University of Southern Indiana
Pott College of Science, Engineering, and Education
Engineering Department
8600 University Boulevard
Evansville, Indiana 47712

Automated Medicine Sorter/Counter/Cutter

Brayden Scarlett and Langdon Briles
ECE 471 - Senior Project
Fall 2021

Acknowledgements

In this section, we would like to acknowledge,

Devon Rolfe

He was an original member of our team and he, unfortunately, did not enroll for the Fall of 2021 and is no longer part of the team. He helped us get the project started and had important input at the beginning of this project.

Dr. Paul Kuban and the USI Department of Engineering

He is the head of the engineering department and together with the Department of Engineering, facilitated and enabled this project to be done.

Dr. Ronald Diersing

He is a ECE engineering professor. In the initial stages of the project, our team did not have a concrete idea for a project. His criticisms and advice pushed us to pick a more concise project idea as shown in this report.

Dr. Julian Davis

He is the customer of the project. He provided the idea for the project as well as numerous feedback on design ideas and choices. He also provided support to team outside of project.

Dr. Mina Asghari Heidarlou

She is an ECE engineering professor and this project's advisor. She ensured we kept on track with our project and provided additional support such as proofreading, practice sessions, etc.

Abstract

The purpose of this project is to design and build an automated pill sorter. The pills will be sorted into a typical seven-day pill planner with AM and PM containers. Additionally, the project will take advantage of a cutter system to sort pills by half dose as specified by the user and store the other half for later use. The project will also allow user input from an onboard and computer-based user interface regarding pill medication, dosage, and regimen. These actions will be controlled by a NORDIC Semiconductor microcontroller which will retain pill data, operate the user interface, and control when doses are sorted via a master and minion system with two Arduino microcontrollers which will control the sorting and cutting systems, respectively. The project is also designed with user modularity in mind, allowing the user to sort as few as one type of pill or as many as six types. Finally, the cutter system being used has already been designed by a previous team. This cutter system was designed to cut pills in half with a minimum amount of mass loss. There were performance issues with the cutter and dispensing systems. However, both systems ultimately performed their desired functions, and the project achieved all goals set for itself. The contents of this report will detail the development and design of the project and its results.

Table of Contents

List of Figures

List of Tables

List of Equations

1. Introduction

Managing prescribed medication is a common fact of life for many Americans. About 60% of Americans take some form of prescription medication and at least 25% take at least 4 different prescriptions with most being over the age of 50. [1][2][6] Pills are among the most common type of prescription medication. In order to make sure they take their prescriptions correctly and on time, many employ the use of a pill planner which they can use to sort their prescriptions for the week. Additionally, some prescriptions can be costly, costing in the hundreds for a month's worth. In order to save money, many buy their prescriptions at a higher dose (usually double their prescription) in bulk. They then cut the pills to their prescribed dose. Cutting and then sorting their prescriptions can be a time-consuming process and difficult process for those over the age of 50 with difficulty increasing with age due to age related problems such as physical disabilities, deteriorating motor skills, and mental degradation. These issues also have the dangerous potential to enable over or underdosing of their prescriptions.

The objective of this project is to design a device that can automatically sort prescription pills into an ordinary 7-day, night and day pill planner. Additionally, a cutting function that can cut pills in half, when necessary, must also be included. The project serves to save time for the user by automatically sorting and cutting their pill prescriptions as well as help prevent any potential mistakes in dosing. The deliverables for this project will include a working prototype of the machine as a proof of concept. Additionally, the scope of the project is limited to just the sorting mechanism and incorporation of the cutting system. The stake holders for this project are the University of Southern Indiana and the project's customer: Dr. Julian L. Davis. This report includes background market research, concept selection, its detailed design implementation, and testing results.

2. Background

2.1 Products on the Market

Figure 1: Commercial Automated Pill Dispensers (From left to right: Philips, pria, MedaCube)[3]

There are only a handful of automated pill dispensers on the consumer market. From research into each of these products, several conclusions about their designs were found. Firstly, all dispensers are capable of sorting and dispensing a large variety of pills with the largest amount being up to 16 different pills. The dispensers only sort and dispense pills for very specific times as defined by the user. Additionally, all dispensers include some form of E-Health integration with the ability to notify caregivers and physicians. Most dispensers include a rotational and radial storage system for storing and sorting pills. All dispensers are within a cubic foot with the largest being the MediCube with a volume of 13 cubic inches. Finally, none of the dispensers on the market have the ability to cut pills into half doses if necessary. These conclusions were taken into account when designing the automated sorting system. The price range for these dispensers range from $300 USD to $1200 USD[3].

2.2 Cutter Design

The design of the project builds upon that of a past team, which was merely a pill cutter. The past team's project is shown in the Figure 2 below. After reviewing the design, we found both positives and negatives. Positives include a simple, fast, and efficient method of cutting pills and the use of a drum design to grab pills for cutting that is gravity fed. Negatives include an unreliable drum design that does not reliably grab pills and an inability to account for more than one shape or size of pill.

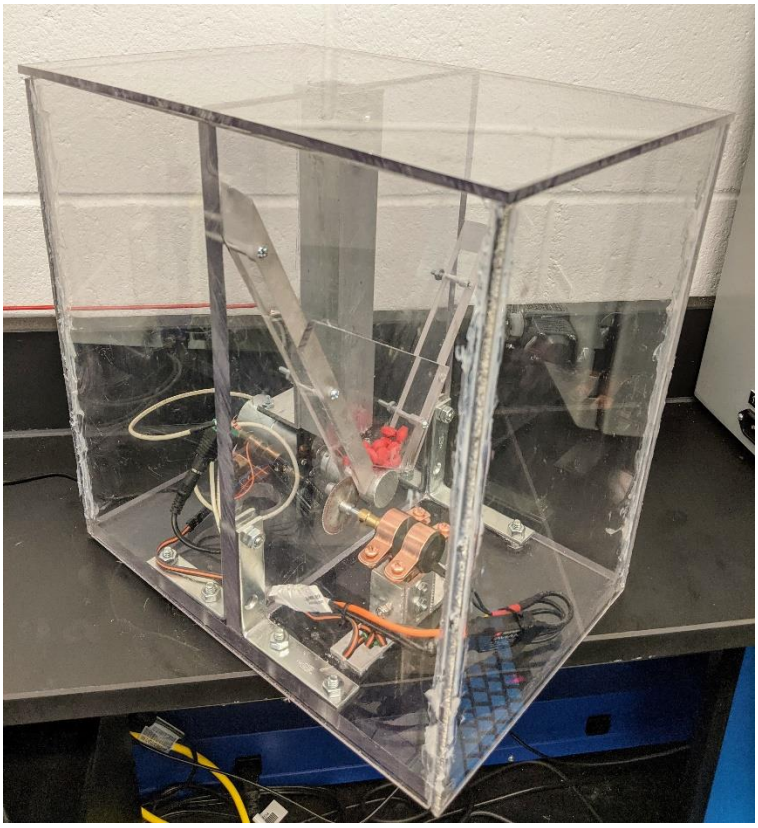


Figure 2: Pill cutter design from previous team.[7]

The system above uses a funnel to deliver pills into a rotating pill hold that rotates to the blade, cutting the pill in half with minimal loss of mass of less than 10%[7][5] as required by FDA regulation. This is done by using a 12V brushed motor to spin a pill holder that collects one pill, where the motor spins a drum clockwise that rotates it towards the cutter. A 6V 3-phase brushless

motor is used to run the blade of the cutter to cut pills in half that the pill holder brings towards it. These motors are controlled via an Arduino.

The past team's project also gave some useful information on the mass loss when the blade cuts the pill in half. Table 1 summarizes the experiments done for measuring mass loss after cutting pills in half. The optimal speed to achieve these results was found to be around 20,000 RPM[7]. This speed was regulated via an electronic speed controller.

Table 1: Mass loss of cutter design.[7]

Trial	Initial Mass	Half 1	Half 2	Variation	Mass Lost
	g	g	g	%	%
1	0.55	0.31	0.21	38.462	5.455
2	0.57	0.18	0.35	64.151	7.018
3	0.58	0.26	0.21	21.277	18.966
4	0.57	0.24	0.21	13.333	21.053
5	0.57	0.26	0.28	7.407	5.263
			Average	28.926	11.551

This design was only designed to cut a certain type of pill. According to their research, extended release and controlled release pills and capsules, coated pills, and critical dosage type pills should not be cut. As such, only compressed tablet pills that do not fall under the listed categories are to be cut.

2.3 Microcontrollers

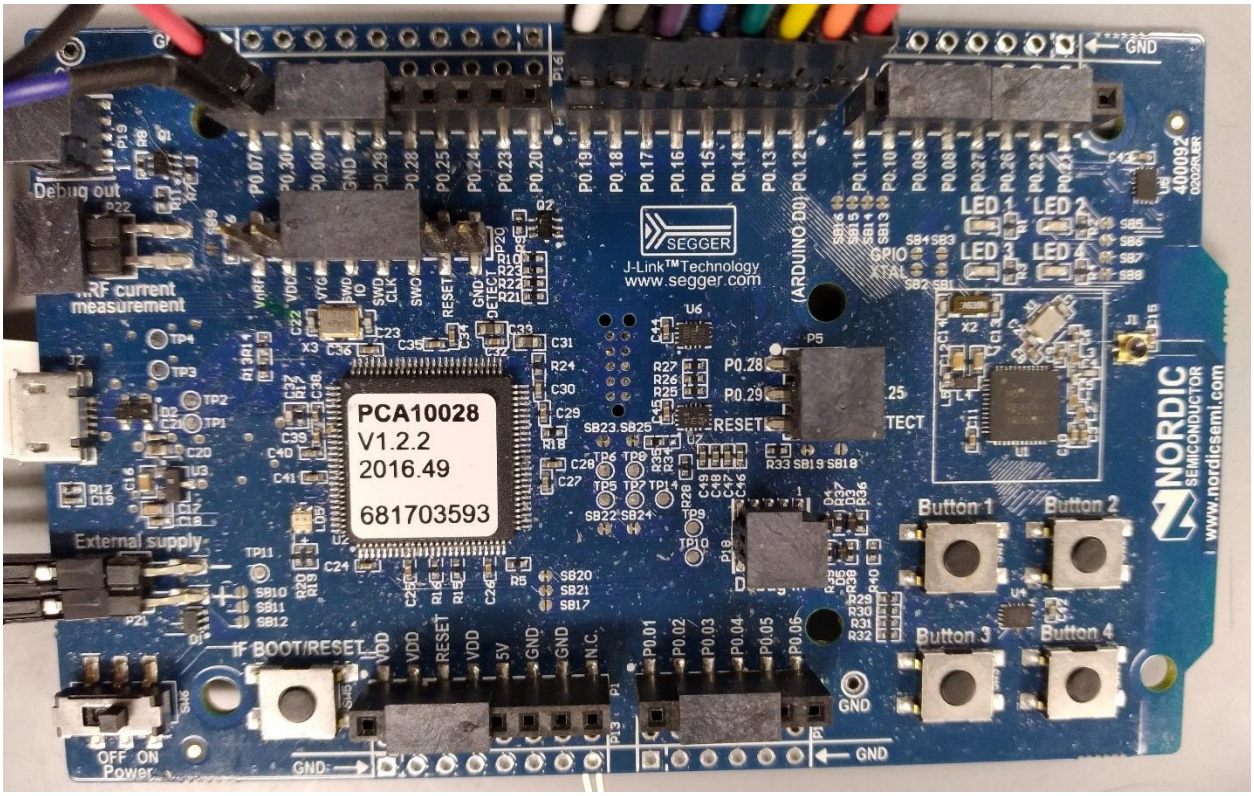


Figure 3: nRF51 Microcontroller

In order to control the motors and facilitate the actual functions of the machine, a microcontroller needed to be selected. Two microcontrollers were selected. The first microcontroller that was selected was the Nordic Semiconductor nRF51 microcontroller as shown in Figure 3. There are several reasons this board was selected over the more popular boards such as Arduino. Firstly, there was an increased familiarity with this board and its capabilities as it was used in a microcontroller class by one of the team members. Additionally, the board comes with a dedicated software development kit which has rudimentary implementation and examples of all its peripherals, free to manipulate and change as needed. Secondly, it has over 31 modular GPIO pins that can each be customized for specific functions, mapped to specific peripherals of the board, or interrupts. Thirdly, the board has over 6 software interrupts. These interrupts were

integral for the design and execution of the control system software. Fourthly, the most important reason for selecting this microcontroller is the board's ability to allow direct manipulation of registers. A register is a unique partition of data in the microcontroller RAM. This access to registers allows for more complex and custom functions to be created. This proved especially important for the implementation of the communication protocol used, I2C or two wire communication. Lastly, the board has dedicated I2C functionality, which is the communication protocol that was chosen to be used.

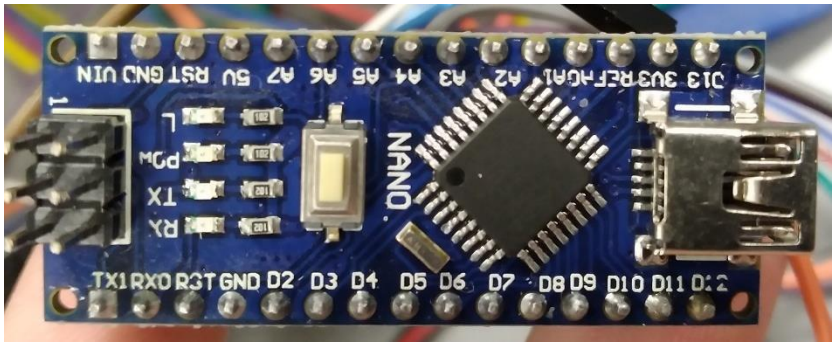


Figure 4: Arduino Nano

The second microcontroller selected was the Arduino Nano as shown in Figure 4. There are several reasons for choosing this microcontroller. Firstly, given that the project has a size constraint, the small size of the Nano allows the project to save space for more important functions like the motors and for creative placement of the microcontroller. Secondly, the Nano is very cheap: costing about 10 USD. This helps cut down on overall construction costs and lowers the cost of replacement should a Nano break. Thirdly, the Nano comes with dedicated I2C functionality, which is the communication protocol of choice for the project. Lastly, Arduino boards are supported by a dedicated integrated development environment and a plethora of community made libraries to use when programming the Nano to control things like servos and stepper motors. This makes development of the control code much simpler.

2.4 TWI/I2C Communication

This project plans to have two different microcontrollers communicating with each other; as such, a communication protocol had to be selected. For this project, it was decided that I2C or two wire, TWI, communication was to be used. There are 2 main reasons for doing so. Firstly, as the name suggests, it allows data to be transferred and received over only two wires. Secondly, these 2 wires can be shared with all respective masters and minions through a bus system. As such, 2 and only 2 wires are necessary for a master/minion system. This is advantageous as it saves pin space for other functions and simplifies wiring. Other communication protocols like serial peripheral interfaces need 4 wires for it to function properly. Additionally, other communication protocols such as serial peripheral interfaces need additional wires for each device connected. In the case for serial peripheral interfaces, if 3 minion devices were used, then at least 7 wires will be needed. A UART protocol, however, can only connect two devices and as such cannot be used in a master/minion setup. However, serial peripheral interfaces do have two advantages: it has a very high speed limit for transmitting and receiving (over 10 Mbps), and it operates at full duplex which allows transmitting and receiving to happen simultaneously. I2C on the other hand only operates at half duplex which only allows one action of transmitting or receiving to happen and has a low speed limit for transmitting and receiving (100Kbps to 400Kbps). However, since the project is only transmitting small amounts of data at a time and operates sequentially, speed and simultaneous operations are not needed. As such, I2C is the best communication protocol for the project.

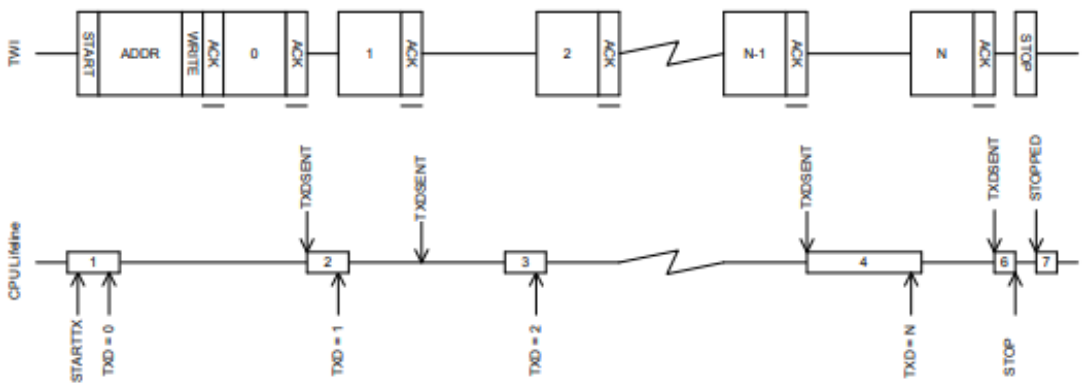


Figure 5: I2C Data Transmission [8]

As shown in Figure 5, the functionality of I2C is simple, one wire is responsible for the data. This is called the data line. Since this protocol is synchronous, the other wire is responsible for synchronizing the minions with the master's clock speed. This is called the clock line. In the case of this project, the subsystems operate at 100Kbps, so that will be the clock speed for the clock line. Additionally, the max number of bits that can be transferred at a time are 8 bits. In order to differentiate the different minions from one another, unique address bits are assigned to each minion. These addresses are defined via hardware or software of the minion. These address bits are 7 bits long. In order to communicate with a minion, the master first sends out a command bit followed by the address bits for the minion it wants to communicate with. Once this 8-bit line of data is received, the minion will send an ACK or acknowledge bit, which lets the master know that it is ready to transmit or receive. From then on, communication between the master and minion is established and other minions cannot be accessed. In the case of transmitting, the master sends the 8-bit line of data bit by bit, from least significant to most significant bit. With each bit sent, the master waits for an ACK bit from the minion. Once the master receives the ACK bit, the next bit of data is sent. This continues until all 8 bits of data are transmitted. From then, the master sends a command bit to end communication with the minion. This frees up the bus for the master to

interact with the other minions as necessary. In the case of receiving, the same process applies except that the minion is the one that transmits the data bit by bit. When designing the communication protocol, the project made use of the already provided TWI examples provided by NORDIC Semiconductor.

2.5 Motors

In order to actuate the necessary functions of this project, a variety of motors were chosen. Stepper motors were chosen to actuate the pill holders. Specifically, the project uses NEMA 17 stepper motors. These are the most common and widely used class of stepper motors for small scale electronics. This standard designates the size of the stepper motor and the angle of displacement for each step. In the case of a NEMA 17 stepper, the steppers are 1.7x1.7 in and have a 1.8 degree step angle[4]. There are several reasons stepper motors were chosen. Firstly, and most importantly, stepper motors are able to perform precise and repeatable movements in an open loop system. This means that the stepper can move to and from a precise location without any outside sensor input like an encoder. This saves space in the machine and on the microcontroller as well as lowering construction costs as no extra sensors are necessary.

