDs(FAULT)

  IF(FaultConditionClear())
    GOTO InitState
}
```

Figure 23: State Machine Pseudocode

This pseudocode framework has been turned into actual state machine framework written in C, which runs on the PIC24FJ128GA010, with all of the peripherals on the Explorer 16 development board. This state machine code is still in development, but can be used to walk through the state machine diagram shown in Figure 22.

State Machine C Code Header [SmartUVStateMachine.h, Page 1]

```
typedef enum StateName
{
    STATE_UNKNOWN,
    STATE_INITIALIZATION,
    STATE_WAIT_FOR_OBJECT,
    STATE_VERIFY_CHAMBER_READY,
    STATE_WAIT_FOR_CYCLE_START,
    STATE_ACTIVE_CYCLE,
    STATE_WAIT_FOR_RELEASE,
    STATE_FAULT,
    STATE_RUNNING // Parent State Only
} StateName;
```

```
typedef enum FaultName
{
    FAULT_UNKNOWN,
    NO_FAULT,
    FAULT_ESTOP,
    FAULT_DOOR_OPEN
} FaultName
```

State Machine C Code Header [SmartUVStateMachine.h, Page 2]

```
// 8-character representation of state name for display
typedef struct StateNameStr
{
    char str[17]; // 16 char + nullcharacter
} StateNameStr;
StateNameStr getStateNameStr(enum StateName state_enumeration);

typedef struct State
{
    // Parent state or unknown if None. Parent executes first.
    enum StateName state_name;
    unsigned char display;
    enum FaultName active_fault;
    // TODO(NEB): Store last known sensor status, Estop status, etc here.
} State;

/* Takes current state, does all required actions,
 * and then returns the state to process next cycle.
 * "Public Interface", handles calls to all functions below*/
void processCurrentState(State* current_state);

State InitStateMachine();

// Treat below functions as private.
void Initialization(State* state);
void WaitForObject(State* state);
void VerifyChamberReady(State* state);
void WaitForCycleClart(State* state);
void ActiveCycle(State* state);
void WaitForRelease(State* state);
void Fault(State* state);
// "Parent" States
void Running(State* state);

void SetFault(State* state); // TODO(NEB): Fault Codes, for now just set state.
void printFaultState(FaultName fault_name);

// TODO(NEB): Temporary for state proof.
enum Buttons
{
    BUTTON_READY_FOR_NEXT=0x0001, // S4 moves forward as per "usual use case"
    BUTTON_FAULT = 0x0004, // S6 injects a fault (skip S5)
    BUTTON_CLEAR_FAULT = 0x0008 // S3 clears a fault.
};
```

Figure 24: State Machine C code header

State Machine C Code Implementation [SmartUVStateMachine.c, Page 1]

```
// Implementation file for state machine

#include "SmartUVStateMachine.h"
#include "peripherals.h"
#include "AMG88.h" // IR Grid-eye

#include   <stdio.h>
#include   <stdlib.h>
#include   <string.h>

#define LED_PIN PORTGbits.RG14
#define LED_TRIS TRISGbits.TRISG14
#define DOOR_PIN PORTGbits.RG13
#define DOOR_TRIS TRISGbits.TRISG13

// getStateNameStr defined, not relevant to implementation details.


struct State InitStateMachine()
{
    State new_sm;
    new_sm.display = 0x00;
    new_sm.state_name = STATE_UNKNOWN;
    initPortA();
    initButtons(0x000F); // All buttons as inputs: 0xF
    msDelay(100); // Give time to start up
    InitPMP();
    InitLCD();
    I2Cinit(157);

    // Use P97 = RG13 for door input
    DOOR_TRIS = 1;
    // Use P95 = RG14 for LED output
    LED_TRIS = 0;
    LED_PIN = 0;

    new_sm.active_fault = FAULT_UNKNOWN;
    new_sm.state_name = STATE_INITIALIZATION;
    return new_sm;
}
```

State Machine C Code Implementation [SmartUVStateMachine.c, Page 2]

```
void processCurrentState(State* current_state)
{
     // DISPLAY STATE NAME, Not relevant to implementation

    switch(current_state->state_name)
    {
        case STATE_INITIALIZATION:
        {
            Initialization(current_state);
            break;
        }
        case STATE_WAIT_FOR_OBJECT:
        {
            WaitForObject(current_state);
            break;
        }
        case STATE_VERIFY_CHAMBER_READY:
        {
            VerifyChamberReady(current_state);
            break;
        }
        case STATE_WAIT_FOR_CYCLE_START:
        {
            WaitForCycleClart(current_state);
            break;
        }
        case STATE_ACTIVE_CYCLE:
        {
            ActiveCycle(current_state);
            break;
        }
        case STATE_WAIT_FOR_RELEASE:
        {
            WaitForRelease(current_state);
            break;
        }
        case STATE_FAULT:
        {
            Fault(current_state);
            break;
        }
        case STATE_UNKNOWN:
        default:
        {
            // UNSUPPORTED CHILD STATE
            SetFault(current_state);
            return;
        }
    }
   // Display status port A, not relevant to implementation
}
```

State Machine C Code Implementation [SmartUVStateMachine.c, Page 3]

```
void Initialization(State* state)
{
    Running(state); // Parent State
    if (getButton(BUTTON_READY_FOR_NEXT))
    {
        // NOTE(NEB): For now, consider "initialized" when button pressed.
        // Event: Init Complete
        state->state_name = STATE_WAIT_FOR_OBJECT;
        // Clear any initialization faults
        state->active_fault = NO_FAULT;
        return;
    }

    // TODO(NEB): Initialize all sensors.
}

void WaitForObject(State* state)
{
    Running(state); // Parent State
    if (1 == DOOR_PIN) // P97 = RG13 should be used for door input
    {
        // Event: Door Closed
        state->state_name = STATE_VERIFY_CHAMBER_READY;
        return;
    }
}

void VerifyChamberReady(State* state)
{
    bReadTempFromGridEYE();
    msDelay(10); // Give time to Read Everything
    Running(state); // Parent State

    short max_pxl = maxPixel(); // 256 * Temp_In_C

    SetCursorAtLine(2);
    char str[16];
    double max_C = (double) max_pxl / (256);
    int num_pxls_body_temp = numPixelsInRange(27*256, 40*256); // between 27 and 40
C for now

    sprintf(str, "%4.2f C; r=%d", max_C, num_pxls_body_temp);

    putsLCD(str);



… <continued next page>
```

State Machine C Code Implementation [SmartUVStateMachine.c, Page 4]

```
… <continued from previous page>

    if(0 == DOOR_PIN)
    {
        // Door opened verification, wait for closed again
        state->state_name = STATE_WAIT_FOR_OBJECT;
        return;
    }
    if (getButton(BUTTON_READY_FOR_NEXT) || 3 < num_pxls_body_temp) // More than 3
pixels in this range
    {
        // NOTE(NEB): For now, consider "ready" when button pressed.
        // Event: Chamber Ready
        state->state_name = STATE_WAIT_FOR_CYCLE_START;
        return;
    }
}

void WaitForCycleClart(State* state)
{
    Running(state); // Parent State

    if(0 == DOOR_PIN)
    {
        // Door opened during wait for cycle, fault
        state->active_fault = FAULT_DOOR_OPEN;
        SetFault(state);
        return;
    }

    if (getButton(BUTTON_READY_FOR_NEXT))
    {
        // NOTE(NEB): For now, consider "started" when button pressed.
        // Event: Start Cmd Recieved
        state->state_name = STATE_ACTIVE_CYCLE;
        return;
    }
    // TODO(NEB): Wait for start cmd from wireless
}

void ActiveCycle(State* state)
{
    Running(state); // Parent State
    if(0 == DOOR_PIN)
    {
        // Door opened during active cycle, fault
        LED_PIN = 0;
        state->active_fault = FAULT_DOOR_OPEN;
        SetFault(state);
        return;
    }
… <continued next page>
```

State Machine C Code Implementation [SmartUVStateMachine.c, Page 5]

```
… <continued from previous page>

    if (getButton(BUTTON_READY_FOR_NEXT))
    {
        // NOTE(NEB): For now, consider "stopped" when button pressed.
        // Event: EITHER Stop Cmd Recieved OR Timer Expired
        state->state_name = STATE_WAIT_FOR_RELEASE;
        LED_PIN = 0;
        return;
    }

    LED_PIN = 1;

    // TODO(NEB): Wait for stop cmd from wireless OR timer complete
}

void WaitForRelease(State* state)
{
    // TODO(NEB): Unlock when get command from UI.
    Running(state); // Parent State
    if (0 == DOOR_PIN) // Move to next state when door opens/pull down
    {
        // Event: Unlock Cmd Recieved
        state->state_name = STATE_WAIT_FOR_OBJECT;
        return;
    }

    // TODO(NEB): Wait for unlock cmd
}

void Fault(State* state)
{
    if (getButton(BUTTON_CLEAR_FAULT))
    {
        // NOTE(NEB): For now, consider "fault cleared" when button pressed.
        // Event: Fault Cleared
        state->active_fault = NO_FAULT;
        state->state_name = STATE_INITIALIZATION;
        return;
    }

    printFaultState(state->active_fault);
}
```

State Machine C Code Implementation [SmartUVStateMachine.c, Page 6]