The way it accomplishes this is simple. Inside the stepper there are two sets of windings that constitute an electromagnet. The outer windings are called the stator as they are stationary. These windings are attached to small gear like teeth pointed toward the inner set of windings. These inner set of windings are the rotor as it rotates. These windings are attached to small gear like teeth that are pointed towards the stator teeth. These teeth are displaced from each other by a very small degree. In the case of the NEMA 17 stepper, this displacement is 1.8 degrees. When the stator is energized by a current, it induces a magnet field on points of the teeth. Additionally,

this magnetic field induces a current in the rotor, which in turn creates an opposing magnetic field which branches out to the teeth on the rotor. These teeth have opposing poles. So, when a pulse of current is sent to the motor, the rotor rotates to align the polarity of the teeth of the stator and rotor. This is called a step. Each step rotates the rotor 1.8 degrees in the case of the NEMA 17 stepper. So, in order to rotate 360 degrees, 200 steps must be applied to the stepper. With these steps, the precise position and even speed of the stepper can be easily controlled. The direction of the rotation is determined by the polarity of the current. Other motors do not feature these teeth and as such cannot reliably move to precise positions or speeds without an outside sensor in a closed feedback loop. Using the stepper allows the pill holders to have pre-mapped locations for each pill that it needs to rotate to in order to dispense the pill. These stepper motors are rated for 1.5A and 12V.

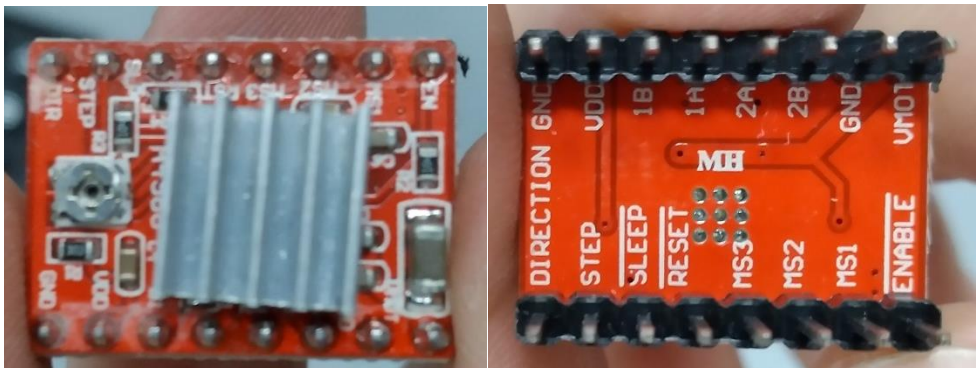


Figure 6: A4988 Stepper Controller (Left to Right: Top and Bottom)

In order to control the stepper motor, a stepper controller was needed. The A4988 was chosen as the controller as shown in Figure 6. This chip was chosen for 2 reasons. Firstly, it has a built in current limiter which throttles the current depending on the position of the adjustable pot. This provides protection to motor in case of power surges. Secondly, it allows for micro-stepping. Micro-stepping is where the steps of a stepper motor are divided into even smaller steps by a certain factor. The A4988 is capable of dividing the steps by up to a factor of 16. However, the project decided to use a factor of 4. As such, the total steps needed for a full rotation was changed

from 200 steps to 800 steps. This allows for more precise control of the stepper. It accomplishes this by only energizing the coils by a fraction of the current needed for a full step. This stepper controller is rated for 5V.

In order to actuate the blade to cut pills in half, a 3-phase brushless motor was chosen. It was chosen because that was the motor that the cutter team chose to achieve the required speed to cut the pills in half efficiently. As such, no other motor was considered for the cutter. This motor is controlled by an electronic speed controller in order to make sure it maintains the required speed to cut the pills as defined by the cutter team. This motor is rated for 1.5A and 12V.

Additionally, the motor to actuate the drum that was chosen was a 12V brushed DC geared motor that is geared for 30 RPM. Again, this was chosen because this was the motor chosen by the pill cutter team to actuate drum and as such, no other motor was considered. However, in order to control the direction of the motor, relays were used in a H-bridge configuration. It is called such because the circuit looks like an H. An additional relay was also added to control power to the motor as shown in Figure 7 below. This motor is rated for 1.5A and 12V.

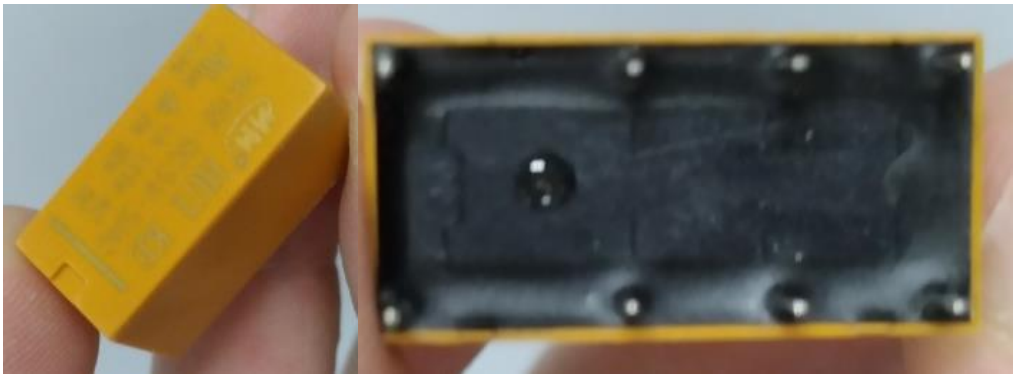


Figure 7: Relay (Left to Right: Top and Bottom)

3. Design

This section details the design process of the project and its subsystems as well as the final designs for all.

3.1 Concepts Generated

We first brainstormed ideas for the funneling and cutting system. Based on the previous team's design, the goal was to make the funneling system smaller, as well as the overall design. Then we moved on to cutting the pill. The first idea was to turn the pill holder right if it needs to stay whole, but turn it left if it needs to be cut by blade. A second idea was to simply cut all the pills and have the sorter/counter add up the correct amount. Another idea was to create a funnel with two holes and a flap that moves a pill to either side based on whether it needs to be cut or not. As a result, we decided to go with turning right if the pill needs to stay whole and turning left if it should be cut. This was the easiest way to achieve our aim.

To sort the pills, a tray was inserted into and taken out of the system. The next step was to determine how the entire system is going to be set up to make the most sense and also to be the easiest to sort. All of the ideas were similar in their set up and ultimately had the same outcome. There were three insertable/removable boxes that cut and sorted pills based on their required amounts. It is then put into the capsule at the bottom of the box with a servo-controlled door. This idea is used for all three boxes. So, when the dose for each pill is at the desired amount then all three servos open the doors and drop the pills into a funnel that leads to a specific time of day in a tray that catches the pills. After that is complete the system resets for the next time of day and continues doing so until the 7 days of the week for both Am and PM are full.

The first design generated focused more on the basics of sorting a whole or cut pills.

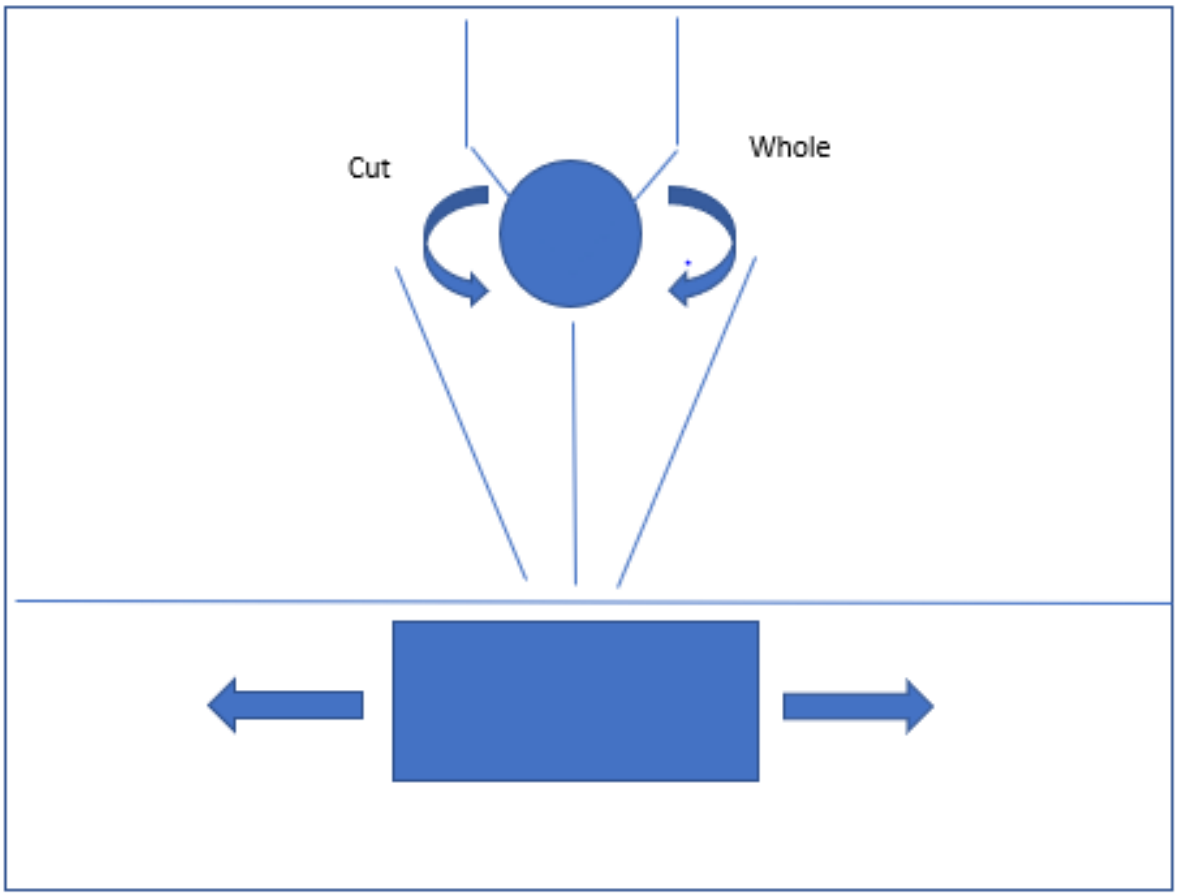


Figure 8: Design Concept 1

Since this was the first design it was very basic and did not account for some of the parts on the scope. First being it can only sort one pill at a time. However, this was a good first concept to put the team in the right direction with focus on the main parts of the project of cutting and sorting.

The draw backs showed that something needed to be done with the cut pills when one half needed to be dispensed and not both halves. And the sorting part doesn't have anything to show how it works or really had any thought behind it. It was just known that it needed to move to sort the pills.

The second concept tried to increase the number of different pills that could go into the system.

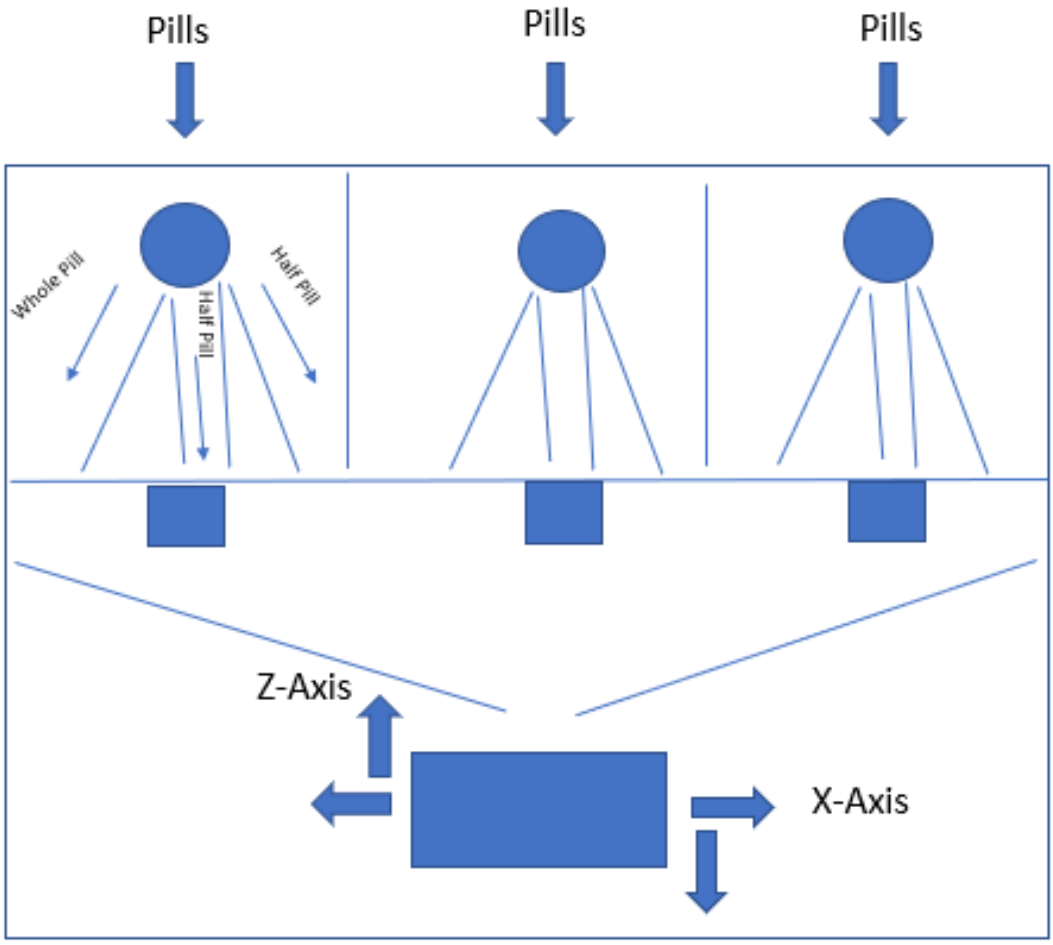


Figure 9: Design Concept 2

The second concept helped increase the number of different pills that can be inputted into the system which each pill section being interchangeable. This means that the customer could take that little box out on the top and replace it with another box for a different pill. And the idea of how to move the pill planner at the bottom to catch the pill was brainstormed on how to move it with no final idea put in place, but it got the team thinking about how to do it.

The problem with this design was the cost effectiveness and the connections. With how every box would be made, each one would have to have a cutter in it and that didn't seem like it was worth the time and money to implement it. Then the problem on how to connect the individual boxes to

the whole system came into play. The electrical connects and the physical ones would cause more problems over time making it harder to operate at 100%.

The final design that the team decided on was the third design concept.

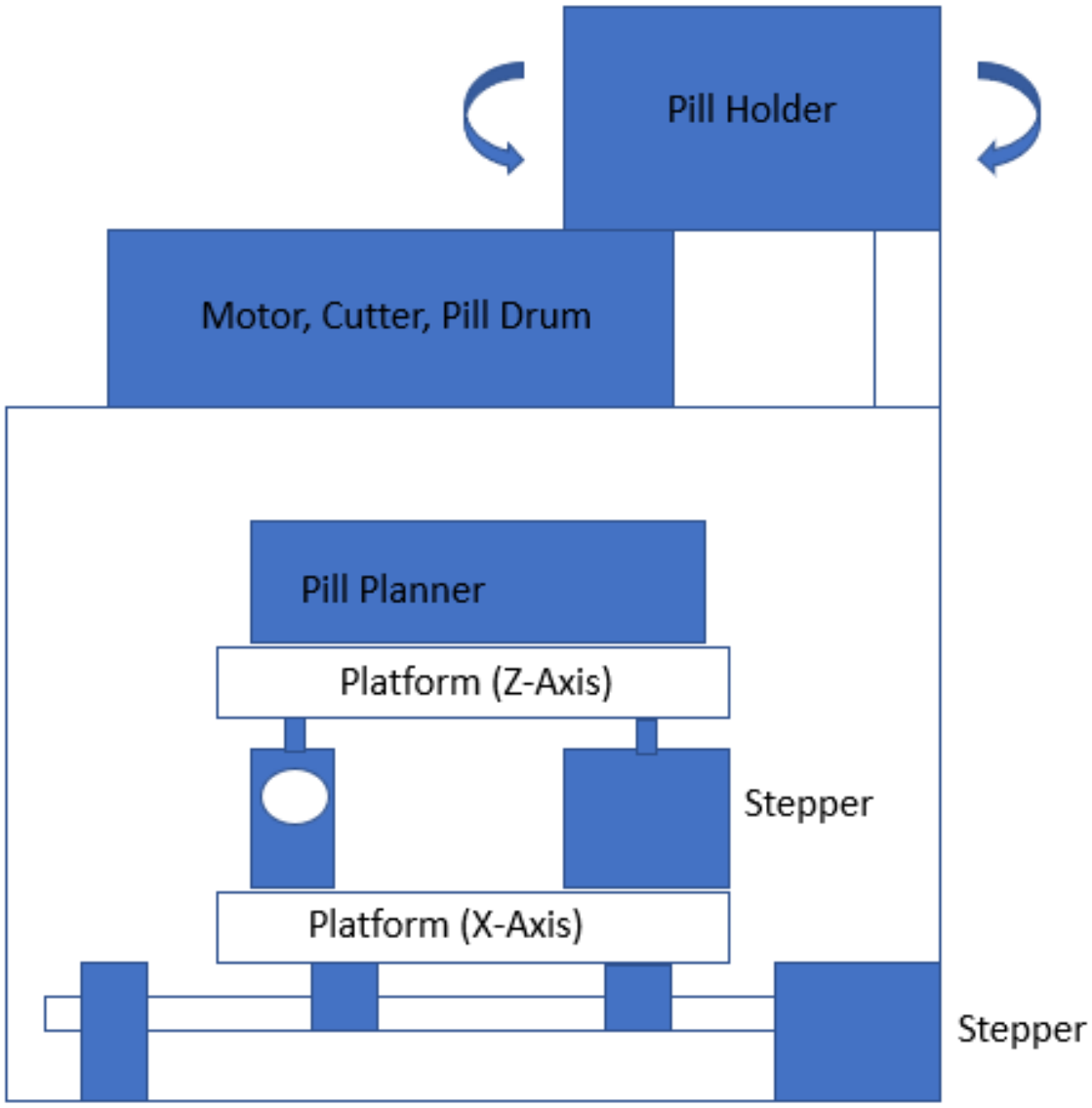


Figure 10: Design Concept 3

The third design helped fix the problems in the first two designs such as the number of different pills that could be put into the system. The pill holder was made to have a different number of slots in it for the number of pills. Then the cutter is used for all pills making it more cost effective

than the second design. As well as the sorting part being figured out. Two stepper motors being used for a different axis to move the pill planner for the specific time of day for the pills to be sorted into.

The first draw back is that this system will be made for a specific pill planner and not everyone's pill planner will fit or work with this system.

3.2 Design Scope

From market research, previous team considerations, and objectives defined by the customer, several requirements were defined to narrow down the scope of the project.