```
// Checks for faults & other "general" parent state needs
void Running(State* state)
{
    if (getButton(BUTTON_FAULT))
    {
        SetFault(state);
        return;
    }

    printFaultState(state->active_fault);
}

void SetFault(State *state)
{
    state->state_name = STATE_FAULT;
    LED_PIN = 0; // Turn LEDs off
}


// printFaultState not relevant for implementation.
```

Figure 25: State machine C code implementation

As can be seen, the state machine code is very large. As it is, this could be slightly refactored to improve readability, but for the most part should be final-design control-level code. (NB)

5.8.1. LED and Motor Control

LED and Motor control are both single-level digital inputs. The motor and LED should simply be binary; no intensity or speed control is required. Therefore, controlling these modules should be as simple as changing the status of an output pin, which can be shared as the LED and motor can always be operated together. Similarly, the Estop and door status should also be digital inputs, and only require a single input pin each to read. All but the Estop button have already been worked into the state machine, though a small library should be built to improve their readability. All code which is already working is in Figure 25 in dark green. However, the pseudocode for this library is important for improving code readability and is shown in Figure 26.

LED and motor peripherals pseudocode

```
DEFINE ESTOP_INPUT=RG12
// Tested, also controls motor
DEFINE DOOR_INPUT=RG13
DEFINE LIGHT_OUTPUT=RG14

InitializeLedSystem
{
   ConfigureTrisInput(ESTOP_INPUT)
   ConfigureTrisInput(DOOR_INPUT)
   ConfigureTrisOutput(LIGHT_OUTPUT)
}

Bool IsDoorOpen()
{
   Return (true == DOOR_INPUT)
}
```

```
Bool IsEstopActive()
{
   Return (true == ESTOP_INPUT)
}

Bool SetLedSystemActive()
{
   LIGHT_OUTPUT = TRUE
}
```

Figure 26: Light and motor pseudocode

(NB)

5.8.2. AMG8833 Grid-Eye Control

The AMG8833 grid-eye sensor uses I2C and must query a series of registers to get the temperature data; most code is provided by the AMG8833 datasheet to properly query the temperature registers (“Application Notes”, 25-28). This code was used to create a framework for the AMG88 temperature query, but three functions require implementation. These have both pseudocode, shown in Figure 27 and some implemented C code.

Grid-eye pseudocode

```
// Given a register to query, read the
value of that register
ReadAMG88I2C():
   RestartI2Cbus()
   SendI2CAddress()
   SendWriteToRegisterCommand()
   RestartI2Cbus()
   SendReadRegisterCommand()

   RETURN GetByteFromI2CBus()


Provided code:
readTempFromGridEye()
   ReadAMG88I2C(Thermisters)
   ReadAMG88I2C(Temperatures)
   pixels =
      ParseTemp(Thermisters,Temperatures)
```

```
DEFINE PIX_TO_C=@(p)/256

GetNumberOfPixelsInTempRange(degC1,
degC2):
   // Assume C1 < C2
   numInRange = 0
   FOR pixel IN pixels:
      IF degC1 < PIX_TO_C(pixel) < degC2:
         numInRange++
   RETURN numInRange


GetMaxPixelTemp():
   maxPixel = 0
   FOR pixel IN pixels:
      IF maxPixel < pixel:
         maxPixel = pixel
   RETURN PIX_TO_C(maxPixel)
```

Figure 27: Grid-eye pseudocode

(NB)

The majority of the provided code comes from the grid-eye datasheet. The altered and added code is bolded in Figure 28.

Grid-eye library implementation [Page 1]

```
#include "AMG88.h"
#include "peripherals.h"
/**************************************************************************
variable value definition
**************************************************************************/

short g_shThsTemp; /* thermistor temperature */
short g_ashRawTemp[SNR_SZ]; /* temperature of 64 pixels */

short g_maxThsTemp;

/**************************************************************************
method
**************************************************************************/

/*-------------------------------------------------------------------------
Read temperature from Grid-EYE.
-------------------------------------------------------------------------*/
bool bReadTempFromGridEYE( void )
{
    uchar aucThsBuf[GRIDEYE_REGSZ_THS];
    uchar aucTmpBuf[GRIDEYE_REGSZ_TMP];
    /* Get thermistor register value. */
    if( FALSE == bAMG_PUB_I2C_Read( GRIDEYE_ADR, GRIDEYE_REG_THS00,
GRIDEYE_REGSZ_THS, aucThsBuf ) )
    {
        return( FALSE );
    }
    /* Convert thermistor register value. */
    g_shThsTemp = shAMG_PUB_TMP_ConvThermistor( aucThsBuf );
    /* Get temperature register value. */
    if( FALSE == bAMG_PUB_I2C_Read( GRIDEYE_ADR, GRIDEYE_REG_TMP00,
GRIDEYE_REGSZ_TMP, aucTmpBuf ) )
    {
        return( FALSE );
    }
    /* Convert temperature register value. */
    vAMG_PUB_TMP_ConvTemperature64( aucTmpBuf, g_ashRawTemp );
    return( TRUE );
}
```

Grid-eye library implementation [Page 2]

```
/*---------------------------------------------------------------------------
Read data form I2C bus.
---------------------------------------------------------------------------*/
bool bAMG_PUB_I2C_Read( uchar ucI2cAddr, uchar ucRegAddr, uchar ucSize, uchar*
ucDstAddr )
{
    /* ucI2cAddr : I2C Address ( In the case of the connection with GND of
AD_SELECT PIN ) */
    /* ucRegAddr : Source address of Grid-EYE */
    /* ucSize : Data Size */
    /* ucDstAddr : Destination address of MCU */
    /* return : TRUE: success, FALSE: failure */


    uchar *arr_ptr = ucDstAddr;
    int i;
    for(i = 0; i < ucSize; i++)
    {
        // To read a register:
        // 1. Send the write command + register
        I2CStart();
        I2Csendbyte(ucI2cAddr); // Send addr, signify a read
        us_delay(100);
        I2Csendbyte(ucRegAddr + i); // Send the register to read
        us_delay(100);
        I2CStop();

        // 2. Send start command + read command.
        I2CStart();
        us_delay(100);
        I2Csendbyte(ucI2cAddr + 1); // Send addr, signify a read

        //. 3 wait for result.
        us_delay(100);
        char temp = I2Cgetbyte();
        us_delay(100);
        I2CStop();

        *arr_ptr = temp;
        arr_ptr ++;
    }

    return( TRUE );
}
```

Grid-eye library implementation [Page 3]

```
/*-------------------------------------------------------------------------
Convert thermistor register value.
-------------------------------------------------------------------------*/
short shAMG_PUB_TMP_ConvThermistor( uchar aucRegVal[2] )
{
    /* Convert to 16 bit Two's complement */
    /* bit15 : sign bit */
    /* bit14-8 : integral number bits */
    /* bit7-0 : fixed-point numbers bits */
    short shVal = ((short)(aucRegVal[1] & 0x07) << 8) | aucRegVal[0];
    if( 0 != (0x08 & aucRegVal[1]) )
    {
        shVal *= -1;
    }
    shVal *= 16;
    return( shVal );
}

/*-------------------------------------------------------------------------
Convert temperature register value for 1 pixel.
-------------------------------------------------------------------------*/
short shAMG_PUB_TMP_ConvTemperature( uchar aucRegVal[2] )
{
    /* Convert to 16 bit Two's complement */
    /* bit15 : sign bit */
    /* bit14-8 : integral number bits */
    /* bit7-0 : fixed-point numbers bits */
    short shVal = ((short)(aucRegVal[1] & 0x07) << 8) | aucRegVal[0];
    if( 0 != (0x08 & aucRegVal[1]) )
    {
        shVal -= 2048;
    }
    shVal *= 64;
    return( shVal );
}

/*-------------------------------------------------------------------------
Convert temperature register value for 64 pixel.
-------------------------------------------------------------------------*/
void vAMG_PUB_TMP_ConvTemperature64( uchar* pucRegVal, short* pshVal )
{
    uchar ucCnt;
    for( ucCnt = 0u; ucCnt < SNR_SZ; ucCnt++ )
    {
        pshVal[ucCnt] = shAMG_PUB_TMP_ConvTemperature( pucRegVal + (ucCnt * 2u) );
    }
}
```

Grid-eye library implementation [Page 4]

```
/*-----------------------------------------------------------------------------
Convert value.
-----------------------------------------------------------------------------*/
short shAMG_PUB_CMN_ConvFtoS( float fVal )
{
    return( ( fVal > 0 ) ? ((short)((fVal * 256) + 0.5)) : ((short)((fVal * 256) -
0.5)) );
}