1. Shall automatically sort pills into a standard sized 7-day, AM and PM pill planner.
2. Shall cut pills in half, store, and dispense when necessary.
3. Only compressed tablet type pills shall be cut and sorted.
4. Shall sort up to 6 different shapes and sizes of pills.
5. Shall allow for user modularity.
6. Design shall be around 1 cubic foot.

It was decided that the machine would be designed to sort into standard sized 7-day pill planer that has AM and PM slots. There are two reasons for doing so. Firstly, most people who manually sort their pills already use these pill planners, with the 7-day variant being the most common. This allows for those that still use manual pill planners to still use their pill planners to take their pills when they need to. Additionally, this also allows for the potential addition being able to sort for specific times instead like those on the consumer market. The customer made it a requirement that the machine be able to cut pills in half. As such, this project will incorporate the previous team's cutter design. The only modifications made to the design will be the feeding mechanism and the drum. Since the project is using the previous team's cutter design, only compressed tablet type

pills will be used as defined by the previous team's design requirements. The customer also made it a requirement that the machine must cut and sort at least 6 different shapes and sizes of pills. As such, it was decided that 6 pills would by the upper limit for the design in order to simplify design. Given that many users have many different pill prescriptions of varying shapes and sizes, it was decided that a degree of modularity should be included. That way, the user can customize the machine for their own unique needs and preferences. Since this machine will be designed for use by a regular layman, it was decided that the device must be designed in a way that is comfortable for a customer. Users will more than likely put this in their common area or kitchens with other household appliances and as such, these appliances and other consumer pill dispensers were used to define size regulations for the design. From market research, it was determined that the design must be around 1 cubic foot.

3.3 Design Overview

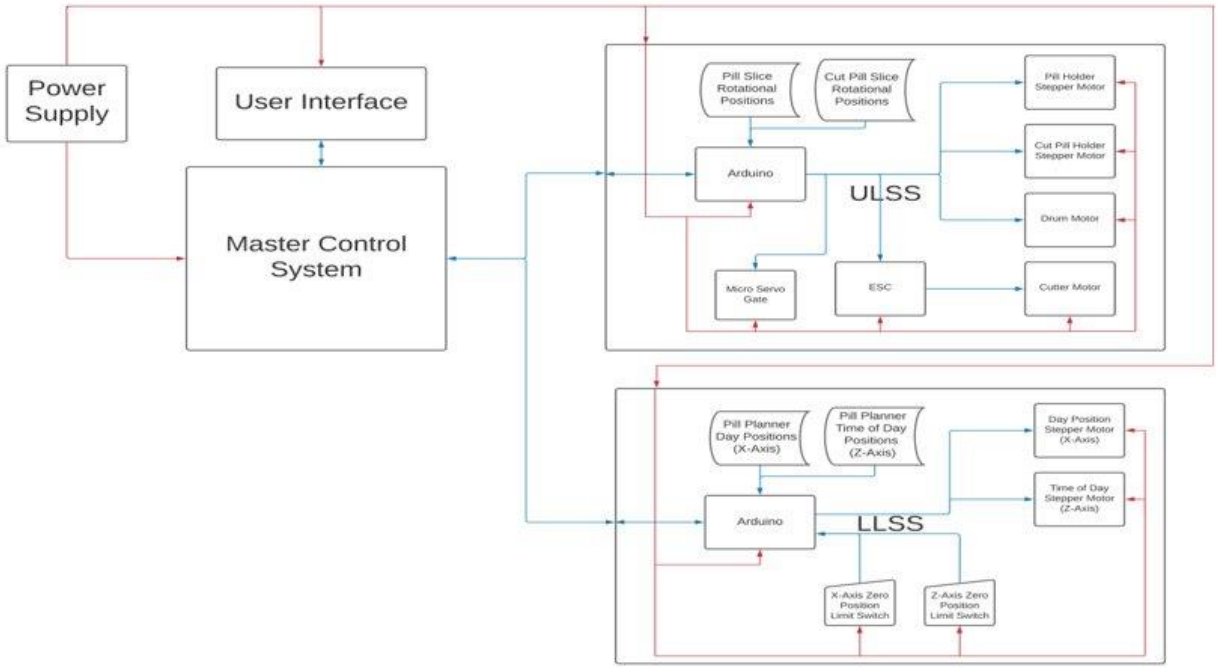


Figure 11: Design Overview

As seen in Figure 11, the design of project is broken up into 4 sub-systems: the master control system, user interface, upper level sorting system or ULSS, and lower sorting system or LLSS. The master control system controls the other 3 sub-systems in a master/minion set up. It is also responsible storing data about pill doses and regimes, and transferring this data to the ULSS and LLSS as needed in order to perform the machine's sorting function. The user interface allows a user to interact with the machine via being able to input pill data and starting the sorting process. The ULSS is responsible for storing and dispensing pills when needed. It is also responsible for cutting pills in half and storing one of the halves for later use as necessary. Lastly, the LLSS is responsible for positioning the pill planner so that its respective compartments receive the dose of pills that is required. The system hierarchy is shown in Figure 12 below.

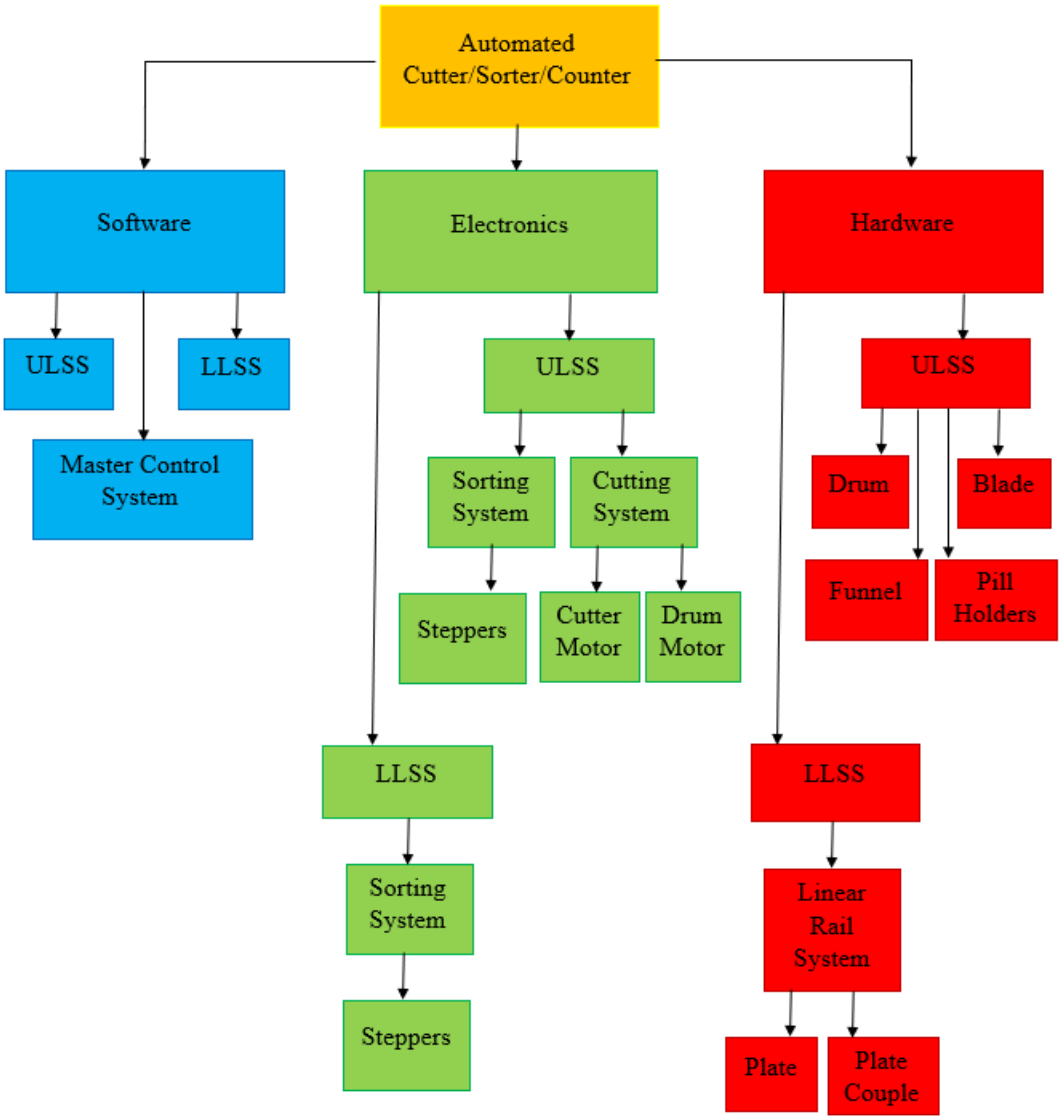


Figure 12: System Hierarchy

Normal operation of the system is as follows. Once the machine is powered up, the user interface will turn on and prompt the user to either input pill data or start the sorting process. If data input is selected, then the user interface will guide the user through all the necessary information needed to properly sort their prescription pills. If the start sorting is selected, then the master control unit will then start the sorting process. It will first tell the LLSS to position the pill planner to the first position so that it can sort the dosages for that particular compartment or time

of day. For a typical pill planner, the first compartment that will be sorted will be the Sunday AM one. Once the compartment is in place, the LLSS will notify the master control unit. Then the master control unit will transmit the dosage data for Sunday AM to the ULSS. This data includes which pills are being sorted, the dosage, and whether a half-dosage is needed or not. Once the ULSS receives this data, it will parse it into the necessary information needed to begin dispensing pills. It will first align the pill holder compartment for the first pill being sorted with the drum that grabs pills. Once aligned it will then actuate the drum: rotating it counterclockwise where it will brag one pill from the holder and then deposit down a funnel to the pill planner compartment. This will continue until all doses are dispensed. Then it will decide if a pill needs to be cut or not. If not, then it moves on to the next pill. If one does, then the cutter motor will be activated, and the drum will rotate clockwise and grab one pill, bringing it to the cutter to be cut. One half of the pill falls down to the funnel while the other half is deposited into a smaller holder for future use. If there is already a half dose available, that dose will be dispensed, and no other pill will be cut. This process will repeat for all pills that are being sorted for the pill planner compartment. Once dispensing and cutting are done, the ULSS will notify the master control unit. From then on, the entire process continues until all 14 compartments of the pill planner have been sorted. Once complete, the LLSS moves the pill planner into a position for retrieval from the user.

A detailed explanation of each of the subsystems and their processes are included in the following sections.

3.4 Master Control System

The master control system is responsible for controlling and coordinating the other subsystems of the machine. The master control system is also responsible for storing and

transmitting the necessary pill data such as dosage and their respective regime or when the pill needs to be taken. The nRF51 was selected as the microcontroller for the master control system.

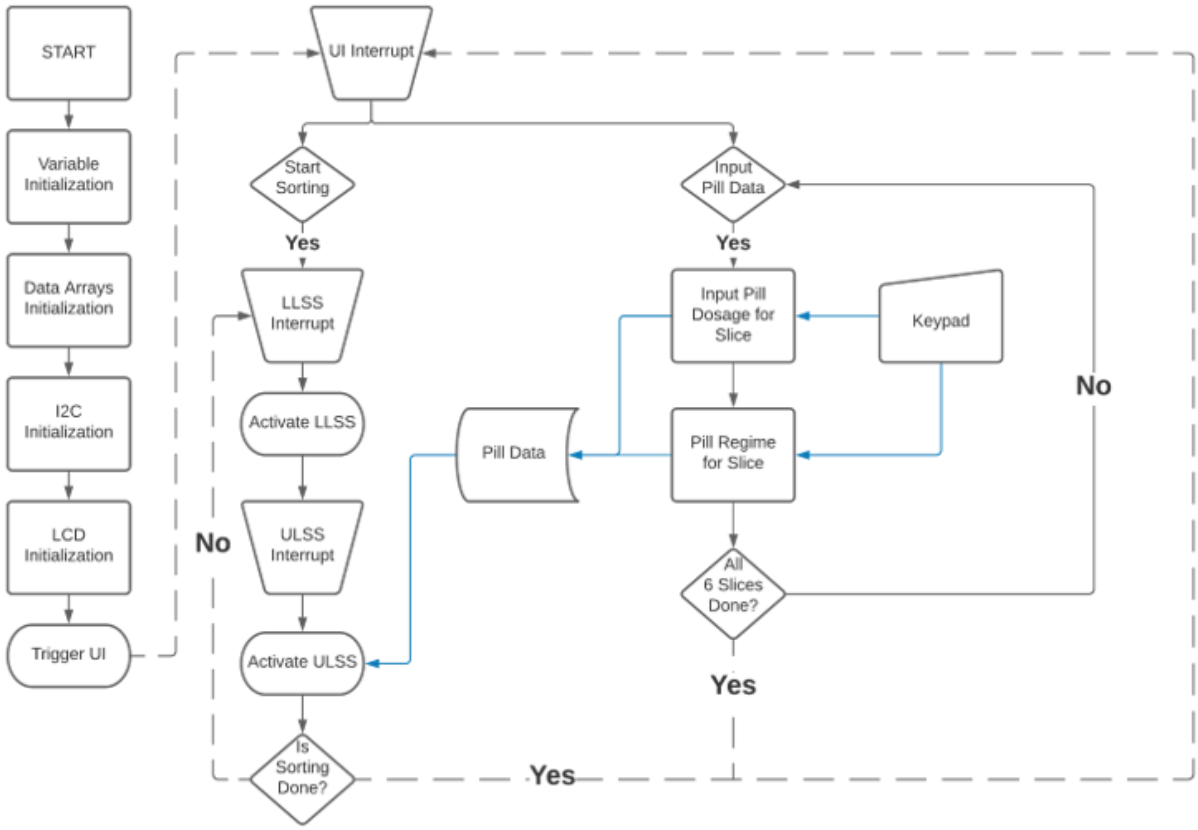


Figure 13: Master Control Unit Block Diagram

Normal operation of the master control system is as follows in Figure 10. Firstly, once power is enabled, the master controller unit goes through initialization of variables and data arrays that will be used for storing pill data. Then the I2C peripheral for the LCD and data lines are initialized as well as the LCD for the user interface. Once initializations are done, the program falls into a continuous loop where the user interface interrupt is triggered. This interrupt is responsible for controlling the display and visual feedback of the user interface as well as determining whether or not to input pill data or start sorting. Additionally, the I2C peripheral for

the data line is disabled while the I2C for the LCD is enabled. This is done because both peripherals share the same I2C bus line and pins, so they both cannot function at the same time.

If input pill data is selected, then the display will prompt the user to input the dosage for 1 pill for the first pill. These inputs are from the keypad with each button being mapped to a 4x4 array. The dosage is in milligrams (mg). The max dosage is 4 significant figures. A for loop is used to determine what the number input is. The number of inputs and their respective numbers are recorded with the first number being input starting at 0 and the last number input ending at 3. Once the user presses the 'enter' button on the keypad, the numbers will then be parsed into their correct dosage. For example, if 3 numbers were input, then the dosage is a number with 3 significant figures. In order to parse this dosage, the first number will be multiplied by 10 to the power of 2. The second number will be multiplied by 10 to the power of 1 and added to the product of the first number. Lastly, the third number will be multiplied by 10 to the power of 0 and added to the sum of the last two numbers. The resulting number is then passed to its respective position in a 1x6 data array with each column corresponding to the dosage of one pill up to six different pills as shown in Table 2 below.

Table 2: Pill dosage for individual slices.

	Slice 1	Slice 2	Slice 3	Slice 4	Slice 5	Slice 6
Dosage (mg)	#	#	#	#	#	#

Once the dosage for 1 pill is input, the display will then prompt the user to input the dosage for the first time of day. In the case of this project, the first time of day will be Sunday AM. The method of inputting numbers is the same as inputting the dosage for one pill. Once the number has been input and parsed, it will be passed to its respective position in a 6x14 data array with the rows

of the array being the individual pills and the columns being the time of day for the dosage starting at Sunday AM and ending at Saturday PM as shown in Table 3 below. This process will continue until all positions for the respective pill row are filled. Once this is complete, the user interface will loop back for the first prompt of inputting pill dosage for one pill for the next pill. The entire process will repeat until all six pills are accounted for.

Table 3: Pill dosage for individual days and time of days.

	Sunday AM	Sunday PM	Monday AM	- - ->	Saturday AM	Saturday PM
Slice 1 (mg)	#	#	#	- - ->	#	#
Slice 2 (mg)	#	#	#	- - ->	#	#
Slice 3 (mg)	#	#	#	- - ->	#	#
Slice 4 (mg)	#	#	#	- - ->	#	#
Slice 5 (mg)	#	#	#	- - ->	#	#
Slice 6 (mg)	#	#	#	- - ->	#	#

Once all 6 pills are accounted for, the user interface loops back to the beginning prompts of starting the sorting process or inputting pill data. If starting the sorting process is selected, then the interrupt for controlling the LLSS will be triggered. The I2C for the LCD is disabled and the I2C for the data line is enabled. In this interrupt, the master control unit will send a command to the LLSS to position the pill planner for the respective compartment to be sorted. Once it receives confirmation from the LLSS that the pill planner is in position, the interrupt ends and the interrupt for controlling the ULSS will be triggered. In this interrupt, the master controller sends the necessary data for the ULSS to sort the pills for pill planner's specific compartment that will be sorted. This compartment is associated with a specific time of day. This data is transferred in an 8-bit line as shown in Table 4 below. The first three least significant bits is the respective pill being sorted. The next four significant bits is the dosage being sorted or number of pills being dispensed for that time of day. In order to parse this dosage into the number of pills being sorted, the

microcontroller takes the dosage for the time of day being sorted and divides it by the dosage of 1 pill. The resulting quotient is the number of pills being dispensed. This number is then added to the 8-bit line. Additionally, a 0 bit is added as the last bit to the line to tell the ULSS not to cut a pill. If the quotient is not a whole number, then the number is rounded down to the nearest whole number which is used as the number of pills being dispensed. Then a 1 bit is added as the last bit to the line to tell the ULSS to cut a pill. Once the 8-bit line is parsed, the master control unit transmits it to the ULSS. This process repeats for the 5 other pills for that time of day. Once all 6 data lines are transmitted, the master waits for the ULSS to send a confirmation that the sorting is done.

Table 4: TWI 8-Bit Data Line

Cut/No Cut	Number of Pills to be Sorted				Slice Number		
Bit	Bit	Bit	Bit	Bit	Bit	Bit	Bit

Once received, the master once again triggers the LLSS interrupt to maneuver the pill planner for the next time of day compartment. Then the ULSS interrupt is triggered and the same process for transmitting the pill data is repeated for the next time of day. This entire process is repeated until all 14 compartments on the pill planner have been sorted. Once complete, the master sends a final completion command to the LLSS to tell it to maneuver the pill planner into a position for user retrieval. For a detailed look at the programming design, please refer to the appendix.

3.5 User Interface

Figure 14: LCD with I2C Module (Left to Right: Top and Bottom)

The user interface is comprised of an LCD screen with an I2C communication adapter and a 4x4 keypad as shown in Figure 14. The LCD is rated for 5V. The LCD functions by inputting a series of 8-bit lines of commands and data to write specific characters to the screen. These data bits are already predefined by the LCD. The master control unit handles writing the individual 8-bit lines to the LCD that produce the necessary characters for visual feedback of the user interface. These commands are parsed by a I2C communication adapter which takes necessary 8-bit commands and translates them into usable functions for the LCD to follow. Since the adapter was made for use with Arduino, the team had to develop their own I2C communication protocol for the module. Since Arduino is an open source project, the developers of the module already had a library freely available. By studying the communication code in the library, which was made for the Arduino IDE, the team was able to reverse engineer and adapt it for use with the nRF51 software. For a detailed look at the programming design, please refer to the appendix.

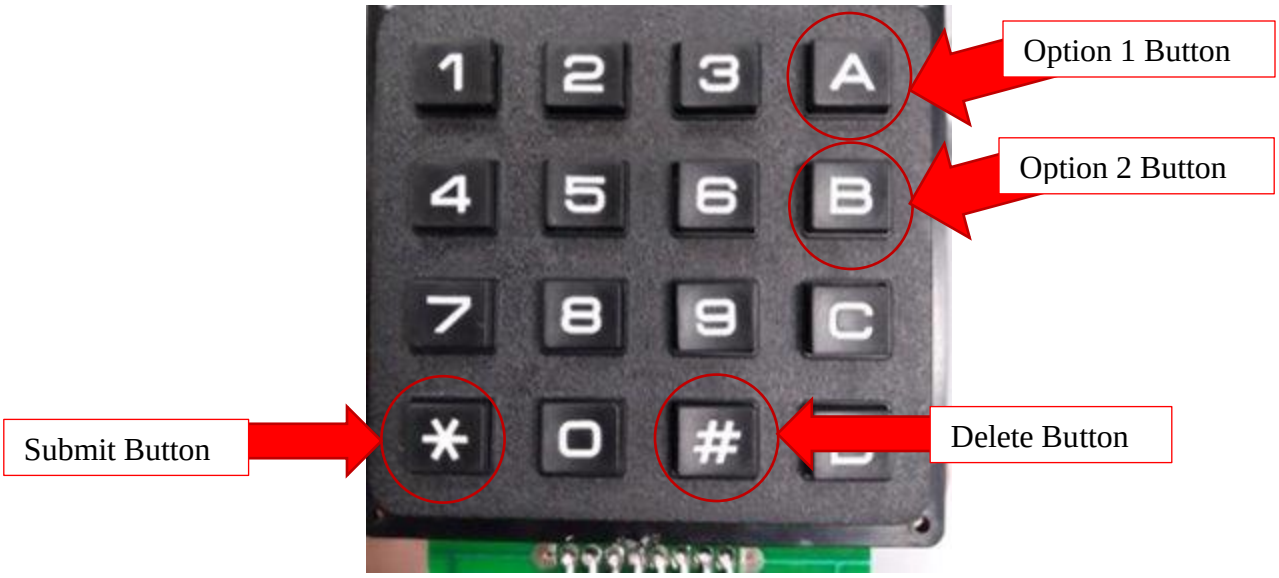


Figure 15: 4 x 4 Keypad

The keypad functions by having all the rows and columns mapped to 4 pins, respectively as shown in Figure 15. The buttons are the cross sections of the rows and columns and are simple normally open buttons so that when a button is pushed, the circuit for that button's respective row and column is complete. In order to detect which button is pressed, each row and column pin are mapped to a 4x4 matrix with the rows being the rows of the keypad and the columns being the columns of the keypad. The pins for the rows start out as output pins and the pins for the columns start out as inputs. When a button is pressed, the circuit for its respective row and column is complete. The master control unit will record which column pin received an input. Then the rows and columns are immediately switched, with the rows becoming inputs and the columns becoming outputs. Then the master control unit records which row pin received the input. From there, the row and column pins are referenced to the 4x4 array to determine which button was pressed. Lastly, the row and column pins are reset to their initial input and output designations. For a detailed look at the programming design, please refer to the appendix.

3.6 Upper Level Sorting System (ULSS)

In order to actuate the motors and control what pills are sorted and how many, a microcontroller is used. It was decided that the best microcontroller to use for this subsystem is a Arduino Nano microcontroller. This microcontroller actuates the motors necessary for sorting and dispensing the pills.

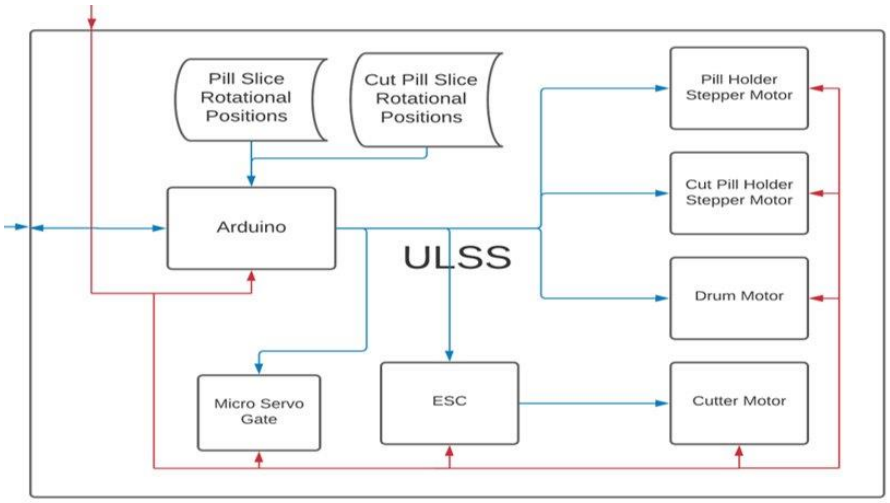


Figure 16: Upper Level Sorting System Overview

The upper level sorting system, or ULSS, is responsible for dispensing and cutting pills as necessary. As seen in Figure 16, the Arduino controls the various motors that actuates the functions of the ULSS using pre-mapped positions. The following sections will explain the ULSS and its functions in detail. Figure 17 below shows the final design of the ULSS.

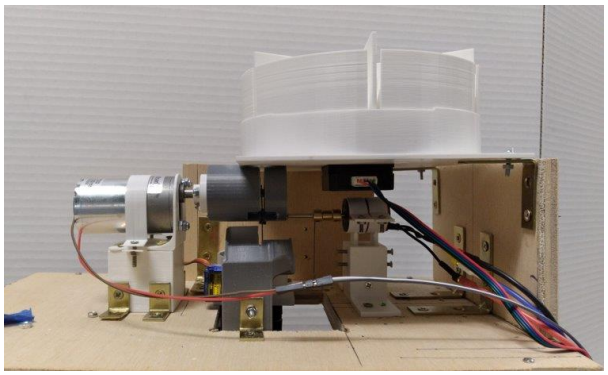


Figure 17: ULSS Final Design

3.6.1 ULSS Programming

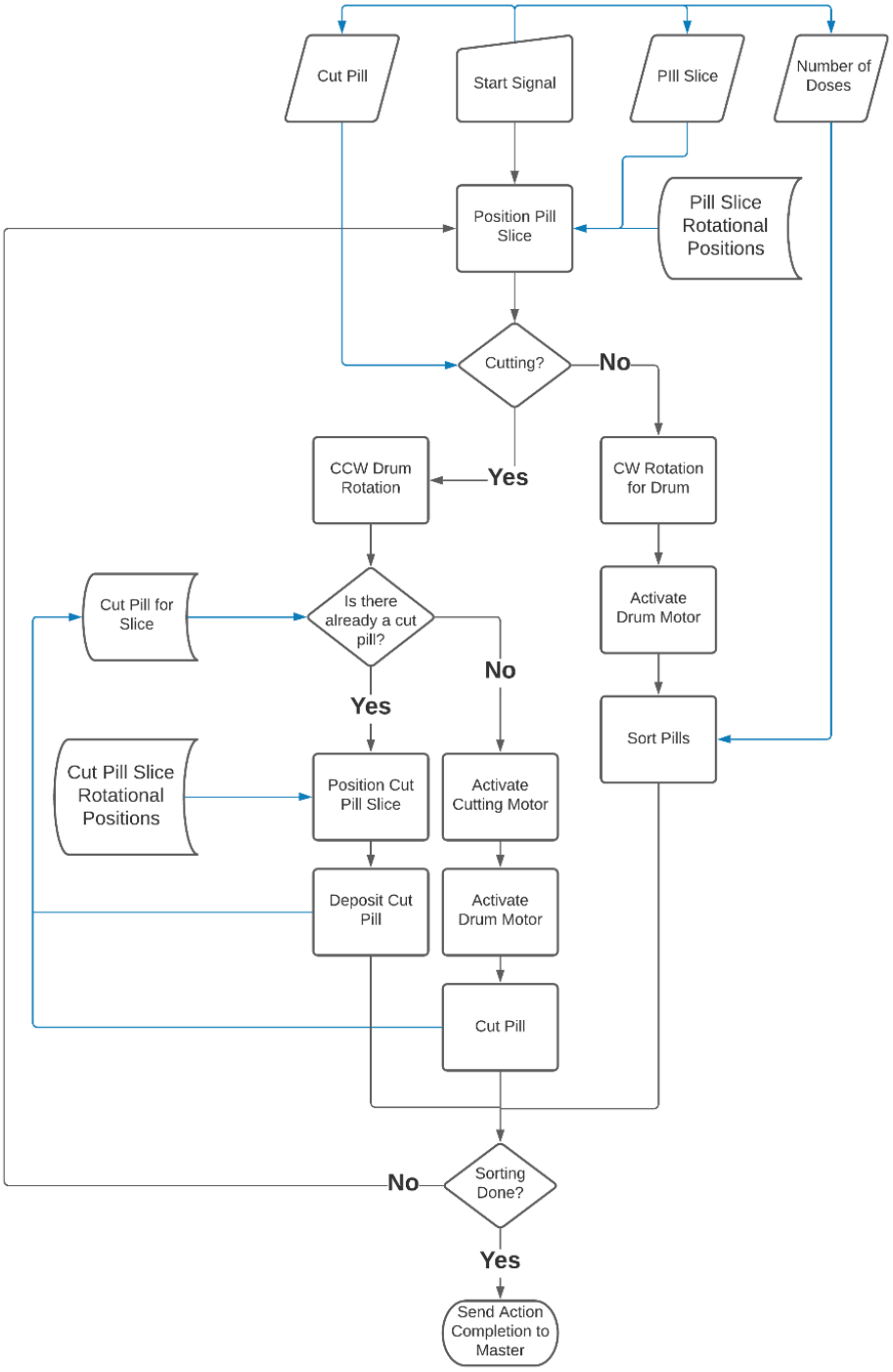


Figure 18: Upper Level Sorting System Program Block Diagram

Normal operation of the ULSS as shown in Figure 18 is as follows. Firstly, all variables and constants are initialized. The stepper motors are disabled in order to prevent noise from the Arduino startup moving them out of step. The cutter motor is then calibrated using the method from the previous cutter team. TWI event functions and the digital address of the Arduino are then initialized. Once calibration is complete, the stepper motors are then reenabled. The Arduino waits for a data input from the master control system via the TWI connection. Once the Arduino receives a data line of pill info, it will parse it into its respective data of pill slice, number pills to be sorted, and if a pill will be cut or not. This will repeat five more times until all six slices for the time of day being sorted are accounted for. The number of pills being sorted and whether to cut a pill or not are stored in separate arrays with their position associated with their respective slice number. Once all six slices are accounted for, the main sorting code will initialize in the main loop of the code. It will start at slice position one. Firstly, it will activate a shaking mechanism where the stepper motor for the main pill holder will rotate back and forth rapidly twenty times. This is to make sure the pills are properly slotted for dispensing and to clear any potential blockages. The number of stepper steps used for the rotations was experimentally found to be 25 steps. For more details on how this value was derived, please refer to testing section of this report. Once the shaking is complete, the dispensing will then commence. A for loop is then called where the end condition is the number pills that must be dispensed. Firstly, the stepper motors are disabled. This is done to save power and to provide the drum motor with enough current to function because the stepper motor each draw too much current from the power bus. Once the steppers are disabled, the drum motor will then be activated to rotate counter-clockwise where it will grab one pill from the main holder and then dispense it. This will continue until all pills have been dispensed. Once all pills have been dispensed, the stepper motors will be reenabled and it will then move on to an if

statement where it will decide whether to cut a pill or not. If no pill is being cut, then nothing will be done. If a pill is being cut, then it will call another if statement where it will determine whether a pill for that slice has already been cut. If no pill has been cut, then the machine will proceed to cutting a pill. Firstly, the stepper motors are disabled to conserve power and the shaking function is called. Once the shaking is finished, the cutter motor is activated where it will gradually increase its speed up to the required 20,000 RPM. Once the cutter motor is up to the required speed, the drum motor is activated to rotate clockwise where it will grab one pill from the main holder and bring it towards the cutter to be cut in half. Once the pill has been cut, a 1 will be added to an array in the position of the current slice recording that a pill has been cut. This value will be used to dispense the other half of the pill that has been cut instead of cutting a new pill. The steppers are also reenabled. If a pill has been recorded to have already been cut for that slice, then the micro servo will be activated to open the gate for current slice which will dispense the cut pill. Once both the dispensing and cutting processes have been completed, the main holder and the half pill holder will both rotate to the next slice counter-clockwise. The main holder rotates 54 degrees or 120 steps while the half pill holder rotates 60 degrees or 133 steps. These values are dependent on the design of the respective pill holders and their angle of displacement between slots of each slice. This process will repeat for each slice until all six slices are accounted for. Once all six slices have been sorted, both pill holders will rotate clockwise back to their starting positions where slice one is aligned with the dispensing drum. The Arduino will then send a confirmation bit to the master control system via TWI to notify it that the sorting is done. For a detailed look at the programming design, please refer to the appendix.

3.6.2 Pill Holder Design

This section will detail the development of the main and cut pill holders and their final designs. The design process for the main pill holder iterative with four different versions including the final version.

Figure 19: Main Pill Holder First Design

The first holder design was experimenting with potential modularity of the slices and position of the slots for the pills. As shown in Figure 16, this design included removable walls. The intention for this was that the user could combine the slices for one large slice if they desired. These walls allowed for a maximum of four slices or four different pills. Additionally, each slice has a slot like hole in the shape of two type of pills: rectangular and circular pills, as these are the most common shapes of pills. However, this design included several flaws. Firstly, with the way the walls were held down to the base, it would make it impractical combine the slices. Additionally, the thickness of the base would not let the pills be slotted correctly; especially with the rectangular pills as they could sometimes get slotted in sideways rather than flat. Lastly, since the slices are flat, there is no way to guarantee that the pills will even be slotted in the holes at all. As such, this design was not accepted.

Figure 20: Main Pill Holder Second Design

As shown in Figure 20, the second holder design gets rid of the removable walls in favor of solid walls attached to the base. Additionally, the slotted holes are moved towards the end of the outer walls. This was done in anticipation of using the drum to retrieve pills for dispensing. Finally, the walls were tapered towards the slots, forming a sort of funnel. This allows for the pills to fall or slide towards the slots to be slotted without any further assistance. However, there is one flaw with this design. For the slices with the rectangular pills, sometimes the pills crowd into the slot where even a shaking movement is not enough to dislodge them. As such, this design was deemed unsuitable.

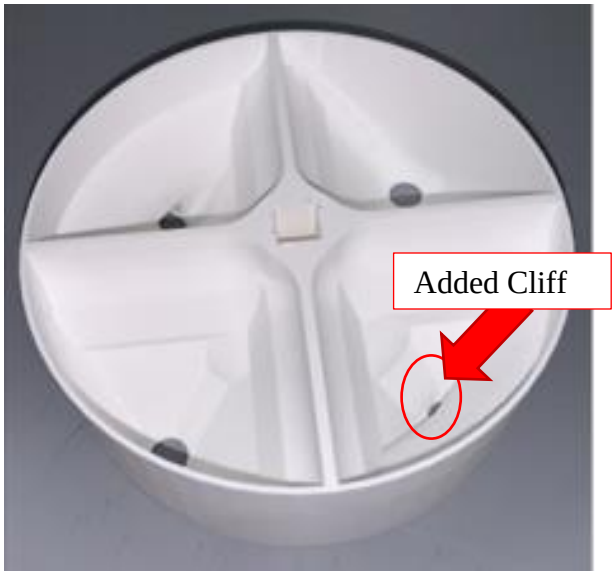


Figure 21: Main Pill Holder Third Design

As shown in Figure 21, the third holder design rectifies the flaw from the seconds design by adding a small cliff over the slot that forms a cave like structure. The entrance to this cave has the same dimensions as the rectangular. This ensures that only one rectangular pill will be slotted at a time. As all problems with slotting the pills correctly have been rectified this design will be used as a template; more specifically, the design of the slices in this design will be used as templates for the individual slices that will make up the fourth and final design.

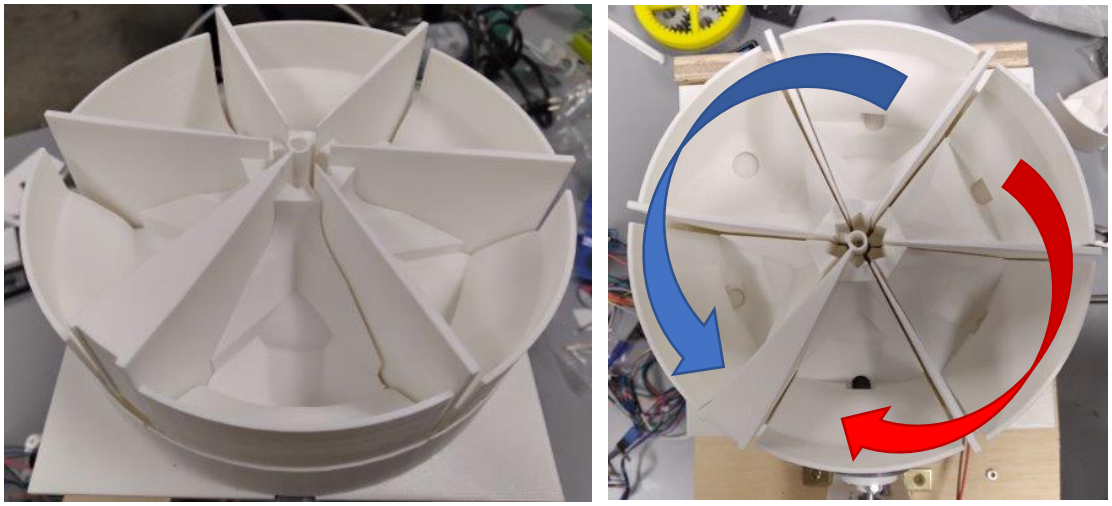


Figure 22: Main Pill Holder Final Design (Left to Right: Isometric and Top Down)

Figure 22 shows the fourth and final design for the main pill holder. This design comes with many changes and improvements. Firstly, there are a maximum of six slices. Two additional slices were added as that is the average number of pills that Americans take. Additionally, this also fulfills the project's design scope requirement 4 which states that this design shall sort up to 6 different shapes and sizes of pills. Secondly, the slices are interchangeable as shown in Figure 23 below. This allows the user to decide what pills are being sorted, how many will be sorted, and in what order they will be sorted. Each slice can be customized to be able to slot a specific shape and size of pill. This also fulfills the project's design scope requirement 5 which states that the design shall allow for user modularity. The design for the slices were taken from the third design, using its slices as templates. Each slice has an angle of 54 degrees. As such, the angular displacement between the slots of the slices are also 54 degrees. This angle was chosen to allow for a whole number of steps (120 steps) to be used for the stepper motor that actuates the holder while also allowing for at least six slices. These slices were taken from the third design, using its slice shapes as templates. Additionally, as shown in Figure 23 below, there is a cut that allows the drum to interface with the slice slots. This is so that sides of the drum where the slot is absent, it acts like

the floor so that the pill holder and slotted pills can move over it safely. This also allows for the slotted pills to be inserted into the drum slot correctly when it rotates to grab a pill. Figure 23 also shows that the stepper motor that actuates the holder is attached directly to the base from below.