/*-----------------------------------------------------------------------------
Convert value.
-----------------------------------------------------------------------------*/
float fAMG_PUB_CMN_ConvStoF( short shVal )
{
    return( (float)shVal / 256 );
}

int numPixelsInRange(short t1, short t2)
{
    int num_in_range = 0;
    int i;
    for (i = 0; i < SNR_SZ; i++)
    {
        if(t1 <= g_ashRawTemp[i] && g_ashRawTemp[i] < t2)
          num_in_range++;
    }

    return num_in_range;
}

short maxPixel()
{
    int largest_value = -100;
    int i;
    for (i = 0; i < SNR_SZ; i++)
    {
        if(largest_value < g_ashRawTemp[i])
          largest_value = g_ashRawTemp[i];
    }

    return largest_value;
}
```

Figure 28: AMG8833 Library Implementation

This library is not expected to change much, except for possibly filtering the readings. It has been tested for the subsystem demonstrations using the PIC24FJ128GA010 microcontroller. (NB)

5.8.3. HC-SR04 Ultrasonic Control

The HC-SR04 Ultrasonic Module is queried by a pulse from the microcontroller in the TRIG pin; it then sends out a set of 8 ultrasonic bursts and pulls up the ECHO pin for a pulse proportional to the distance from the sensor that the nearest surface was detected ("Ultrasonic Ranging Module HC - SR04", 2).

Pseudocode for this module can then be put together as in Figure 29.

HC-SR04 pseudocode

```
DEFINE TRIGGER=RG0
DEFINE ECHO=RG1
DEFINE EXPECTED_DIST_CM = 46 //~1.5 ft
DEFINE TOLERANCE_CM // Experiment
DEFINE TIMEOUT_US 3500 // About 2 ft

InitializeUltrasonicModule()
{
   ConfigureTrisInput(ECHO)
   ConfigureTrisOutput(TRIGGER)
}

Bool IsUltrasonicSensorReady()
{
   dist = GetUltrasonicDistanceCm();
   return TOLERANCE_CM <=
        abs(dist - EXPECTED_DIST_CM);
}


Bool IsObjectDetected()
{
   Dist = GetUltrasonicDistanceCm();

   return ((dist > TOLERANCE_CM) &&
      (dist <
        EXPECTED_DIST_CM - TOLERANCE_CM));
}
```

```
float GetUltrasonicDistanceCm()
{
   TRIGGER=1;
   us_delay(10);
   TRIGGER=0;

   while ( !ECHO &&
      !IsTimedOut(TIMEOUT))
   {
      // Missing 5 us=0.1cm
      // insignificant
      us_delay(5);
   }

   Start_timer();
   while ( ECHO &&
      !IsTimedOut(TIMEOUT))
   {
      // Missing 5 us=0.1cm
      // insignificant
      us_delay(5);
   }
   Time_elapsed = timer();

   Return timer/58;
}
```

Figure 29: Ultrasonic Module Pseudocode

5.8.4. Servo Motor Control

From section 2.6.2.1, and an informational sheet about the MG996R, pseudocode can be written for controlling the servo. It will be assumed that the rest position of the servo when the door is open or locked is 0 degrees, and the servo will move to 45 degrees to open the lock. These may be changed slightly from experimentation based on lock mechanics. The PWM period of the MG996R servo is 20ms, with a 5% duty cycle corresponding to -90 degrees and 10% duty cycle corresponding to +90 degrees; the lock can move 60 degrees in under 200 ms ("MG996R Metal Gear Servo Motor", 3). Using the process defined in section 2.6.2.1, it is found that to put the servo into a neutral position, a 7.5% duty cycle should be held, and to unlock the door, a duty cycle of 8.75 should be held. At least 0.2 seconds of this should be held to safely ensure the servo

MG996R pseudocode

```
DEFINE LOCK_PWM=RG15
DEFINE NEUTRAL_US = 7500;
DEFINE ACTIVE_US = 8750;
DEFINE PERIOD_US = 20000;
DEFINE ITERATIONS= 10

InitializeLock()
{
   ConfigureTrisOutput(LOCK_PWM);
}
```

```
ActivateUnlock()
{
   Low_period_us = PERIOD_US - ACTIVE_US;
   For (ITERATIONS)
   {
      LOCK_PWM = 1;
      us_delay(ACTIVE_US);
      LOCK_PWM = 0;
      us_delay(Low_period_us);
   }
}

ReleaseLock()
{
   Low_period_us = PERIOD_US - NEUTRAL_US;
   For (ITERATIONS)
   {
      LOCK_PWM = 1;
      us_delay(NEUTRAL_US);
      LOCK_PWM = 0;
      us_delay(Low_period_us);
   }
}
```

Figure 30: Servo Lock Pseudocode

makes it to the goal position. This is 10 iterations of the PWM period. This can be used to write the pseudocode in Figure 30.

5.9. Level 2 User Interface

The user interface in this project can be visualized as a flowchart. The flow chart is shown in Figure 31 and explores how specific components of the software will function together. The input and output information is located in *Table 9*.

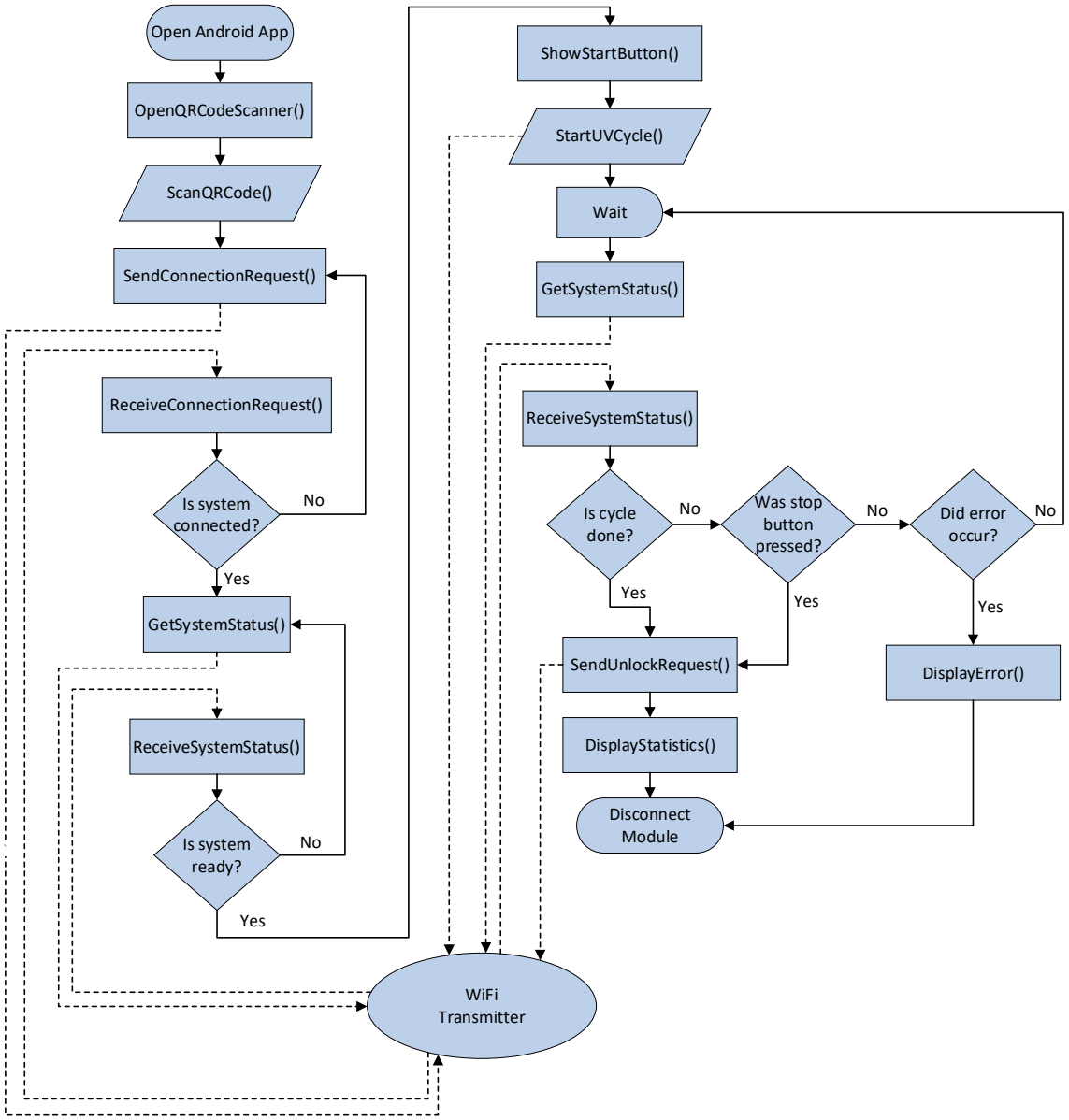


Figure 31: User Interface Level 2 Flowchart

Table 9 : Software (User Interface) Level 2 Functional Requirements Table

Function	**Open Android Application**
Designer	Luke Rogers
Inputs	N/A
Outputs	N/A
Description	The user opens the Android application on their device.
Function	**OpenQRCodeScanner()**
Designer	Luke Rogers

Inputs	N/A
Outputs	Opens the camera
Description	Opens the Android device camera for the user to scan the QR code on the side of the module.
Function	**ScanQRCode()**
Designer	Luke Rogers
Inputs	Image from the camera
Outputs	Connection signal to the module
Description	Scans the QR code on the camera view and connects to the module.
Function	**SendConnectionRequest()**
Designer	Luke Rogers
Inputs	QR code information
Outputs	Connection request
Description	Sends a connection request with information from the QR code sent to the module via a Wi-Fi transmission.
Function	**ReceiveConnectionRequest()**
Designer	Luke Rogers
Inputs	Connection status
Outputs	Connection status
Description	The connection status is received from the module via a Wi-Fi transmission.
Function	**Is system connected?**
Designer	Luke Rogers
Inputs	Connection status
Outputs	Binary signal
Description	Checks if the Android application is connected to the module: Yes – Moves to GetSystemStatus(). No – Returns to SendConnectionStatus().
Function	**GetSystemStatus()**
Designer	Luke Rogers
Inputs	Binary signal
Outputs	System status request
Description	Sends a system status request to the module via Wi-Fi transmission.
Function	**ReceiveSystemStatus()**
Designer	Luke Rogers
Inputs	System status
Outputs	System status

Description	The system status is received from the module via Wi-Fi transmission.
Function	**Is system ready?**
Designer	Luke Rogers
Inputs	System status
Outputs	Binary signal
Description	Checks if the system status is ready: Yes – Moves to ShowStartButton(). No – Returns to GetSystemStatus()
Function	**ShowStartButton()**
Designer	Luke Rogers
Inputs	Binary signal
Outputs	Boolean signal
Description	Sets the visibility for the Start button to True.
Function	**StartUVCycle()**
Designer	Luke Rogers
Inputs	User input (button)
Outputs	Start signal
Description	Sends a start signal to start the UV cycle to the module via Wi-Fi transmission.
Function	**Wait**
Designer	Luke Rogers
Inputs	Cycle status
Outputs	Cycle status
Description	Wait block until GetSystemStatus() is run during the UV cycle.
Function	**Is cycle done?**
Designer	Luke Rogers
Inputs	Cycle status
Outputs	Binary signal
Description	Checks if the received cycle status is finished from the ReceiveSystemStatus(): Yes – Moves to SendUnlockRequest() No – Moves to next binary check
Function	**Was stop button pressed?**
Designer	Luke Rogers
Inputs	Cycle status, User input (button)
Outputs	Binary signal
Description	Checks if the cycle was stopped by the user Yes – Moves to SendUnlockRequest()

	No – Moves to next binary check
Function	**Did error occur?**
Designer	Luke Rogers
Inputs	Cycle status
Outputs	Binary signal
Description	Checks to see if an error occurred during the cycle: Yes – Moves to DisplayError() No – Returns to Wait
Function	**SendUnlockRequest()**
Designer	Luke Rogers
Inputs	Cycle status
Outputs	Unlock request
Description	Sends an unlock request to the module via Wi-Fi transmission.
Function	**DisplayStatistics()**
Designer	Luke Rogers
Inputs	N/A
Outputs	Cycle statistics
Description	Displays the UV cycle statistics on the application
Function	**DisplayError()**
Designer	Luke Rogers
Inputs	Cycle status, Error state
Outputs	Error state
Description	Displays the error state that occurred during the cycle.
Function	**Disconnect Module**
Designer	Luke Rogers
Inputs	N/A
Outputs	N/A
Description	Disconnects the Android application from the module.

5.10. Level 3 User Interface

The pseudocode for the user interface can be divided into several levels: the android application, which interfaces with the user directly, and the ESP-WROOM-32 communication code, which interfaces with the microcontroller. The android application will have actual code written for the preliminary stage of the application while the ESP chip does not have code at this time.

5.10.1. Android Application

The code for the android application is written using Java and the Android Studio Integrated Development Environment (IDE). The code for the application is stored on GitHub in order to have backups and version control. This initial version of the application finds the MAC addresses of the already connected Bluetooth devices to the android device. Then, the MAC address for the ESP-WROOM-32 chip is selected and using that MAC address a message is sent to the ESP chip. There are two activities to this section, the main activity and the Bluetooth activity. The code for both activities is shown below in the MainActivity.java code and the BluetoothActivity.java code blocks. See Figure 32 for both the main activity screen and the Bluetooth activity screen.

Android Application Main Activity [MainActivity.java, Page 1]