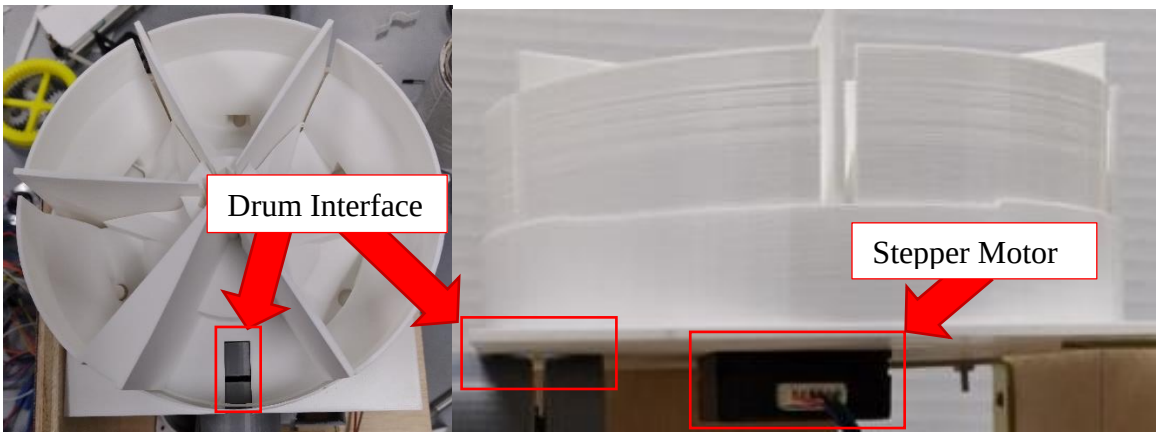


Figure 23: Main Pill Holder (Left to Right: Top and Bottom)

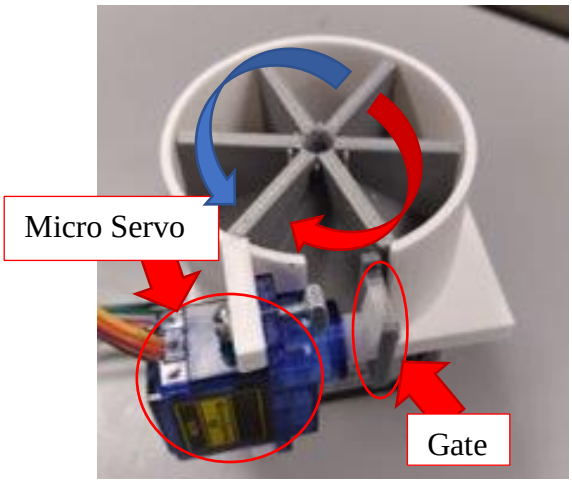


Figure 24: Half Pill Holder Design

This design has six slices to account for the six types of pills that may be cut as shown in Figure 24. A micro servo is attached directly to the base. The arm of the micro servo has a gate attached to it that acts like a wall when closed so that the half pills don't fall out when the holder

rotates. Each slice has a slight taper downward to assist the half pill in being deposited if the gate of the micro servo is lifted open. The stepper motor that actuates the holder is attached directly to the base from below the holder like the main pill holder.

3.6.3 Drum Design

This section will detail the development of the drum and its final design. The design process for the drum was iterative with three different versions including the final version.

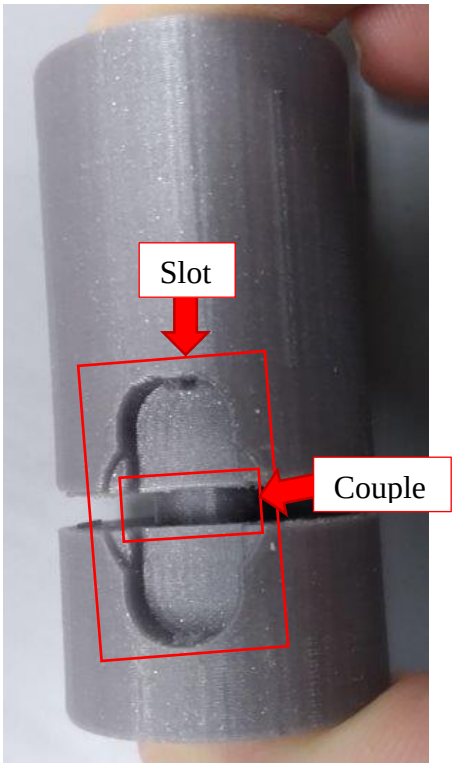


Figure 25: Drum Design One

As shown in Figure 25, this design was experimenting with allowing two different shaped pills: round and rectangular pills. The design also included a slit in between the slot for the pills, making a shorter upper half and longer lower half of the drum. These two halves were coupled by

a 0.5in diameter couple rod in the center. This slit is to allow the blade to cut any pills slotted when the drum rotates towards the blade. The size of the slit was derived from the previous cutter team's drum design and is approximately 1/8in thick. This design succeeded in slotting rectangular pills, but it did not adequately slot round pills as they easily fell out of the drum when rotated. As such, this design was rejected.

Figure 26: Drum Design Two

As shown in Figure 26, the flaw in design one was fixed by adding a circular indent deeper than the rectangular slot. This successfully allowed a circular pill to be slotted in the drum without it falling out while rotating the drum. However, this design only allowed for two different shapes and sizes of pills and as such, was not accepted as the final design. However, the design philosophy of including smaller shapes deeper in the drum was followed.

Figure 27: Drum Final Design

The final design for the drum is shown in Figure 27. As shown in Figure 27, the drum design has many shaped and sized slots with the larger sizes higher up in the drum and the smaller sizes deeper in the drum. The pills needed to be slotted in with their flat side facing up. This design successfully allowed for a large variety of pills (up to six) to be slotted and dispensed. In order to accommodate for the extra sizes and shapes, the diameter of the drum was increased from the previous design by 1.5 times from 2in to 3in. Additionally, the inner coupling rod that connects the two halves of the drum needed to be shrunk as well to a diameter of 0.25in. This did introduce a degree of fragility to the design as this rod was sensitive to large applications of force along the side of the upper drum half, resulting in the coupling rod breaking. As such, extra care had to be taken when mounting the drum to the DC motor. This design helps fulfill the project's design

scope requirement 4 which states that this design shall sort up to 6 different shapes and sizes of pills.

3.6.4 Funnel Design

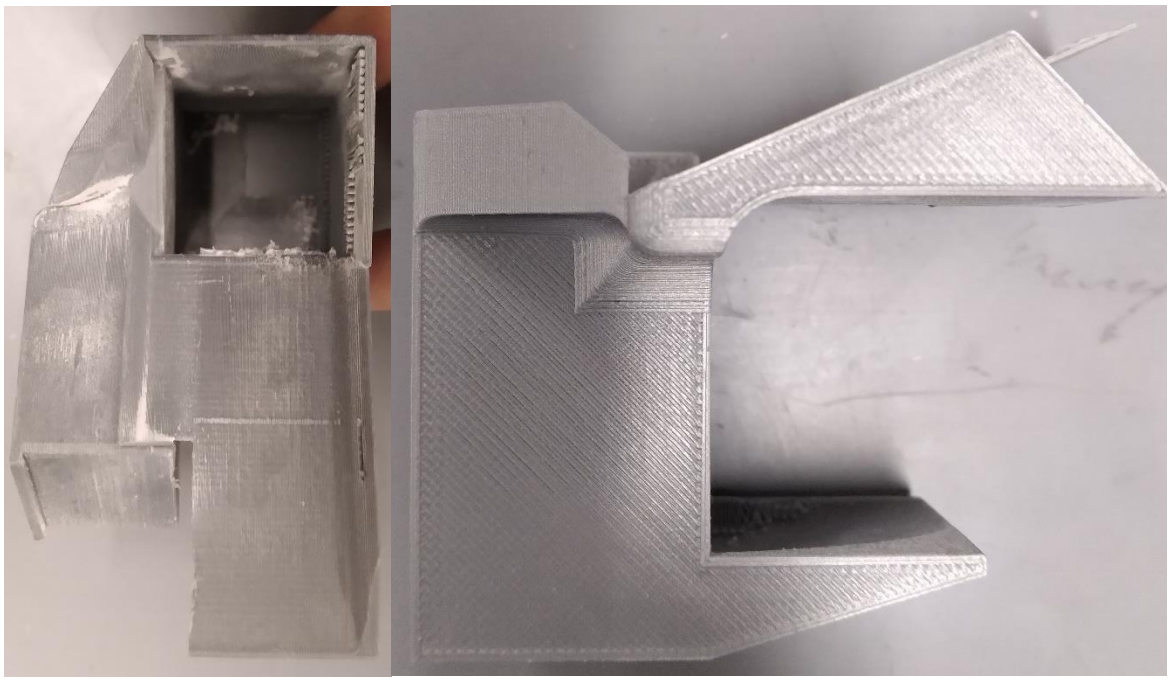


Figure 28: Funnel Design (Left to Right: Top Down and Side)

Figure 28 shows the design for the funnel of the dispensing system. Firstly, the funnel design include two different ramps. The right hand side ramp allows for whole pills that are dispensed to be deposited in the funnel as well as one half (left half) of a cut pill to be deposited for sorting. The left side ramp allows for the other half of the pill to be deposited in the half pill holder for later dispensing. Additionally, there is a smaller ramp near the bottom of the funnel. This allows for the half pills that are dispensed from the half pill holder to slide into the funnel to be properly sorted. The dispensing hole of the funnel is half of the size of the total funnel and has a size of 0.5x0.5in. This was done to minimize the positions that the pills fall into the compartment of the pill holder so that they accurately fall into the compartment being sorted. There is a small

taper in the funnel towards the hole that prevents any pills from getting stuck so that they slide into the hole to be sorted.

3.6.5 Cutter Design

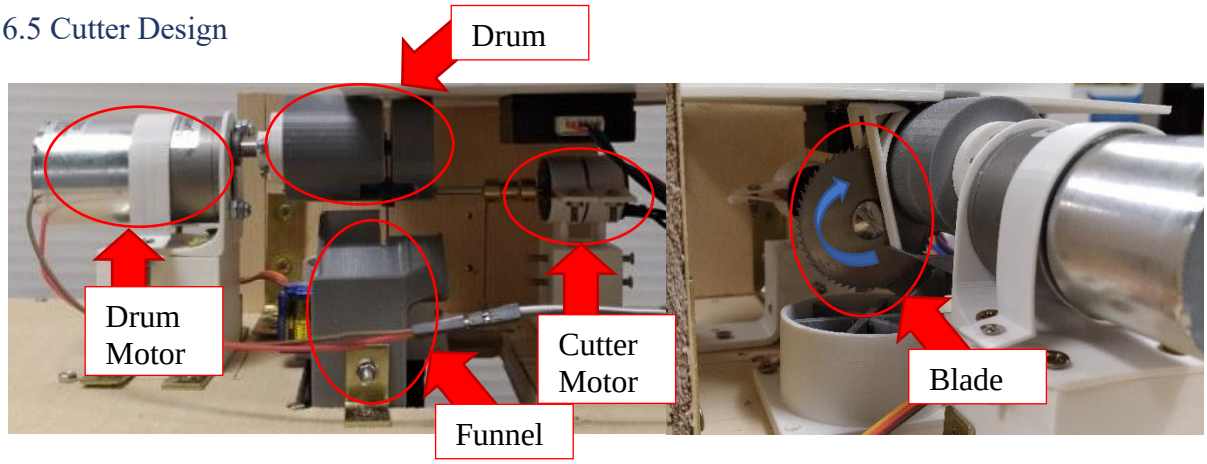


Figure 29: Cutter System (Left to Right: Front and Side)

Figure 29 shows the implemented cutter system. As stated before, the general design of the cutter system was unaltered including positions of the cutter and the drum. The only modification to the system was the drum, which allowed for up to six different shapes and sizes of pills to be dispensed or cut. Figure 26 also shows the orientation and position of the main pill holder, cut pill holder, and funnel in relation to the cutter system. The main pill holder is situated on top of the cutter system where the front end of it rests on the drum where it interfaces with it. The holder is positioned 1.5 in above the center of the drum so that it is flat and level when interfacing with the drum. The cut pill holder is situated below the cutter system so that it can receive the cut half of the pills when the cutter system cuts a pill. There is also a cover that wraps around the cutter side of the drum that is directly attached to the bottom base of the pill holder as seen in Figure 229. This cover ensures that the pill remains in the drum when cut and that the two halves fall onto the correct ramps. Lastly, the funnel is situated below the cutter system but in between the blade. This ensures that when a pill is cut only one half of the pill will be dispensed and the other half will be

stored in the half pill holder. This design succeeded in fulfilling design scope requirement 2 where it states that the design shall cut pills in half, store, and dispense when necessary.

3.7 Lower Level Sorting System (LLSS)

The LLSS is responsible for positioning the pill planner so that the required compartment that is being sorted is aligned with the funnel of the ULSS. It was decided that the best microcontroller to use for this subsystem is an Arduino Nano microcontroller. This microcontroller actuates the motors necessary for positioning the pill planners.

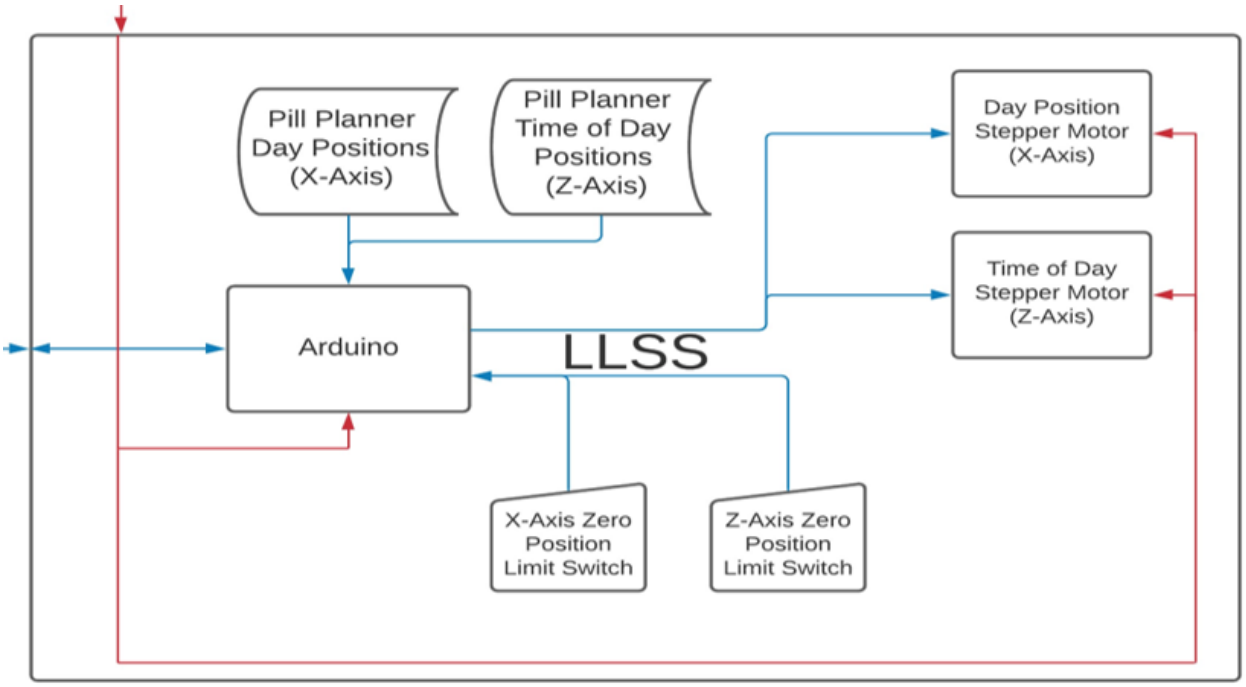


Figure 30: Lower Level Sorting System Overview

The LLSS is responsible for sorting whole or cut pills into the correct compartment (day) of the pill planner. As shown in Figure 30, an Arduino controls the stepper motors that actuate the sorting mechanism of the system. There are two levels, an x and z axis. The positions of the system are pre-mapped to 14 different positions, with 7 positions on the x axis and 2 positions on the z axis.