```
package com.example.sdp_app;

public class MainActivity extends AppCompatActivity {

   @Override

   protected void onCreate(Bundle savedInstanceState) {

      super.onCreate(savedInstanceState);

      setContentView(R.layout.activity_main);

      Button btn = findViewById(R.id.button_send);

      btn.setOnClickListener(v -> {

         EditText txt = findViewById(R.id.bluetooth_message);

         String btMessage = txt.getText().toString() + "\n";

         if(((ApplicationEx)getApplication()).writeBt(btMessage.getBytes(StandardCharsets.UTF_8))){

            Toast.makeText(getApplicationContext(), "Message sent.", Toast.LENGTH_SHORT).show();

            txt.setText("");

         }

      });

   }

   @Override

   protected void onResume() {

      super.onResume();

      final ApplicationEx globalVar = (ApplicationEx) getApplicationContext();

      TextView deviceName = findViewById(R.id.device_name);

      TextView deviceMAC = findViewById(R.id.device_mac);

      deviceName.setText(globalVar.getBtDeviceName());

      deviceMAC.setText(globalVar.getBtDeviceMACAddress());

   }
```

Android Application Main Activity [MainActivity.java, Page 2]

```
    public void openBluetooth(View view) {

        Intent intent = new Intent(MainActivity.this, BluetoothActivity.class);

        startActivity(intent);

    }

}
```

Android Application Bluetooth Activity [BluetoothActivity.java, Page 1]

```
public class BluetoothActivity extends AppCompatActivity {

    private ListView lstView1, lstView2;

    private ArrayAdapter<String> arrayAdapter1, arrayAdapter2;

    private BluetoothAdapter btAdapter = BluetoothAdapter.getDefaultAdapter();

    private boolean isFindAvailabledClicked = false;

    private int REQUEST_ENABLE_BT = 1;  // used to identify adding bluetooth names

    private BluetoothSocket mBTSocket = null;

    private Set<BluetoothDevice> mPairedDevices;

    private ArrayAdapter<String> mBTArrayAdapter;

    private ListView mAvailableDevicesListView;

    private BluetoothDevice deviceToPair;

    private Handler mHandler; // Our main handler that will receive callback notifications

    // #defines for identifying shared types between calling functions

    private final static int MESSAGE_READ = 2; // used in bluetooth handler to identify
message update

    private final static int CONNECTING_STATUS = 3; // used in bluetooth handler to identify
message status
```

Android Application Bluetooth Activity [BluetoothActivity.java, Page 2]

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_bluetooth);


        final ApplicationEx globalVar = (ApplicationEx) getApplicationContext()
        TextView textView2 = (TextView) findViewById(R.id.textView2);
        TextView pairedDevices = (TextView) findViewById(R.id.paired_devices);
        Button btnFindPaired = (Button) findViewById(R.id.button_find_paired);
        Button btnFindAvailable = (Button) findViewById(R.id.button_find_available);
        lstView1 = (ListView) findViewById(R.id.pairedDeviceList);
        textView2.setVisibility(View.GONE);
        checkPermission();
        mBTArrayAdapter = new
ArrayAdapter<String>(this,android.R.layout.simple_list_item_1);
        mAvailableDevicesListView = (ListView) findViewById(R.id.availableDeviceList);
        mAvailableDevicesListView.setAdapter(mBTArrayAdapter); // assign model to view
        btnFindPaired.setOnClickListener(new View.OnClickListener() {
          @Override
          public void onClick(View v) {
            if (btAdapter == null) {
                btnFindPaired.setEnabled(false);
                textView2.setVisibility(View.VISIBLE);
            }
```

Android Application Bluetooth Activity [BluetoothActivity.java, Page 3]

```
        else {
            btnFindPaired.setEnabled(true);
            pairedDevices.setVisibility(View.VISIBLE);
            if (!btAdapter.isEnabled()) {
                Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
                startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
            }
            // Get paired devices.
            Set<BluetoothDevice> pairedDevices = btAdapter.getBondedDevices();
            ArrayList list = new ArrayList();
            if (pairedDevices.size() > 0) {
                // There are paired devices. Get the name and address of each paired device.
                for (BluetoothDevice device : pairedDevices) {
                    String deviceName = device.getName();
                    String deviceMACAddress = device.getAddress(); // MAC address
                    list.add("Name: " + deviceName + "\nMAC Address: " + deviceMACAddress);
                   arrayAdapter1 = new ArrayAdapter(BluetoothActivity.this,
              android.R.layout.simple_list_item_1, list) {
                        @Override
                        public View getView(int position, View convertView, ViewGroup parent) {
                            View view = super.getView(position, convertView, parent);
                            TextView tv = (TextView) view.findViewById(android.R.id.text1);
```

Android Application Bluetooth Activity [BluetoothActivity.java, Page 4]

```
                        tv.setTextColor(Color.WHITE);
                        return view;
                    }
                };
                lstView1.setAdapter(arrayAdapter1);
            }
        }
        else {
            list.add("There are no paired devices.");
            arrayAdapter1 = new ArrayAdapter(BluetoothActivity.this,
android.R.layout.simple_list_item_1, list) {
                @Override
                public View getView(int position, View convertView, ViewGroup parent) {
                    View view = super.getView(position, convertView, parent);
                    TextView tv = (TextView) view.findViewById(android.R.id.text1);
                    tv.setTextColor(Color.WHITE);
                    return view;
                }
            };
            lstView1.setAdapter(arrayAdapter1);
        }
    }
  }
});
```

Android Application Bluetooth Activity [BluetoothActivity.java, Page 5]

```
btnFindAvailable.setOnClickListener(new View.OnClickListener() {

        @Override

        public void onClick(View v) {

          isFindAvailabledClicked = true;

          mBTArrayAdapter.clear(); // clear items

          btAdapter.startDiscovery();

          // Create a BroadcastReceiver for ACTION_FOUND.

          final BroadcastReceiver receiver = new BroadcastReceiver() {

            @Override

            public void onReceive(Context context, Intent intent) {

              String action = intent.getAction();

              ArrayList list2 = new ArrayList();

              if (BluetoothDevice.ACTION_FOUND.equals(action)) {

                // Discovery has found a device. Get the BluetoothDevice

                // object and its info from the Intent.

                BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);


                // add the name to the list

                mBTArrayAdapter.add(device.getName() + "\n" + device.getAddress());

                mBTArrayAdapter.notifyDataSetChanged();

              }
            }
          };
          // Register for broadcasts when a device is discovered.

          IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);

          registerReceiver(receiver, filter);

        }
      });
```

Android Application Bluetooth Activity [BluetoothActivity.java, Page 6]

```
        lstView1.setOnItemClickListener(new AdapterView.OnItemClickListener() {

            @Override

            public void onItemClick(AdapterView<?> parent, View view, int position, long id) {

                String[] btDevice = ((String) ((TextView) view).getText()).split(": |\n");

                String btName = btDevice[1];

                String btMAC = btDevice[3];



                globalVar.setBtDeviceName(btName);

                globalVar.setBtDeviceMACAddress(btMAC);

            }

        });

    }

    public void checkPermission() {

        if (Build.VERSION.SDK_INT >= 23) {

            if (checkSelfPermission(android.Manifest.permission.ACCESS_FINE_LOCATION) ==
PackageManager.PERMISSION_GRANTED &&
checkSelfPermission(android.Manifest.permission.ACCESS_COARSE_LOCATION) ==
PackageManager.PERMISSION_GRANTED) {



            } else {

                ActivityCompat.requestPermissions(this, new String[]{

                        android.Manifest.permission.ACCESS_FINE_LOCATION,

                        android.Manifest.permission.ACCESS_COARSE_LOCATION,}, 1);

            }

        }

    }
```

Android Application Bluetooth Activity [BluetoothActivity.java, Page 7]

```
    @Override

    public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions,
@NonNull int[] grantResults) {

        super.onRequestPermissionsResult(requestCode, permissions, grantResults);

        if (requestCode == 1 && grantResults[0] == PackageManager.PERMISSION_GRANTED &&
grantResults[1] == PackageManager.PERMISSION_GRANTED) {

        } else {

            checkPermission();

        }

    }

    private void pairDevice(BluetoothDevice device) {

        try {

            Method method = device.getClass().getMethod("createBond", (Class[]) null);

            method.invoke(device, (Object[]) null);

        } catch (Exception e) {

            e.printStackTrace();

        }

    }

}
```

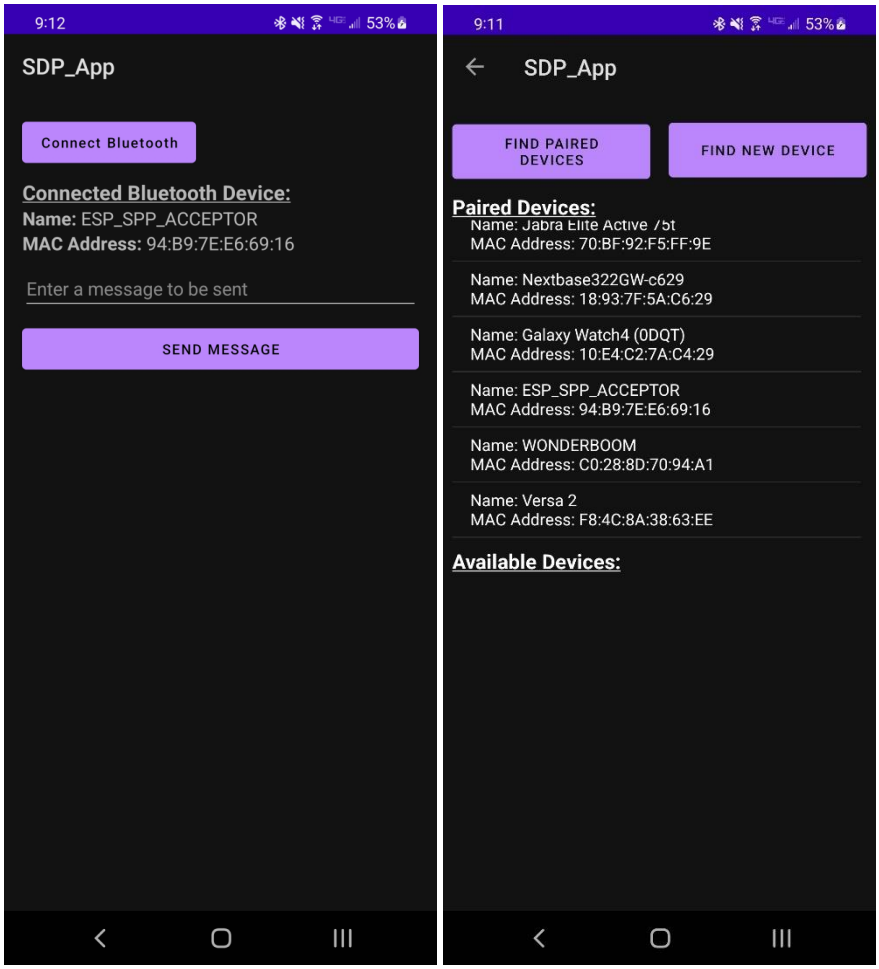


Figure 32:Android application example with the main activity screen on the left and the Bluetooth activity screen on the right.

6. Mechanical Sketch

Figure 33 shows a mechanical view of the system – it includes the UV-C LEDs, the IR sensor (for person detection), the rotating platform motor, status LEDs, and gate sensor. The device will be approximately 2’ x 2’ x 2’ and UV-C LEDs will light the back, top, and bottom of the sanitizing chamber in a single line on one side of the module. This way, when the platform motor rotates, the entire object will be exposed to UV-C light. *Figure* 34 shows a sketch of the Android application for the user interface. The user interface has buttons for starting and stopping the UV-C light cycle. In addition, the user interface shows

how much time is remaining on the current cycle. Finally, the user interface shows various statistics about the light cycles and any faults that occurred during the cycle.

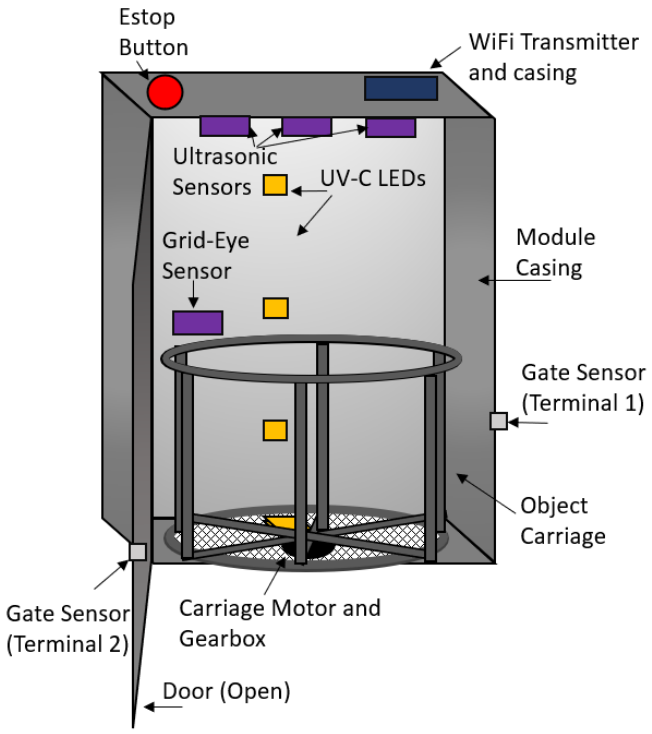


Figure 33: Mechanical Sketch of the System

(NB, with input from LR, HS, HSA)

It can be seen in the mechanical sketch that the LEDs line a single side of the wall, ceiling, and bottom of the sanitizing chamber. As discussed in section 2.2.1.1, 10 LEDs with 5 parallel sets of 2 series LEDs shall be used. Therefore, each of the 4 visible LED symbols in the diagram represent 2 actual LEDs in parallel; not in the diagram due to the angle are the 2 LEDs on the ceiling. The grid-eye sensor shall be positioned in the middle of the back wall of the chamber so that it can detect a human even when the door is open and will not pick up as many LEDs if they are close to human body temperature. The Estop button should be on the top or front of the module such that it can be easily reached in the case of a malfunction.

The user interface should contain the options to connect to the module, display usage information, and either start or stop the cycle depending on the current state of the system.