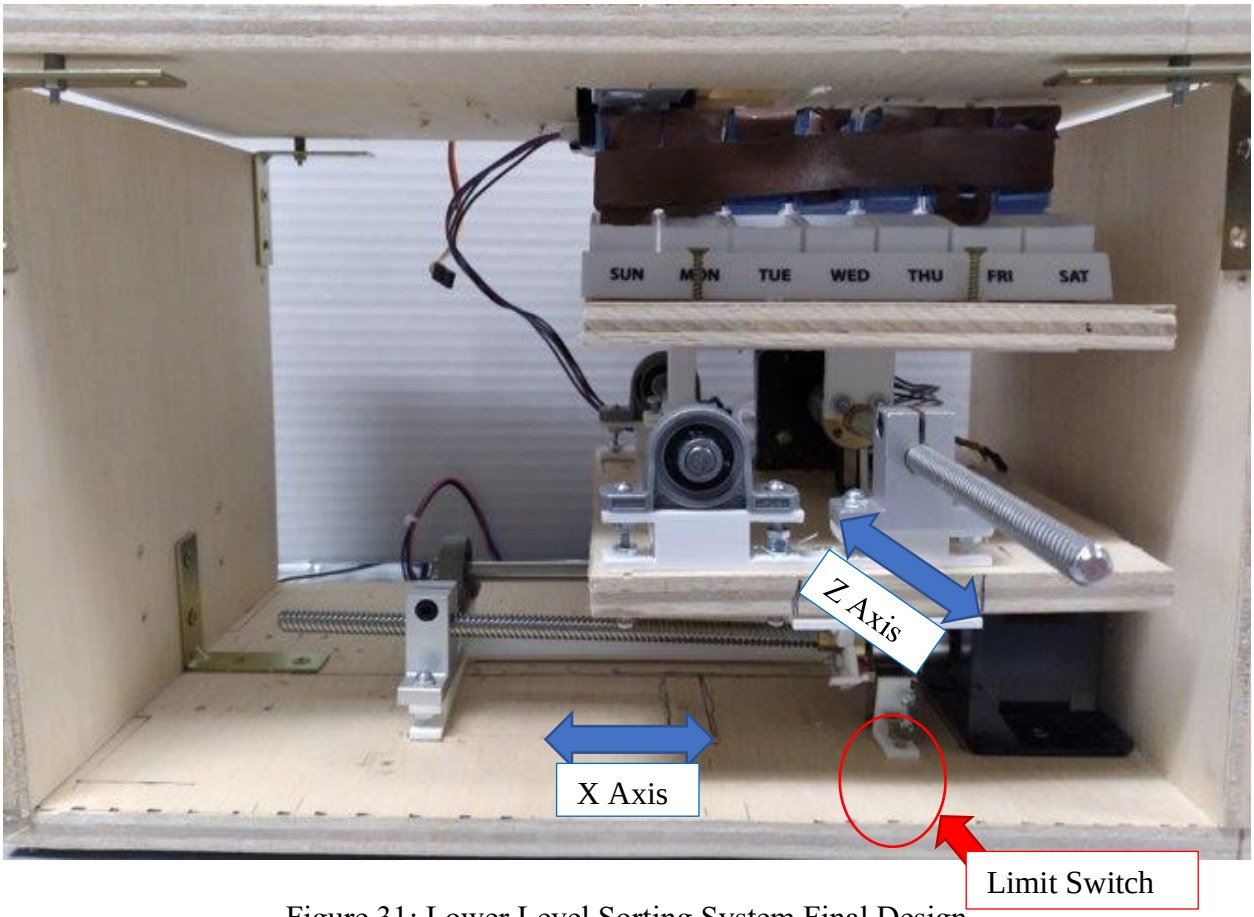


Figure 31: Lower Level Sorting System Final Design

As seen in Figure 31, the design of the lower level sorting system follows the same design as that of a 3D printer. There is a threaded rod that is directly coupled to a stepper motor. The allows for actuation along a straight line. However, one rail is not enough to actuate an object. As such, a second, smooth rod is added to the side of the threaded rod. This smooth rod acts as a guiding rod, keeping the object being actuated straight. Traditionally, 3D printers include two smooth rods to ensure stability as well. However, due to size constraints, only one rod was used. This does introduce a degree of instability in the system when actuated. As such, in order to compensate, the system is actuated a slow speed to reduce shaking and vibration that accompanies it. Additionally, the design makes use of a limit switch. This limit switch is used to calibrate the actuator and define a zero position. When the object hits the limit switch it will send a signal to

the Arduino, telling it that the zero position has been found. This is done so that the actuator always moves the object to the correct position in relation to the zero position. This system is divided into three levels. The lowest level is the x axis with the middle level being the z axis and the top level is where the pill planner is situated. Each level, excluding the top level, actuates a square plate. These plates are connected to the actuators through a coupler that is coupled directly to the threaded and smooth rods. The x axis level actuates the plate holding the z axis actuator. The z axis level actuates the plate holding the pill planner. Together, this allows for the top plate, or pill planner, to be moved in any direction on a 2D plane. The limits of this system are 6 inches on both levels. That is near the total length of the pill planner, allowing for all compartments on the pill planner to have positions pre-mapped to them. There was also an additional problem that was encountered when designing the system. After examining a traditional 3D printer, the distance between the threaded rod and smooth rod was found to be approximately 5 in. A slightly smaller length of 4.5 in was used. However, when the system actuates, the plate couple for the smooth rod would end up catching and getting stuck, which would then stop the entire actuation. Upon further inspection it was found that since the system only had one smooth rod, there was a degree of instability introduced where the plate couple slightly rotated along the y axis of the smooth rod. This rotating movement was enough to produce a torque at that position with the couple on the threaded rod acting as a force on the lever arm of the total couple as shown in Figure 32 below.

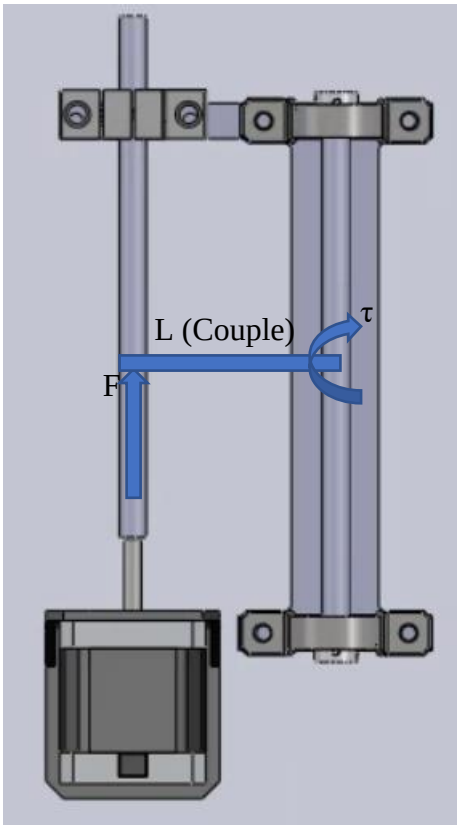


Figure 32: Freebody diagram of rail system.

$$\tau = F * L \tag{3.1}$$

This torque was enough to completely stop the system. Several modifications were made. Firstly, a dry lubricant was added to the couple on the smooth rod, but this was not enough to prevent the torque from forming. Secondly, the gap in the couple for the smooth rod was shrunk so that it was a snug fit. However, this was too much and prevented the couple from moving at all even with a dry lubricant. Lastly, it was decided to shrink the length of the moment arm, or couple, in order to reduce the amount of torque produced by half as can be seen in equation 3.1. The length of the couple was reduced by half to be 2.25 in long. This succeeded in reducing the torque enough that it did not cause the couple on the smooth rod to catch and get stuck and instead just harmlessly slip. This does produce a slight jerk when moving but this is mitigated by slow actuation. With

that problem fixed, this design succeeded in fulfilling design scope requirement 1 where it states that the design shall automatically sort pills into a standard sized 7-day, AM and PM pill planner.

3.7.1 LLSS Programming

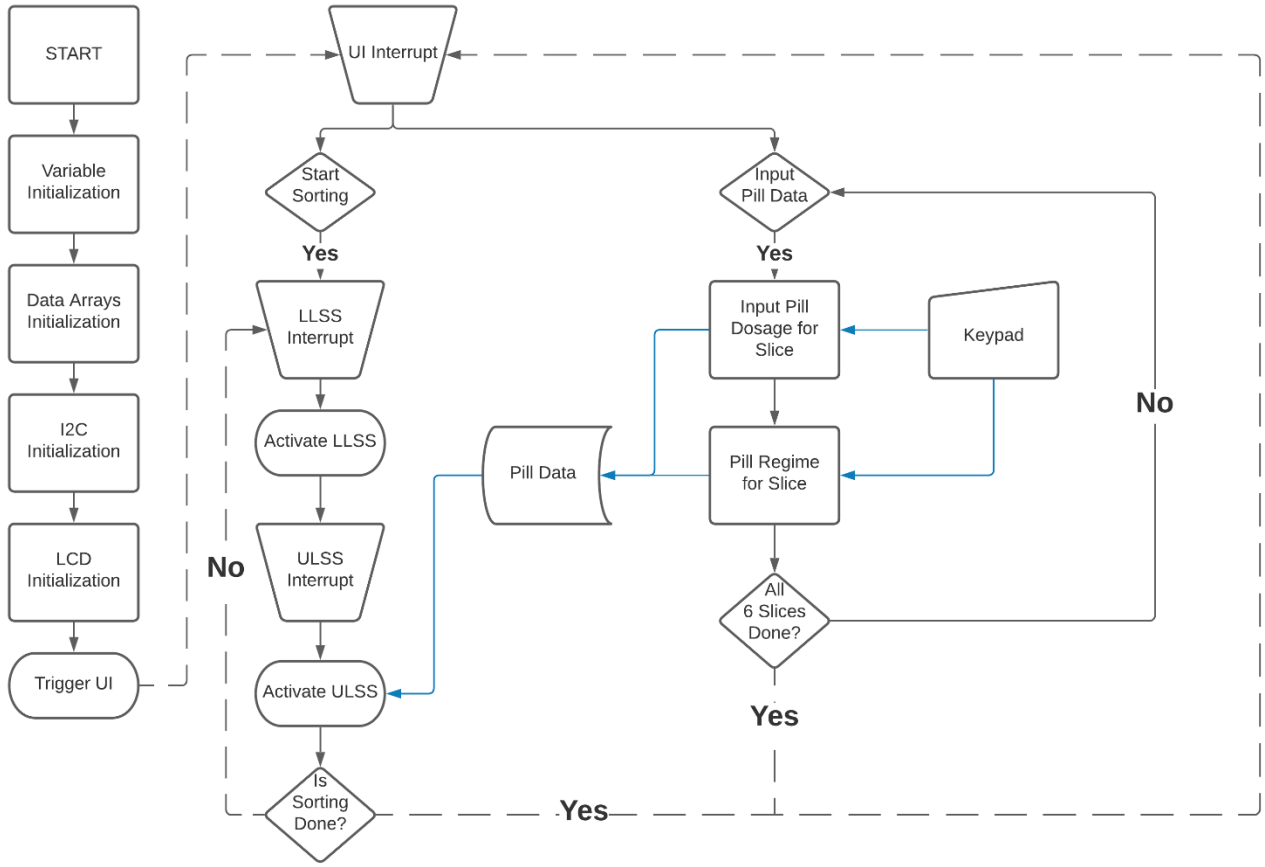


Figure 33: Lower Level Sorting System Program Block Diagram

Normal operation of the LLSS as shown in Figure 33 is as follows. Firstly, all variables and constants are initialized. The stepper motors are disabled in order to prevent noise from the Arduino startup moving them out of step. Once initial set up is finished, the stepper motors are reenabled. Once reenabled, the x axis and z axis call the zero function where the stepper motors rotate counter-clockwise to bring the couples of the plates toward the limit switches. This continues

in a while loop until the limit switches are triggered by the couples. Once the limit switches are triggered, the zero function ends. From here, the Arduino waits for a command signal from the master control system via TWI to move the pill planner into the next position. Once a command signal is received, an if statement is called where it decides whether to move to the first position of Sunday AM or to move to another position. If the first position is called, then nothing will happen. If the first position has already been called, then another if statement will be called. This if statement decides if the pill planner needs to move to the next day or to the next time of day. If the next day is called, then the Arduino will call a function that actuates the x axis to move the pill planner to the next day by rotating clockwise. If the next time of day is called, then a third and final if statement is called. This if statement decides whether the pill planner is in the AM or PM position. If the pill planner is in the AM position, then the Arduino calls a function that actuates the z axis stepper to move the pill planner to the PM position by rotating clockwise and vice versa for the AM position. These functions are only called once for a total of fourteen times and only after the sorting for the previous position has finished with exception for the first position. The Arduino will then send a confirmation bit to the master control system via TWI to notify it that the pill planner is in position. For a detailed look at the programming design, please refer to the appendix.

3.8 Power Supply

The design requires a power supply system capable of providing power to two different voltages: 5V and 12V. The design incorporates a commercial variable power supply that can supply a maximum of 16V and 1.5A. This voltage supply feeds into two sperate power lines for 5V and 12V with ports to power their respective electronics. The 5V supply is used to power the

microcontroller and the other various control electronics. The 12V is used to power the motors in the various systems. In order for the machine to function properly, a voltage of at least 12V needs to be applied to the system but no more than 14V to prevent damage to the electronics and motors.

4. Construction

This section details the construction of the machine and its various electronics.

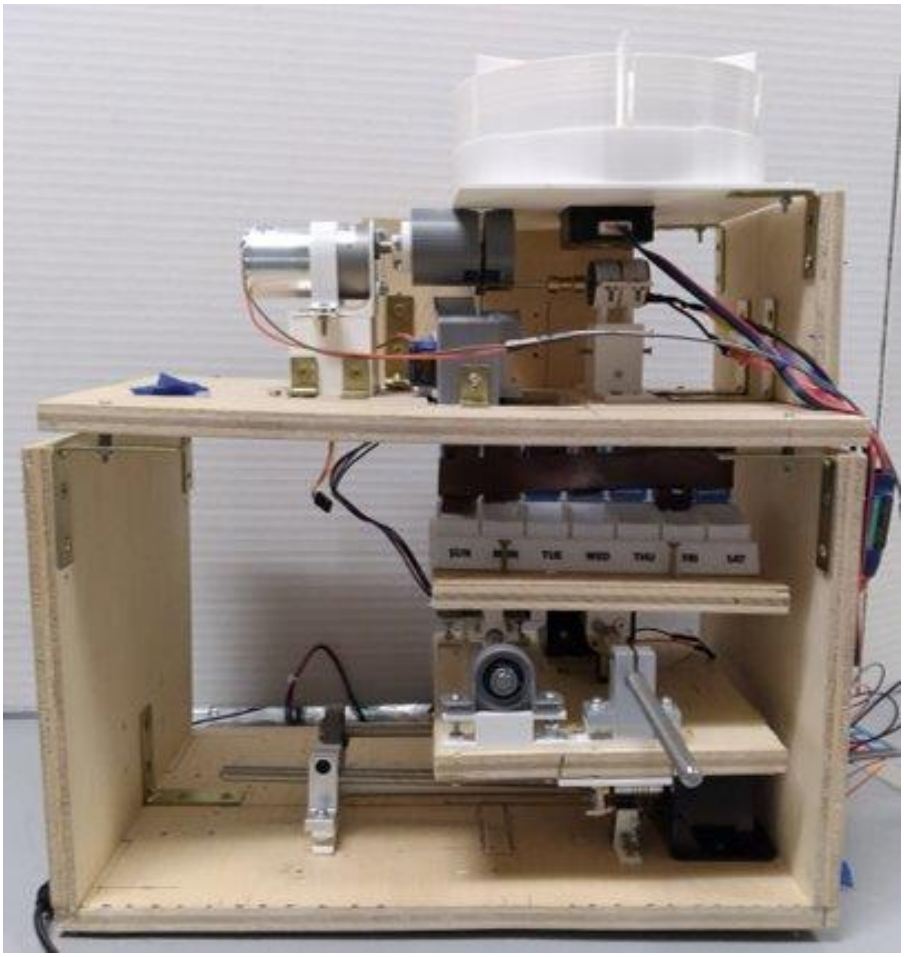


Figure 34: Final Design

As seen in Figure 34, the body of the machine is made of plywood. The thickness of this plywood is 0.5 in. These parts are secured via angle brackets with four 0.5 in wood screws each. The other parts of the machine were secured with M3 screws and nuts. These parts were sourced directly from Home Depot. Most custom parts were printed via a 3D printer with exception to the motors, microcontrollers, electronics, and LLSS 3D printer parts which were all sourced directly from Amazon. All parts are centered on the geometric center of the wooden body frame, both the ULSS and LLSS. The total dimensions of this design are 5.5x8.5x13in. This is within the desired

cubic foot and as such fulfills scope requirement 6 which states that the design shall remain within a cubic foot.

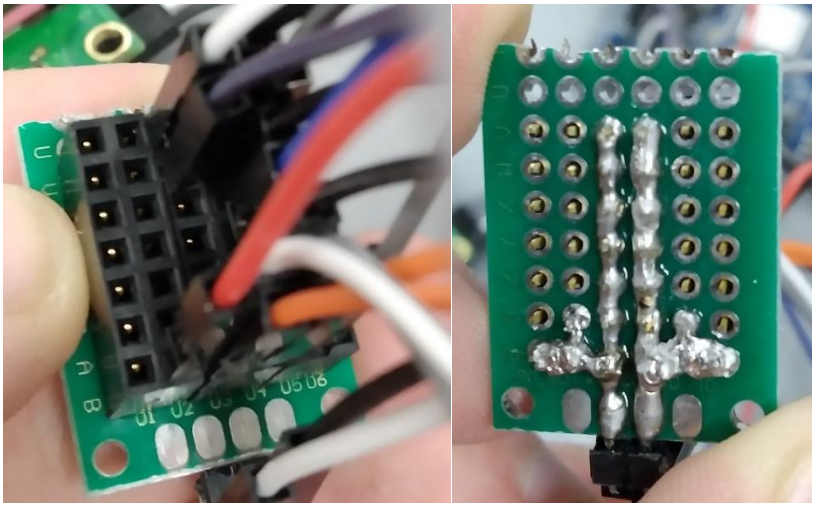


Figure 35: Power Bus (Left to Right: Top and Bottom)

As seen in Figure 35, a protoboard was used to create the bus. As can be seen, the various connections were soldered directly together with solder. Two female input pin connections were used to separate the ground and power. Two male input pin connections were used to allow the power supply to connect to the bus.

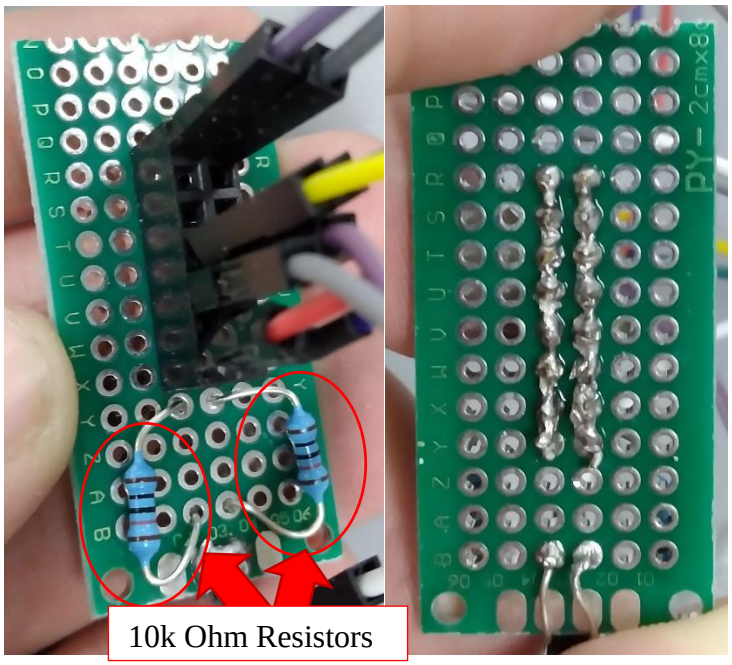


Figure 36: I2C/TWI Bus (Left to Right: Top and Bottom)

As seen in Figure 36, a protoboard was used to create the bus. As can be seen, the various connections were soldered directly together with solder. Two female input pin connections were used to separate the ground and power. Two male input pin connections were used to allow the power supply to connect to the bus. Two 10k Ohm pullup resistors are soldered along both bus lines directly to a 3.3V input pin. This ensures that the logic level of the TWI lines remain at 3.3V.