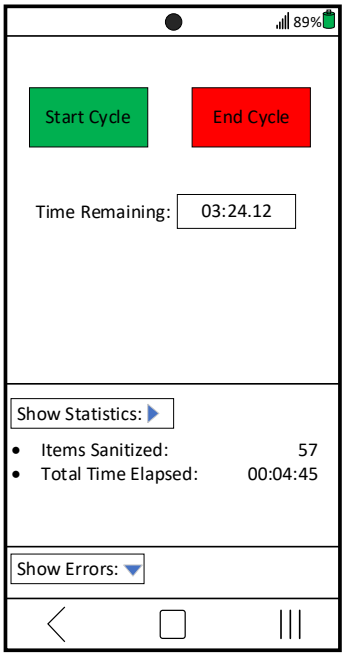


Figure 34: Sketch of the Android Application for the User Interface

(LR, with input from NB, HS)

7. Design Team Information

The Design Team which will be responsible for finalizing the design and implementing the project consists of three students:

Current Members of Design Team 12:

Nicole Baldy, Computer Engineering.

Subsystem responsibilities: Control system.

Luke Rogers, Computer Engineering.

Subsystem responsibilities: User Interface.

Haitham Saleh, Electrical Engineering

Subsystem responsibilities: Electrical system.

Previous Members of Design Team 12:

Kyle Law, Electrical Engineering

Hailey Saint-Amand, Electrical Engineering

8. Parts List

8.1. Parts List

All schematic components from hardware sections 5.4 and 5.5 have had part numbers decided upon. Some mechanical components will be decided on in the next week (such as sheet plywood for the module walls). These were still added to the parts list as expected parts. The parts ordered in the first part request for prototyping but will not be used in the final design have been greyed out in **Table *10***.

Table 10: Part List

Qty.	Refdes	Part Num.	Description
1	U2	ESP32	WiFi/Bluetooth + development board for initial prototyping
1	U3	SEN-14607	IR Grid-eye I2C Sensor w/ development board
4	U4-U7	HC-SR04	Parallax Ping Ultrasonic Sensor
20	LED1-20	WP7113ID	WP7113ID
1	M1	HG37-300-AB-00	High-torque (>3Nm) low-speed motor
1	M2	FXX-3037-TOP	Servo Motor (position control, non-continuous)
10	R1-R10	AC05AT0003009JAC00	30 olm Resistor
2	P1, P2	PDB181-A425K-203B	Linear Turn Potentiometer
2	H1, H2	TBD	Strip Metal
1	H3	TBD	Spring
1	H4	TBD	Washer for motor
4	H5-H8	TBD	Plywood for box
24	H9-H32	TBD	Square brackets for box
1	H33	TBD	Chicken Wire for sanitizing platform
1	Reg 4	TBD	9 V Regulator
4	REL1	833H-1A-S-5VDC	3.3V Relay
1	M1	HG37-200-AB-00	Higher Torque Motor
4	D1, D2, D3, D4	1N4002RLG	Rectifier Diodes
2	R11, R12	CBT50J100R	Carbon Composition 100 ohm resistor
2	REG1, REG2	LM350AT/NOPB	3 Amp adjustable Regulator
1	C1	T110A105K035AT	Tantalum Capacitors 1uF
1	C2	S104Z93Z5VL83L0R	Ceramic Disc Capacitors 0.1uF
1	C3	ESY106M035AB2AA	Aluminum Electrolytic Capacitors 10uF
1	R13	CBT25J1K0	Carbon Composition Resistors 1 KOhms

20	LED21-31	GD PSLR31.13-1U3U-25-1-150-R18	High Power LEDs - Single Color Blue
1	REG 3	UA78M05CKCSE3	5 V regulator

Error! Not a valid link.

8.2. Materials Budget

Many schematic components from hardware sections 5.4 and 5.5 have been ordered; however, some mechanical components will be decided on in the next week (such as sheet plywood for the module walls). These have been given approximate budgets based on a preliminary search of the McMaster Carr catalog. However, the actual amounts are expected to change slightly. Parts which have already been bought in the first part request but will not be used in the final design have been greyed out in **Table *11***.

Table 11: Materials Budget List

Qty.	Part Num.	Description	Cost	Cost
1	ESP32	WiFi/Bluetooth + development board for initial prototyping	$9.00	$9.00
1	SEN-14607	IR Grid-eye I2C Sensor w/ development board	43.75	43.75
4	HC-SR04	Parallax Ping Ultrasonic Sensor	5.95	23.80
20	WP7113ID	WP7113ID	0.28	5.54
1	HG37-300-AB-00	High-torque (>3Nm) low-speed motor	23.99	23.99
1	FXX-3037-TOP	Servo Motor (position control, non-continuous)	12.50	12.50
10	AC05AT0003009JAC00	30 olm Resistor	0.54	5.40
2	PDB181-A425K-203B	Linear Turn Potentiometer	0.52	1.04
2	TBD	Strip Metal	20.00	40.00
1	TBD	Spring	5.00	5.00
1	TBD	Washer for motor	5.00	5.00
4	TBD	Plywood for box	14.00	56.00
24	TBD	Square brackets for box	0.93	22.32
1	TBD	Chicken Wire for sanitizing platform	16.45	16.45
1	TBD	9 V Regulator	4.00	4.00
4	833H-1A-S-5VDC	3.3V Relay	1.31	5.24
1	HG37-200-AB-00	Higher Torque Motor	24.99	24.99
4	1N4002RLG	Rectifier Diodes	0.32	1.28
2	CBT50J100R	Carbon Composition 100 ohm resistor	0.71	1.42
2	LM350AT/NOPB	3 Amp adjustable Regulator	3.62	7.24
1	T110A105K035AT	Tantalum Capacitors 1uF	3.90	3.90
1	S104Z93Z5VL83L0R	Ceramic Disc Capacitors 0.1uF	1.74	1.74

1	ESY106M035AB2AA	Aluminum Electrolytic Capacitors 10uF	0.22	0.22
1	CBT25J1K0	Carbon Composition Resistors 1 KOhms	0.71	0.71
20	GD PSLR31.13-1U3U-25-1-150-R18	High Power LEDs - Single Color Blue	0.53	10.60
1	UA78M05CKCSE3	5 V regulator	3.65	3.65
			Total	$334.78

9. Project Schedules

A Gantt Chart for the creation of the report and project implementation has been created and is pictured below in Figure 35, Figure 36, and Figure 37. Note that this covers both the Fall 2021 and Spring 2022 semesters.

ID	Task Name	Duration	Start	Finish
1	SDP I 2021			
2	**Project Design**	**191.38 days?**	**Wed 8/25/21**	**Fri 3/4/22**
3	**Midterm Report**	46 days	Wed 8/25/21	Sun 10/10/21
4	Cover page	46 days	Wed 8/25/21	Sun 10/10/21
5	T of C, L of T, L of F	46 days	Wed 8/25/21	Sun 10/10/21
6	**Problem Statement**	**46 days**	**Wed 8/25/21**	**Sun 10/10/21**
7	Need	46 days	Wed 8/25/21	Sun 10/10/21
8	Objective	46 days	Wed 8/25/21	Sun 10/10/21
9	Background	46 days	Wed 8/25/21	Sun 10/10/21
10	Marketing Requirements	46 days	Wed 8/25/21	Sun 10/10/21
11	Engineering Requirements Specification	46 days	Wed 8/25/21	Sun 10/10/21
12	**Engineering Analysis**	**46 days?**	**Wed 8/25/21**	**Sun 10/10/21**
13	Circuits (DC, AC, Power, …)	46 days	Wed 8/25/21	Fri 3/4/22
14	Electronics (analog and digital)	46 days	Wed 8/25/21	Fri 3/4/22
15	Signal Processing	46 days	Wed 8/25/21	Sun 10/10/21
16	**Communications (analog and digital)**	46 days?	**Wed 8/25/21**	**Sun 10/10/21**
17	User Interface	1 day?	Wed 8/25/21	Fri 8/27/21
18	Microcontroller	1 day?	Wed 8/25/21	Fri 8/27/21
19	Electromechanics	46 days	Wed 8/25/21	Fri 3/4/22
20	Computer Networks	46 days	Wed 8/25/21	Fri 3/4/22
21	Embedded Systems	46 days	Wed 8/25/21	Fri 3/4/22
22	Controls	46 days	Wed 8/25/21	Fri 3/4/22
23	**Accepted Technical Design**	**46 days**	**Wed 8/25/21**	**Sun 10/10/21**
24	**Hardware Design: Phase 1**	**46 days**	**Wed 8/25/21**	**Sun 10/10/21**
25	Circuits Block Diagram	46 days	Wed 8/25/21	Fri 3/4/22
26	**Software Design: Phase 1**	**46 days**	**Wed 8/25/21**	**Sun 10/10/21**
27	Software Microcontroller	46 days	Wed 8/25/21	Fri 3/4/22
28	Software User Interface	46 days	Wed 8/25/21	Fri 3/4/22
29	**Mechanical Sketch**	46 days	Wed 8/25/21	Sun 10/10/21
30	**Team information**	46 days	Wed 8/25/21	Sun 10/10/21
31	**Project Schedules**	**46 days**	**Wed 8/25/21**	**Sun 10/10/21**
32	Midterm Design Gantt Chart	46 days	Wed 8/25/21	Sun 10/10/21
33	**References**	46 days	Wed 8/25/21	Sun 10/10/21
34	**Midterm Parts Request Form**	50 days	Wed 8/25/21	Thu 10/14/21
35	**Midterm Design Presentations Day 1**	0 days	Wed 9/22/21	Wed 9/22/21
36	**Midterm Design Presentations Day 2**	0 days	Wed 9/29/21	Wed 9/29/21
37	**Project Poster**	14 days	Sun 10/10/21	Sun 10/24/21
38	**Final Design Report**	45 days	Sun 10/10/21	Wed 11/24/21
39	**Abstract**	45 days	Sun 10/10/21	Wed 11/24/21
40	**Hardware Design: Phase 2**	**45 days?**	**Sun 10/10/21**	**Wed 11/24/21**
41	**Power System**	**45 days?**	**Sun 10/10/21**	**Wed 11/24/21**