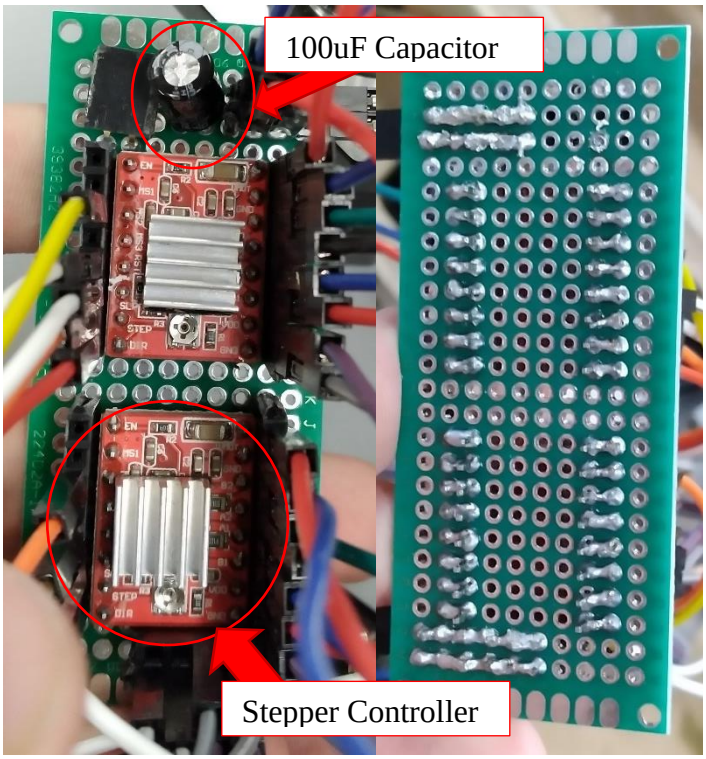


Figure 37: Stepper Control Board (Left to Right: Top and Bottom)

As seen in Figure 37, a protoboard was used to create the control board. As can be seen, the various connections were soldered directly together with solder. Two male input pin connections were used to allow the power supply to connect to the board. Two sets were used to 5V and 12V. These connections each had female input connections to connect the power to the required positions on the board via wires. The stepper control chips were soldered directed to the board with female input pins soldered next to their connection pins to allow for wires to be

connected. Finally, a 100uF capacitor was soldered to the 12V power input pins to protect the chip from potential power surges as directed by the reference manual.

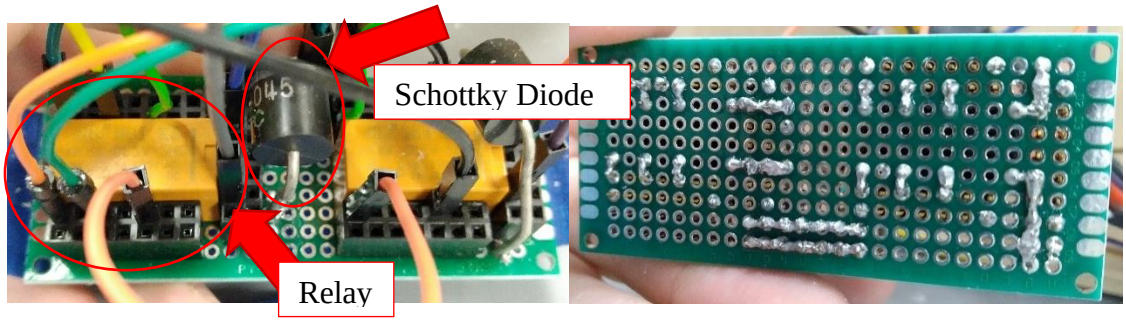


Figure 38: Relay Control Board (Left to Right: Top and Bottom)

As seen in Figure 38, a protoboard was used to create the control board. As can be seen, the various connections were soldered directly together with solder. The relays were soldered directly to the board with female input pins soldered next to their connection pins to allow for wires to be connected. Finally, there is a potential for a voltage to be generated from the coils of the relays that is greater than the voltage input of the microcontroller. Since the relays are controlling a DC motor, when the DC motors are turned off, they will still continue rotate a little due to momentum. During this rotation, they act as a generator and will produce an electrical current. This current could then induce a current in the coils of the relay, generating a voltage that will go directly to the microcontroller pins. Schottky diodes were soldered to the coil input to act as flyback diodes. This is to prevent potential voltage generation exceeding 3.3V from the coils from reaching the microcontroller and damaging it.

5. Testing

This section details the testing of the project design. There was a total of three tests conducted: a shaker test, sorting test, and cutting test. The following sections will detail the testing methodology, results, and discussion of results.

5.1 Shaker Test

The shaker test was conducted to assess the number of steps necessary to displace the pills in the holder and dislodge any potential blockage. However, care had to be taken to make sure that this rapid movement did not cause the stepper to skip a step as that would potentially prevent the drum from grabbing a pill if the slots were displaced from each other. The test was conducted as follows. The steps were incremented by 10 starting at 10 steps up to 50 steps. At each step number, the shaker and pills were observed to see if the pills were excited or displaced and the stepper skipped a step or not. The results are recorded in Table 4 below.

Table 5: Shaker Test Results

Steps	Pill Displacement	Step Skipped
10	No	No
20	No	No
30	Yes	Yes
40	Yes	Yes
50	Yes	Yes
25	Yes	No

As seen in Table 5, steps below 30 do cause the stepper to skip steps, but they also do not excite or displace the pills. Steps above 20 do excite or displace the pills but they also cause the stepper to slip and skip steps due to the momentum of the holder. A final value in between 20 and 30 steps of 25 steps was found to excite or displace the pills but also not cause the stepper to skip

a step. As such, a value 25 steps was decided as the number of steps the shaker function will use to shake the pill holder.

5.2 Sorting Test

The sorting test was conducted to assess whether the correct number of pills would be sorted for a full day. If the sorting system can sort the correct number of pills for the first day of the pill planner, then it can do so for the other 13 days in the pill sorter. The test was conducted as follows. A test number of pills for each slice for the first day was input and the pill holder slices filled with their respective pills. Then the sorting process was started for the first day of Sunday AM. The number of pills dispensed for each slice was then recorded. Results are recorded in Table 5 below.

Table 6: Sorting Test Results

Test	
Slices	Pills Needed
Slice 1	3
Slice 2	1
Slice 3	2
Slice 4	5
Slice 5	3
Slice 6	2
Results	
Slices	Pills Sorted
Slice 1	3
Slice 2	1
Slice 3	3
Slice 4	4
Slice 5	3
Slice 6	2

As seen in Table 6, the sorting system successfully sorted the correct number of pills from each slice except for slices 3 and 4. Slice 3 had an extra pill dispensed and slice 4 had one less pill dispensed. The reason for this was found with the drum motor. Since the drum motor is a DC

machine, once power is stopped to the motor, it will continue to spin for a little bit due to momentum. This drift gradually displaces the drum slot position so that it will either sort an extra pill or skip a pill. This appeared to occur every five rotations for two rotations. After two rotations, the drum drifts enough where normal operation can resume. This problem could potentially be mitigated or fixed by a more sophisticated control system for the DC motor that controls its exact position. Another solution could be to replace the DC motor with one with easier and more precise position control such as a stepper motor.

5.3 Cutting Test

The cutter test was conducted to assess whether the cutter correctly cut the pills in half and with a mass loss of less than 10% as designated by the previous cutter team. This is important as dosages of pills can be critical and any large variations can lead to dangerous and potentially even consequences for the user. The test was conducted as follows. Each pill was cut was a set number of times. Each pill was weighed before being cut and then each half was weighed after being cut. Pill one was an over-the-counter acetaminophen pill and as such allowed for more intensive testing. However, the other three pills tested came from the customer and were given in limited numbers. As such, only two to three tests were conducted for these pills. Results were recorded in Figure 36 and Table 6 below. The following equations were used to assess the mass loss of the pills and mass of each cut half of the pill in relation to the total mass.

$$\% \text{ Mass Loss} = \frac{M_{Half1} + M_{Half2}}{M_{Total}} * 100 \quad (5.1^{[5]})$$

$$\% \text{ Mass of Total Mass} = \frac{M_{Half}}{M_{Total}} * 100 \quad (5.2^{[5]})$$

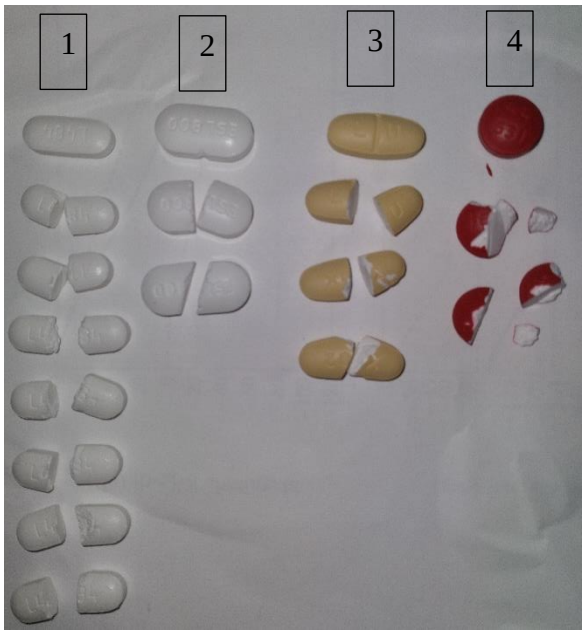


Figure 39: Cut Pills Result (Qualitative)(Right to Left: Pill 1, 2, 3, and 4)

As can be seen in Figure 39, the pills do not have a consistent cut pattern. The pills from pill 1 show that the pills slightly fracture at the cut line. Additionally, it can be seen that one half is larger than the other half for many of the pills. The same can be said for pills 2 and 3 as well. However, pill 4 shows that the pill completely fractures into two uneven pieces. This suggests that the cutter cuts rectangular pills more easily than round pills.

Table 6: Cut Pills Results (Quantitative) (Cont. on pg53)

	Total Dosage (mg)	Half 1 Dosage (mg)	% of Total Dosage	Half 2 Dosage (mg)	% of Total Dosage	New Total (mg)	Mass Loss (%)
Pill 1 Test 1	550	250	45.455	290	52.727	540	1.818
Pill 1 Test 2	510	260	50.980	240	47.059	500	1.961
Pill 1 Test 3	490	200	40.816	280	57.143	480	2.041
Pill 1 Test 4	510	220	43.137	260	50.980	480	5.882
Pill 1 Test 5	530	240	45.283	280	52.830	520	1.887
Pill 1 Test 6	550	250	45.455	280	50.909	530	3.636

Pill 1 Test 7	490	220	44.898	260	53.061	480	2.041
	Total Dosage (mg)	Half 1 Dosage (mg)	% of Total Dosage	Half 2 Dosage (mg)	% of Total Dosage	New Total (mg)	Mass Loss (%)
Pill 2 Test 1	910	430	47.253	450	49.451	880	3.297
Pill 2 Test 2	910	420	46.154	450	49.451	870	4.396
Pill 3 Test 1	670	350	52.239	290	43.284	640	4.478
Pill 3 Test 2	660	270	40.909	310	46.970	580	12.121
Pill 3 Test 3	660	260	39.394	290	43.939	550	16.667
Pill 4 Test 1	580	410	70.690	20	3.448	430	25.862
Pill 4 Test 2	600	250	41.667	250	41.667	500	16.667

As can be seen in Table 7, mass loss for all tests excluding the last 3 are within the 10% mass loss requirement with some as low as 2%. The mass loss for the last three tests are over 10% mass lass with one as high as 25%. These results match up with the results from the qualitative results for the round pills where the round pills do not get cut but completely fracture. As such, a large amount of mass loss is expected. As for the size of the halves, each half is no more than 55% of the total mass and no less than 40% of the total mass. As such, each half holds roughly 50% of the total mass with a variation of about 15% less than the total mass. However, such a large variation it is not ideal. The ideal variation would be less than 5% of the total mass for each half, accounting for the total maximum mass loss of 10%.

One potential cause for the errors is that in initial testing of the cutter system, the original blade used by the previous cutter team failed and was lost. Due to time constraints, a full replacement could not be retrieved. As such, a close alternative was used of similar diameter but with a rougher blade. Using this rougher blade could be the reason the pills fracture when cut.

Another potential cause for the errors is that the pills are not correctly slotted in the drum slot. Since the drum is designed to slot pills with their flat sides facing up, there is a large surface area where the pills could be slotted incorrectly. If the pills are slotted at odd angles, then the pills will not be cut down the middle as anticipated, resulting in uneven halves. Additionally, trying to cut the pills at odd angles could also result in the pill not cutting properly and as a result fracture.

6. Budget

This section will discuss the total cost of the design in an itemized list.

Table 8: Bill of Materials

Part	Price Per Unit ($)	Units	Cost ($)
nRF51 Microcontroller	50.00	1	50.00
Arduino Nano	10.00	2	20.00
Spektrum RC Avian 30 Amp Smart ESC	16.50	1	16.50
Gyros 81-11515 Blade	13.99	1	13.99
Brushed 30RPM DC Motor	12.99	1	12.99
Brushless 3 Phase Motor uxcell	36.99	1	36.99
8 mm Threaded Rod (200 mm)	11.00	2	22.00
Rod Rail Shaft Support	2.25	4	9.00
150 mm Linear Shaft	2.00	5	10.00
16 Key Matrix keypad 8 pin	8.99	1	8.99
20x4 LCD Module with I2C Adapter	10.99	1	10.99
Lever Arm Micro Limit Switch	0.50	2	1.00
NEMA 17 Stepper Motor 39mm Body	9.33	3	27.99
NEMA 17 Stepper Motor 23mm Body	8.99	1	8.99
NEMA 17 Stepper Motor Mounts	2.50	2	5.00
HiLetgo A4988 Stepper Motor Driver	1.89	2	3.78
2mm to 3mm Motor Shaft Coupler	6.99	1	6.99
24V Variable Power Supply	21.59	1	21.59
ELEGOO 120pcs Ribbon Wire	7.47	2	14.94
100uF Radial Electrolytic Capacitor	6.20	1	6.20
15SQ045 15A Schottky Diode	6.41	1	6.41
10 Pairs 12V DC Power Connector Jack	7.91	1	7.91
4pcs XT60 Male Female Connector	8.99	1	8.99
Nowepai 5Pcs PCB Power DPDT 5V Coil Micro Relay	7.99	1	7.99
AUSTOR 100Pcs PCB Board Kit	16.04	1	16.04
0.5in Wood Screws	30	1	50
M3 Machine Screws and Nuts	25	1	25
0.5x8x8in Plywood	33.50	1	33.50
Total cost			452.77

As shown in Table 8, the total cost of the project amounted to around $452.77 USD. The costliest parts for the project were the motors, construction materials, and microcontrollers. When compared against the cost of automated sorters on the market, it has a competitive cost since the

costs for current products range from $300 USD to $1200 USD[3]. Costs could have been driven down if some parts were sourced directly rather than through Amazon as most of these parts were.

7. Requirement Specifications

In this section the requirement specifications as required by the project guidelines that have not already been discussed within the report will be discussed.

7.1 Environmental

Environmental factors are also important when designing a product. An efficient design uses less energy. Thus, in the future, the machine should be designed with a low power setting when not in use. Therefore, the power usage of the machine will be minimized, minimizing the carbon footprint.

7.2 Public Health, Safety, and Welfare

Designing a product with the public's health and safety in mind is essential. If a product does not function in the way it was intended, users may be harmed. This machine was designed to prevent potential medical emergencies associated with prescription drug use, such as overdosing or underdosing. Sorting and counting must be performed with minimal error by the system (within 5%). In addition, the machine must follow the FDA's Code of Federal Regulations Title 21[5] regarding dispensers.

7.3 Global/Political

Currently, the machine will only be marketed to the USA, so global or political concerns outside of the USA are not being considered. As such, only American defined standards and

regulations will be followed, such as the FDA regulations on a pill cutter only having 10% mass loss for each cut.

7.4 Ethical and Professional

This is a machine that is supposed to help people save time. There are not any ethical problems when making this machine. There is nothing skipped in the production of this device or in the coding. Nothing involving safety has popped up that was skipped over, and everything was taken seriously. When it came to professional standards, it was not up to those. This machine was made as the best that it can be for college students and not being made at a professional company. This is not something that is mass produced or have a final product made from steel and plastic. While the production is not that professional the design process was above average. But since this was done by two college students that do not have professional experience it's hard to compare to professional standards.

8. Lessons Learned

Many lessons were learned over the course of the project. Firstly, time management was an important lesson. Throughout the semester, meeting deadlines has been a challenge for the team. There have been several times where deadlines have not been met and the resulting work to catch up only piled up. This ultimately increased our workload in the long run and potentially affected the quality of our work. Additionally, there were additional functions and features that we wished to implement but we had to scrap because of time constraints.

Secondly, it is important make sure high RPM motors such as the cutter motor are attached and coupled tightly to the base. There was one incident where the mounts on the cutter motor were not tightened enough, so the torque produced by the blade pushed the motor and blade backward

and out of the brackets. This resulted in a critical failure of the cutter system where the axle of the motor became bent as the blade tried to cut through the drum. Additionally, since the design is under a lot of vibration, especially during cutting, screws can very easily come loose. As such, we learned that it was imperative to check to make sure all screws and mounts are tightened down properly after running cutter. There was an incident where the screw that secured the blade to the cutter motor came loose. So, as the cutter motor got up to speed, it fell off the mount and became a projectile.

Thirdly, we learned important safety lessons. When there was a critical failure of the blade where it became a projectile, it was spinning at near 20,000 RPM when it launched. That presented an incredible danger. While we took appropriate precautions such as safety glasses and making sure we weren't in the direction of the blade, others may not be so fortunate. As such, in response, a wall was added in front of the blade so that if it did come off its mounting again, it would fly into the wall and stop there rather than become a projectile. This also later prevented the cutter motor and blade from flying out of the body when another critical failure occurred.

Fourthly, we learned about the conservation of power for our subsystems. Even though we had a power supply that connected directly to an outlet that supplied the required 1.5A of current needed for most of our motors, they all did not share the current equally. We especially learned that when implementing the cutter system. When we initially implemented the cutter system, all the motors remained powered including the stepper motors. However, the stepper motors take a lot current to maintain their position. As such, when the drum motor needed to rotate it could not rotate at all. As such, we implemented a function that would turn off the stepper motors when operating the drum motor and then turn them back on again after.