Figure 35: Gantt Chart (Part 1)

ID	Task Name	Duration	Start	Finish
42	**AC-DC Adapter**	**45 days**	**Sun 10/10/21**	**Wed 11/24/21**
43	Simulations	45 days	Sun 10/10/21	Wed 11/24/21
44	Schematics	45 days	Sun 10/10/21	Wed 11/24/21
45	**Voltage Regulator**	**44.38 days**	**Sun 10/10/21**	**Tue 11/23/21**
46	Simulations	45 days	Sun 10/10/21	Wed 11/24/21
47	Schematics	45 days	Sun 10/10/21	Wed 11/24/21
48	**Detection System**	**22.38 days**	**Sun 10/10/21**	**Mon 11/1/21**
49	**Ultrasonic Sensor**	**22 days**	**Sun 10/10/21**	**Mon 11/1/21**
50	Simulations	45 days	Fri 9/17/21	Mon 11/1/21
51	Schematics	45 days	Fri 9/17/21	Mon 11/1/21
52	**Grid-Eye Sensor**	**22 days**	**Sun 10/10/21**	**Mon 11/1/21**
53	Simulations	45 days	Fri 9/17/21	Mon 11/1/21
54	Schematics	45 days	Fri 9/17/21	Mon 11/1/21
55	**Cage Motor System**	**22 days**	**Sun 10/10/21**	**Mon 11/1/21**
56	**Motor**	**22 days**	**Sun 10/10/21**	**Mon 11/1/21**
57	Simulations	45 days	Fri 9/17/21	Mon 11/1/21
58	Schematics	45 days	Fri 9/17/21	Mon 11/1/21
59	**Cage Platform**	**22 days**	**Sun 10/10/21**	**Mon 11/1/21**
60	Simulations	45 days	Fri 9/17/21	Mon 11/1/21
61	Diagrams	45 days	Fri 9/17/21	Mon 11/1/21
62	**Lock System**	**22 days**	**Sun 10/10/21**	**Mon 11/1/21**
63	**Servo**	**22 days**	**Sun 10/10/21**	**Mon 11/1/21**
64	Simulations	45 days	Fri 9/17/21	Mon 11/1/21
65	Schematics	45 days	Fri 9/17/21	Mon 11/1/21
66	**Lock Springs**	**22 days**	**Sun 10/10/21**	**Mon 11/1/21**
67	Simulations	45 days	Fri 9/17/21	Mon 11/1/21
68	Schematics	45 days	Fri 9/17/21	Mon 11/1/21
69	**Estop System**	**21.38 days**	**Mon 11/1/21**	**Mon 11/22/21**
70	**Estop Button**	**21.38 days**	**Mon 11/1/21**	**Mon 11/22/21**
71	Simulations	22 days	Mon 11/1/21	Tue 11/23/21
72	Schematics	22 days	Mon 11/1/21	Tue 11/23/21
73	**Gate Sensor**	**21.38 days**	**Mon 11/1/21**	**Mon 11/22/21**
74	Simulations	22 days	Mon 11/1/21	Tue 11/23/21
75	Schematics	22 days	Mon 11/1/21	Tue 11/23/21
76	**Wireless Transmitter**	**21.38 days**	**Mon 11/1/21**	**Mon 11/22/21**
77	Simulations	22 days	Mon 11/1/21	Tue 11/23/21
78	Schematics	22 days	Mon 11/1/21	Tue 11/23/21
79	**Microcontroller**	**22.38 days**	Mon 11/1/21	Tue 11/23/21
80	Simulations	22 days	Mon 11/1/21	Tue 11/23/21
81	Schematics	22 days	Mon 11/1/21	Tue 11/23/21
82	**LED System**	44.38 days	**Sun 10/10/21**	**Tue 11/23/21**

Figure 36: Gantt Chart (Part 2)

ID	Task Name	Duration	Start	Finish
83	**UV-C LEDs**	44.38 days	**Sun 10/10/21**	**Tue 11/23/21**
84	Simulations	45 days	Sun 10/10/21	Wed 11/24/21
85	Schematics	45 days	Sun 10/10/21	Wed 11/24/21
86	**Visible LEDs**	**45 days**	**Sun 10/10/21**	**Wed 11/24/21**
87	Simulations	45 days	Sun 10/10/21	Wed 11/24/21
88	Schematics	45 days	Sun 10/10/21	Wed 11/24/21
89	**Status LED Board**	**45 days**	**Sun 10/10/21**	**Wed 11/24/21**
90	Simulations	45 days	Sun 10/10/21	Wed 11/24/21
91	Schematics	45 days	Sun 10/10/21	Wed 11/24/21
92	**Software Design: Phase 2**	**45 days?**	**Sun 10/10/21**	**Wed 11/24/21**
93	**UI Software**	**45 days**	**Sun 10/10/21**	**Wed 11/24/21**
94	Code (working subsystems)	45 days	Sun 10/10/21	Wed 11/24/21
95	System integration Behavior Models	45 days	Sun 10/10/21	Wed 11/24/21
96	**Microcontroller Software**	**44.38 days**	**Sun 10/10/21**	**Tue 11/23/21**
97	Code (working subsystems)	45 days	Sun 10/10/21	Wed 11/24/21
98	System integration Behavior Models	45 days	Sun 10/10/21	Wed 11/24/21
99	**Parts Lists**	**45 days**	**Sun 10/10/21**	**Wed 11/24/21**
100	Parts list(s) for Schematics	44.38 days	Sun 10/10/21	Tue 11/23/21
101	Materials Budget list	44.38 days	Sun 10/10/21	Tue 11/23/21
102	**Proposed Implementation Gantt Chart**	44.38 days	Sun 10/10/21	Tue 11/23/21
103	**Conclusions and Recommendations**	45 days	Sun 10/10/21	Wed 11/24/21
104	Parts Request Form for Subsystems	34.38 days	Wed 9/22/21	Tue 10/26/21
105	Subsystems Demonstrations Day 1	0 days	Wed 11/10/21	Wed 11/10/21
106	Subsystems Demonstrations Day 2	0 days	Wed 11/17/21	Wed 11/17/21
107	Parts Request Form for Spring Semester	8.38 days	Tue 11/23/21	Wed 12/1/21
108	**Winter Break Parts Testing**	17 days?	Mon 12/20/21	Thu 1/6/22
109	Get Servo and HC-SR04 Working	4.67 days?	Mon 12/20/21	Thu 1/6/22
110	Get Simulations working, Voltage Regulators, start designing PCBs	4.67 days?	Mon 12/20/21	Thu 1/6/22
111	Get ESP32 Working with I2C, Make Box	4.67 days	Mon 12/20/21	Thu 1/6/22
112	**Combine Subsystems on Table**	**21.38 days?**	**Mon 1/10/22**	**Mon 1/31/22**
113	State Machine + User Input	5.33 days?	Mon 1/10/22	**Mon 1/31/22**
114	Voltage Regulators + Electronics	5.33 days?	Mon 1/10/22	**Mon 1/31/22**
115	Continue Making PCBs	21.38 days?	Mon 1/10/22	**Fri 4/8/22**
116	**Put Systems into Box**	**11.38 days?**	**Tue 2/1/22**	**Sat 2/12/22**
117	Lock System	3 days?	**Tue 2/1/22**	**Fri 2/11/22**
118	Clean up Systems & Testing	3 days?	**Tue 2/1/22**	**Fri 2/11/22**
119	Assemble PCBs	1 day?	**Mon 2/14/22**	Wed 2/16/22
120	System Testing + Bug Fixing	22.67 days?	Wed 2/16/22	Fri 5/20/22

Figure 37: Gantt Chart (Part 3).

The majority of the implementation semester will be focused on combining subsystems and cleaning them up into PCBs which are located inside of the box, such that the electrical housing is more professional.

10. Conclusion and Recommendations

This project shall be capable of disinfecting objects following the marketing and engineering requirements in sections 1.4 and 3, respectively. This project shall consist of 3 subsystems: the control subsystem, which determines the current state of the system; the electronics subsystem, which is primarily concerned with powering the system and analog outputs; and the user interface subsystem, which is primarily concerned with getting inputs from the user on their personal android device. (NB)

11.References

"Application Notes." *Robot-Electronics.co.uk*, Panasonic, www.robot-electronics.co.uk/files/grideyeappnote.pdf.

"Bluetooth Technology Overview." *Bluetooth® Technology Website*, https://www.bluetooth.com/learn-about-bluetooth/tech-overview/.

Boston Electronics. *265nm UVC LED*, WS3535C48LF-265 datasheet, www.boselec.com/wp-content/uploads/Linear/Violumas/ViolumasLiterature/265nm-UVLED-Catalog-BEC-1.pdf.

Buonanno, Manuela, et al. "Far-UVC Light (222 Nm) Efficiently and Safely Inactivates Airborne Human Coronaviruses." *Scientific Reports*, vol. 10, no. 1, June 2020, pp. 1–8. *EBSCOhost*, doi:10.1038/s41598-020-67211-2.

Buonanno, Manuela, et al. "Germicidal Efficacy and Mammalian Skin Safety of 222-Nm UV Light." *Radiation Research*, vol. 187, no. 4, 2017, pp. 493–501., doi:10.1667/rr0010cc.1.

"Carry-on Bags − Travel Information − American Airlines." *www.aa.com*, American Airlines, www.aa.com/i18n/travel-info/baggage/carry-on-baggage.jsp.

Center for Devices and Radiological Health. "UV Lights and Lamps: Ultraviolet-C Radiation, Disinfection, and Coronavirus." *U.S. Food and Drug Administration*, 1 Feb. 2021, www.fda.gov/medical-devices/coronavirus-covid-19-and-medical-devices/uv-lights-and-lamps-ultraviolet-c-radiation-disinfection-and-coronavirus.

Chen, Linmei. 陈林美.一种公共厕所自动消毒装置 *[Automatic sterilization device for public toilet]*. 5 May 2014. *Google Patents*, www.patents.google.com/patent/CN103977445A. Also accessed from worldwide.espacenet.com/patent/search?q=pn%3DCN103977445A

"Energy Efficiency of LEDs." *Energy.gov*, U.S. Department of Energy, www1.eere.energy.gov/buildings/publications/pdfs/ssl/led_energy_efficiency.pdf.

"ESP32-¬WROOM¬-32 Datasheet." *Mouser.com*, Espressif Systems, 2020, www.mouser.com/datasheet/2/891/esp32_wroom_32_datasheet_en-1510934.pdf.

"Event Planning Services, Meeting Rooms 318, 321, & 322." The University of Akron Jean Hower Student Union, The University of Akron, www.uakron.edu/studentunion/event-services/room-318321322.dot.

"FastStats - Body Measurements." *Centers for Disease Control and Prevention*, 14 Jan. 2021, www.cdc.gov/nchs/fastats/body-measurements.html.

Fraser, Charles J. "2 - Electrical and Electronics Principles." *Mechanical Engineer's Reference Book*, edited by Edward H Smith, 12th ed., Butterworth-Heinemann, 1994, pp. 2–1-2–57.

Heßling, Martin et al. “Ultraviolet irradiation doses for coronavirus inactivation - review and analysis of coronavirus photoinactivation studies.” *GMS hygiene and infection control vol*. 15 Doc08. 14 May. 2020, doi:10.3205/dgkh000343. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7273323/

Haes, Donald, and Mitchell Galanek. “Chapter 11 Radiation: Nonionizing and Ionizing Sources.” *Handbook of Occupational Safety and Health*, by S. Mansdorf, 3rd ed., John Wiley, 2019. *ORielly*, learning.oreilly.com/library/view/handbook-of-occupational/9781118947265/c11.xhtml.

Hopkins, Timothy Nevin. *Portable Light Source Including White and Ultraviolet Light Sources.* 24 Jul. 2007. U.S. Patent 7246920B1. *Google Patents*, patents.google.com/patent/US7246920.

“I2C - Learn.sparkfun.com.” *Sparkfun.com*, 2018, learn.sparkfun.com/tutorials/i2c/all.

“Infrared Array Sensor Grid-Eye (AMG88).” *Mouser.com*, Panasonic, 2 Apr. 2017, www.mouser.com/datasheet/2/813/Grid_EYE_Datasheet-2490109.pdf.

International Commission on Non-Ionizing Radiation Protection. “ICNIRP Guidelines on Limits of Exposure to Ultraviolet Radiation of Wavelengths between 180 nm and 400 nm (Incoherent Optical Radiation).” *Health Physics*, vol. 87, no. 2, Aug. 2004, pp. 171–186., www.icnirp.org/cms/upload/publications/ICNIRPUV2004.pdf.

Jelden, Katelyn C., et al. “Comparison of Hospital Room Surface Disinfection Using a Novel Ultraviolet Germicidal Irradiation (UVGI) Generator.” *Journal of Occupational & Environmental Hygiene*, vol. 13, no. 9, Sept. 2016, pp. 690–698. *EBSCOhost*, doi:10.1080/15459624.2016.1166369.

Kesavan, Jana S, and Jose L Sagripanti. “Chapter 17: Disinfection of Airborne Organisms by Ultraviolet-C Radiation and Sunlight.” *Aerosol Science: Technology and Applications*, by Mihalis Lazaridis and Ian Colbeck, John Wiley, 2014. *ORielly*, learning.oreilly.com/library/view/handbook-of-occupational/9781118947265/c11.xhtml.

Kim, Do-Kyun, and Dong-Hyun Kang. “UVC LED Irradiation Effectively Inactivates Aerosolized Viruses, Bacteria, and Fungi in a Chamber-Type Air Disinfection System.” *Applied and Environmental Microbiology*, vol. 84, no. 17, 2018, doi:10.1128/aem.00944-18.

Kitagawa, Hiroki, et al. “Effectiveness of 222-Nm Ultraviolet Light on Disinfecting SARS-CoV-2 Surface Contamination.” *American Journal of Infection Control*, vol. 49, no. 3, 2021, pp. 299–301. *Crossref*, doi:10.1016/j.ajic.2020.08.022.

McGinn, Conor et al. “Exploring the Applicability of Robot-Assisted UV Disinfection in Radiology*.” Frontiers in robotics and AI* vol. 7 590306. 6 Jan. 2021, doi:10.3389/frobt.2020.590306

“MG996R Metal Gear Servo Motor.” *Handsontec.com*, Handson Tech, handsontec.com/dataspecs/motor_fan/MG996R.pdf.

Nakao, Takashi et al. 風呂場や調理室、トイレ室等の衛生を保ち、ま *[Device for lighting ultraviolet sterilization lamp without detection of human sensor and by lights-out of lighting device]*. 30 Jul. 2015. *Google Patents*, patents.google.com/patent/JP2017029293A/en. Also accessed from worldwide.espacenet.com/patent/search?q=pn%3DJP2017029293A.

"Servo Diagram." *ArcBotics: Learning with Robots*, ArcBotics, static.arcbotics.com/wp-content/uploads/2014/01/Servo-Diagram.jpeg.

Shen, Wenlong, et al. "Secure Device-to-Device Communications over WiFi Direct." *IEEE Network*, vol. 30, no. 5, Sept. 2016, pp. 4–9. *EBSCOhost*, doi:10.1109/MNET.2016.7579020.

Shetty, Akshaya D., et al. "Detection and Tracking of a Human Using the Infrared Thermopile Array Sensor — 'Grid-EYE.'" *2017 International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT)*, 2017, doi:10.1109/icicict1.2017.8342790.

Sikder, Ratul, et al. "A Survey on Android Security: Development and Deployment Hindrance and Best Practices." *Telkomnika*, vol. 18, no. 1, Feb. 2020, pp. 485–499. *EBSCOhost*, doi:10.12928/TELKOMNIKA.v18i1.13288.

The Skin Cancer Foundation. "Skin Cancer Facts & Statistics." *The Skin Cancer Foundation*, 21 Jan. 2021, www.skincancer.org/skin-cancer-information/skin-cancer-facts.

"Specifications for Infrared Array Sensor ." *Adafruit.com*, Panasonic, 2 Dec. 2002, cdn-learn.adafruit.com/assets/assets/000/043/261/original/Grid-EYE_SPECIFICATIONS%28Reference%29.pdf?1498680225.

Stijn, Derammelaere, et al. "A Quantitative Comparison between BLDC, PMSM, Brushed DC and Stepping Motor Technologies." *International Conference on Electrical Machines and Systems (ICEMS)*, vol. 19, 2016, pp. 1–5, biblio.ugent.be/publication/8544760/file/8544762.pdf.

"Stuffed Backpacks: How Much Weight Is Too Much." *Independent School Management*, 2 Oct. 2012, isminc.com/advisory/publications/the-source/stuffed-backpacks-how-much-weight-is-too-much.

"Ultrasonic Distance Measurement Module." *Freesvg.org*, Free SVG, 10 Sept. 2019, freesvg.org/ultrasonic-distance-measurement-module.

"Ultrasonic Ranging Module HC - SR04." *Sparkfun.com*, Elecfreaks, cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf.

"Understanding Bluetooth Range." *Bluetooth® Technology Website*, https://www.bluetooth.com/learn-about-bluetooth/key-attributes/range/.

(NB, LR, HS, KL)