Lastly, we learned a lot about interactions between electrical and mechanical systems. When programming a motor for example, it is important to anticipate or predict the physical behavior of the item being actuated such as the pill holder and when problems are encountered, electrical and physical solution must be explored. One problem we encountered was that the stepper motor wouldn't have enough torque to overcome the friction between the pill slices and the holder. This resulted in attempts to compensate in the programming. This just led to needless code and bloat in the programming. What we eventually did was look at the physical design of the pill holder and slices and made the radius slightly larger for the pill holder. This left wiggle room for the slices and negated any of the previous friction. Additionally, we learned about communication between microcontrollers and efficient methods of applying the communication protocol to transfer the data needed. Since I2C was our chosen communication protocol, it was able to transmit up to 8 bits at a time. From this information we developed a data line protocol that included all the necessary information that needed to be transmitted within 8 bits.

9. Future Considerations

With every project there is always something to improve on and that is no different with this project. One of the more important improves that needs to be made to this machine is to implement a safety measure to prevent overdosing. A safety measurement put in place is a big thing when it comes to medicine being sorted automatically by a machine. Some people need to take specific amounts of their medicine/pills and if they don't take that exact amount there could be some harmful side effects or even deadly. A suggestion for such a system could be to add a digital scale that will weigh the total mass of all the pills sorted to ensure all pills have been sorted correctly.

Another improvement that needs to be implemented is a cleaning system for the cutter. After cutting tablet pills there can be a large amount of dust that piles up over time, so a vacuum could be a good answer to this issue.

The user modularity could also be improved such as the moving just past tablet pills being sorted in this system and being able to sort pills at a specific time of day. This machine is limited to just tablet pills because of the cutting option. This could easily be upgraded to include gel and capsule type pills that can be sorted but not cut. The add in the function of sorting a pill at a specific time of the day for those who need that option. This could be easily implemented using the current code base.

The next improvement is to add a more user-friendly interface for those to use. An LCD screen and a 4x4 punch pad is not the easiest to use. A touch screen can be a lot easier for some people since that happens to be the norm today and it might be easier or those with some motor skill problems to use a touch screen that is easier to use but also see.

Compared to other pill sorters on the market that has E-Health integration, this sorter just doesn't compare. So, adding that function that allows doctors/nurses to know if their patient has taken their medicines can put this machine in a league of its own because of its cutting capabilities.

Additionally, improvements on the drum design can be made to ensure that the pills are slotted correctly. A potential suggestion could be that the drum could be designed to slot pills vertically or on their thin sides rather than their long flat sides.

Lastly, a controller to control the precise position of the DC motor for the drum should be implemented to ensure the drum doesn't under or overdose pills. A potential solution could be the implementation of a voltage control system to control the speed with the addition of a potentiometer or encoder coupled to the shaft of the DC motor to measure position.

10. Teamwork

When it comes to a group project, it is always important for the team to work and communicate efficiently with each other. The team was organized by setting up a shared One-Drive file with all the work that has been done on the project. This eliminates having to email everyone in the group the work that has been done. This eliminates someone forgetting to add something to the email that needs to be sent. This also helped with communication, since it didn't force the team to use email for communication. The team relied on text when someone needed to talk to someone else. Text made it easy and efficient to communicate all information that is needed to be communicated. We were able to figure out each other's strengths quickly, so we were able to pick someone for the job who would be the best fit, as well as someone who would volunteer. We have not faced any disagreements or problems yet, but if we do, we will let both sides speak about the issue and find a solution that is logical and fair.

The main problem of teamwork that we had was the transition from ECE 471 to ENGR 491, which is preliminary class of senior design to the actual senior design class. The first class was in the Spring of 2021 and the second is in the Fall of 2021 and this transition was difficult for this team, because during the summer we had jobs and some of us had summer classes, so we couldn't keep in touch very much or work on the project as much as we would have liked. Additionally, In the Fall 2021 semester started back up, we learned that one of our teammates would not be attending USI that fall and therefore would not be on our team anymore. We had to overcome this and get back to work.

During the semester, the project sections were divided up between the two team members. The ULSS and master control system, and their respective designs were handled by one team member while the LLSS and its design was handled by another team member. We both cooperated

in incorporating and implementing our respective designs. Additionally, we both worked on the senior design report and design presentation together in our own times when we could, adding to it throughout the semester.

11. Conclusion

In conclusion, a design and prototype were made where up to six different shapes and sizes of pills are sorted automatically in a standard sized 7-day, AM and PM pill planner. This design also cuts pills in half, store, and dispense when necessary. Additionally, it incorporates user modularity so that the user may decide what pills are sorted, how many are sorted, and in what order they are sorted. However, the design is not perfect. The drum over or underdoses pills every five rotations. The drum design also does not guarantee the pills are slotted correctly, specifically round pills. This leads to pills being cut improperly resulting in fracturing. This also leads to uneven halves when pills are cut with variations of 40% up to 55% of the total mass. However, the cutter does cut pills with less than 10% mass loss, with mass loss as small as 2%. The exception to this is round pills with mass loses up to 25%. These flaws are rectifiable with potential improvements on the design such as using a finer tipped blade for the cutter, a drum design where the pills are slotted in sideways and implementing a control system to control the precise position of the DC motor.

References

[1] Barret. LL. Prescription Drug Use Among Midlife and Older Americans. https://assets.aarp.org/rgcenter/health/rx_midlife_plus.pdf (2005)

[2] CDC. Therapeutic Drug Use. https://www.cdc.gov/nchs/fastats/drug-use therapeutic.htm (2019)

[3]Clark A. Dispenser for Seniors. TheSeniorList. https://www.theseniorlist.com/medication/dispensers/ (2021)

[4] Electromate. NEMA 17 Motors. NEMA. https://www.electromate.com/products/stepper-motors/nema-stepper-motors/nema-17-stepper-motors/ (2021)

[5] FDA. Code of Federal Regulation Title 21. U.S. Department of Health. https://www.ecfr.gov/cgi-bin/textidx?SID=3764c91cae704aa7c1ae684401ec7141&mc=true&tpl=/ecfrbrowse/Title21/21tab_02.tpl (2021)

[6] Georgetown University Prescription Drugs. Georgetown University. https://hpi.georgetown.edu/rxdrugs/ (2002)

[7] Hess, J. Jeffries, J. Marchand, E. Murdock, D. Final Report Pill Cutter. University of Southern Indiana. (2020)

[8] Nordic Semiconductor. nRF51 Series Reference Manual. Nordic Semiconductor. https://infocenter.nordicsemi.com/pdf/nRF51_RM_v3.0.pdf (2014)

Appendix

Appendix A: Failure Modes and Effect Analysis

Table A.1 Failure Modes and Effects Analysis

Item	Failure Mode	Cause of Failure	Possible Effects	Likelihood	Level	Method to Reduce or Stop Effects
Cutter System	Motor axle bends.	1) Blade gets caught on drum when rotating. 2) Blade makes contact with pill while not spinning. 3) Axle gradually bends due to wear and tear.	1) Pills are not cut. 2) Drum breaks in half.	Medium	Critical	Ensure that the blade is correctly aligned with the drum so that there will be no contact between the drum and blade. Ensure that the drum properly dispenses pills.
Main Pill Holder	Skips a step when rotating.	1) A pill gets stuck under a slice, cause the holder to slightly get stuck. 2) A foreign object comes into contact with the holder when in motion.	1) Slices do not interface correctly with drum preventing dispensing of pills.	Low	High	Add a cap or lid to the pill holder so that no foreign object can come into contact with it. Additionally, possibly redesign pill holder so that there is no possibility of pills getting stuck under slices.
Drum	Drum breaks in half.	1) Critical failure of the blade which causes the blade to push on the drum or cut the center coupling rod. 2) Rough handling of the drum.	1) Pills are not dispensed or cut.	Medium	High	Make the drum out of a stronger material other than 3D printed plastic. Additionally, ensure the blade is appropriately interfaced with the drum.
Drum	Drum grabs an extra pill or does not grab a pill at all.	1) DC motor has a drift when turned off before fully stopping.	1) Over or underdosing.	High	Critical	Add a position control method to the DC motor such as a rotary encoder coupled to the shaft in order to compensate for the drift.

Appendix B: ABET Outcome 2, Design Factor Considerations

ABET Outcome 2 states "*An ability to apply engineering design to produce solutions that meet specified needs with consideration of public health safety, and welfare, as well as global, cultural, social, environmental, and economic factors.*"

ABET also requires that design projects reference appropriate professional standards, such as IEEE, ATSM, etc.

For each of the factors in Table N.1, indicate the page number(s) of your report where the item is addressed, or provide a statement regarding why the factor is not applicable for this project.

Table B.1: Design Factors Considered

Design Factor	Page number, or reason not applicable
Public health safety, and welfare	3,59
Global/Political	3,60
Cultural	1,2
Social	1,2
Environmental	59
Economic	58,59
Professional Standards	60

Appendix C: Master Control System Code

Appendix C.1: Master Control Code

```
/**
 * Copyright (c) 2016 - 2017, Nordic Semiconductor ASA
 *
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without modification,
 * are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice, this
 *    list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form, except as embedded into a Nordic
 *    Semiconductor ASA integrated circuit in a product or a software update for
 *    such product, must reproduce the above copyright notice, this list of
 *    conditions and the following disclaimer in the documentation and/or other
 *    materials provided with the distribution.
 *
 * 3. Neither the name of Nordic Semiconductor ASA nor the names of its
 *    contributors may be used to endorse or promote products derived from this
 *    software without specific prior written permission.
 *
 * 4. This software, with or without modification, must only be used with a
 *    Nordic Semiconductor ASA integrated circuit.
 *
 * 5. Any software provided in binary form under this license must not be reverse
 *    engineered, decompiled, modified and/or disassembled.
 *
 * THIS SOFTWARE IS PROVIDED BY NORDIC SEMICONDUCTOR ASA "AS IS" AND
ANY EXPRESS
 * OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES
 * OF MERCHANTABILITY, NONINFRINGEMENT, AND FITNESS FOR A PARTICULAR
PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL NORDIC SEMICONDUCTOR ASA OR
CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE
 * GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT
```

```
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY OUT
 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.
 *
 */

#include <stdio.h>
#include <math.h>
#include "boards.h"
#include "app_util_platform.h"
#include "app_error.h"
#include "nrf_drv_twi.h"
#include "nrf_delay.h"

#define NRF_LOG_MODULE_NAME "APP"

#include "nrf_log.h"
#include "nrf_log_ctrl.h"
#include "UI/UI.h"



// Pill Data //
/*int pill_Dosage[6] = {1, 1, 1, 1, 1, 1};
int pill_DosageDay[6][14] = {{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
                             {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
                             {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
                             {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
                             {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
                             {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}};*/

// Test Pill Data //
int pill_Dosage[6] = {100, 100, 100, 100, 100, 100};
int pill_DosageDay[6][14] = {{50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50},
                             {50, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
                             {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
                             {50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50},
                             {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
                             {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}};

uint8_t dayFilled = 0; // Filled compartments in pill planner.

/* TWI instance ID. */
#define TWI_INSTANCE_ID     1
```

```
/* TWI instance. */
static const nrf_drv_twi_t DATA_twi = NRF_DRV_TWI_INSTANCE(TWI_INSTANCE_ID);

// Data
uint8_t data = 0x0;
uint8_t address = 0x0;
ret_code_t err_code;

/**
 * @brief TWI initialization.
 */
void DATA_twi_init(void)
{
   ret_code_t err_code;

   const nrf_drv_twi_config_t DATA_twi_config = {
      .scl                = ARDUINO_SCL_PIN,
      .sda                = ARDUINO_SDA_PIN,
      .frequency          = NRF_TWI_FREQ_100K,
      .interrupt_priority = APP_IRQ_PRIORITY_HIGH,
      .clear_bus_init     = false
   };

   err_code = nrf_drv_twi_init(&DATA_twi, &DATA_twi_config, NULL, NULL);
   APP_ERROR_CHECK(err_code);

   //nrf_drv_twi_enable(&DATA_twi);
}

/**
 * @brief Function for main application entry.
 */

// Setting up functions
void mainMenu(void);

void ULSS(void);

void LLSS(void);

int main(void)
{
   // Intializes interfaces and TWI connections
   LCD_init();

   DATA_twi_init();
```

```
   while(1==1) // Runs the user interface
   {
    mainMenu();
   }
}

void mainMenu(void) // Responsible for the main menu of the user interface
{
 DATA_twiEnable(0); // Disable DATA_twi
 UI_twiEnable(0);// Enable UI_twi

 UI_MainMenu(); // Tells LCD to display main menu options

 int option = 0;

 option = UI_mainMenuSelect(Keypad()); // Determines which option the user picks; 1 is start
sorting and 0 is input pill data

 if(option == 1) // Start sorting
 {
  UI_twiEnable(1); // Disable UI_twi
  DATA_twiEnable(1); // Enable DATA_twi

  while(dayFilled <= 13) // sort pills for all 14 compartments of pill planner
  {
   LLSS(); // Tell LLSS to move to the next position

   nrf_delay_ms(500);

   ULSS(); // Tell ULSS to dispense pills for that position
  }

  dayFilled = 0; // Resets sorting

  DATA_twiEnable(0); // Disable DATA_twi
  UI_twiEnable(0);// Enable UI_twi
 }else if(option == 0) // Starts data input for pill data
 {
  int stop = 0;
  int pos = 0;
  int dosage[4] = {0, 0, 0, 0};

  for(int slice = 0; slice < 5; slice++) // Loops for all 6 slices or pills
  {
   UI_inputMenuPill(slice); // Asks user to input dosage for the respective slice
```

```
    stop = 0;
    pos = 0;
    for(int i = 0; i <= 3; i++) // Resets dosage handler for handling dosage number of pills
    {
     dosage[i] = 0;
    }
    while(stop != 2) // Continues through all slices and dosages for specific times of day until the
last slice is done
    {
     dosage[pos] = Keypad(); // Records what number the user inputs
     stop = UI_inputMenuPillInput(dosage[pos], pos); // Determines whether to stop sorting or
continue
     if(stop == 2)
     {
      pos = pos - 1;
     }
     else if(stop == 1) // Resets dosage input from pressing delete key on keypad
     {
      pos = 0;
      for(int i = 0; i <= 3; i++)
      {
       dosage[i] = 0;
      }
     }else if(stop == 0 && stop != 2) // Moves cursor for the LCD to the next position as well as
records the numerical position of the number input
     {
      pos = pos + 1;
     }
    }

    for(int i = 0; i <= pos; i++) // Converts input set of numbers into an actual value for the
dosage
    {
     pill_Dosage[slice] = dosage[i]*(powf(10, (pos - i))) + pill_Dosage[slice];
    }

    for(int day = 0; day <= 13; day++) // Loops for all 14 compartments of pill planner
    {
     UI_inputMenuDays(day);

     stop = 0;
     pos = 0;
     for(int i = 0; i <= 3; i++)
     {
      dosage[i] = 0;
```

```
      }
      while(stop != 2)
      {
       dosage[pos] = Keypad();
       stop = UI_inputMenuPillInput(dosage[pos], pos);
       if(stop == 2)
       {
        pos = pos - 1;
       }
       else if(stop == 1)
       {
        pos = 0;
        for(int i = 0; i <= 3; i++)
        {
         dosage[i] = 0;
        }
       }else if(stop == 0 && stop != 2)
       {
        pos = pos + 1;
       }
      }

      for(int i = 0; i <= pos; i++)
      {
       pill_DosageDay[slice][day] = dosage[i]*(powf(10, (pos - i))) +
pill_DosageDay[slice][day];
      }
     }
    }
   }
}

void ULSS(void) // Handles sending pill data to ULSS via TWI
{
  uint8_t slicePos = 0x0;
  uint8_t pillNum = 0x0;
  uint8_t cut = 0x0;

  address = 0x1; // ULSS TWI address

  for(int i = 0; i <= 5; i++) // Sends pill data from all six slices
  {
   float pillNumTemp = 0.5;

   slicePos = i;
```

```
   pillNumTemp = ((float)pill_DosageDay[slicePos][dayFilled])/((float)pill_Dosage[slicePos]);

   if(pillNumTemp == floorf(pillNumTemp)) // Determines if a half dose is needed; if even, then
no cut
   {
    cut = 0x0;
    pillNum = (uint8_t)pillNumTemp;
   }else // if odd, then cut
   {
    cut = 0x1;
    pillNum = (uint8_t)floorf(pillNumTemp);
   }

   data &= ~((0x1 << 7) | (0xF << 3) | (0x7 << 0));
   data |= ((cut << 7) | (pillNum << 3) | (slicePos << 0)); // Data line that holds pill data for ULSS

   nrf_drv_twi_tx(&DATA_twi, address, &data, sizeof(data), false); // Sends data to ULSS
  }

  data &= ~((0x1 << 7) | (0xF << 3) | (0x7 << 0));
  data |= (0x0 << 0); // Resets data register

  nrf_drv_twi_rx(&DATA_twi, address, &data, sizeof(data));
  while(data != 0x1) // Waits for confirmation bit from ULSS
  {
   // wait for ULSS confirmation //
   nrf_drv_twi_rx(&DATA_twi, address, &data, sizeof(data));
  }

  dayFilled = dayFilled + 1;
}

void LLSS(void) // Handles sending movement data to LLSS via TWI
{
  address = 0x2; // TWI address of LLSS

  data &= ~((0x1 << 8) | (0xF << 4) | (0x7 << 0));
  data |= (0x1 << 0); // Data bit to move LLSS

  nrf_drv_twi_tx(&DATA_twi, address, &data, sizeof(data), false);

  data &= ~((0x1 << 8) | (0xF << 4) | (0x7 << 0));
  data |= (0x0 << 0);

  nrf_drv_twi_rx(&DATA_twi, address, &data, sizeof(data));
  while(data != 0x1) // Waits for confirmation but from LLSS
```

```
  {
    // wait for LLSS confirmation //
    nrf_drv_twi_rx(&DATA_twi, address, &data, sizeof(data));
  }
}

void DATA_twiEnable(int option) // Enables and disables DATA twi line
{
  if(option == 1) // Enable
  {
    nrf_drv_twi_enable(&DATA_twi);
  }else if(option == 0) // Disable
  {
    nrf_drv_twi_disable(&DATA_twi);
  }
}
/** @} */
```

Appendix C.2: Keypad Code

```
#include "nrf.h"
#include "nrf51.h"
#include "nrf_delay.h"
#include "./NRF_BS/NRF_BS.h"

void inputChecker_set(char rc) // Detects which button was pressed on keypad for the numpad
{
 uint32_t rows[4] = {0x10, 0x11, 0x12, 0x13}; // Mapped positions of keypad pins (rows)
 uint32_t columns[4] = {0xC, 0xD, 0xE, 0xF}; // Mapped position of keypad pins (columns)
 int rows_num[4] = {16, 17, 18, 19};
 int columns_num[4] = {12, 13, 14, 15};

 if(rc == 'r') // Checks which rows have been pressed
 {
  for(int j = 0; j <= 3; j++)
  {
   GPIO_WRITE_BS(rows_num[j], 0); // turn off output
   GPIO_CONFIG_BS(rows_num[j], 0, 0, 1, 0, 0); // input
   GPIO_CONFIG_BS(columns_num[j], 1, 1, 0, 0, 0); // output
   GPIO_WRITE_BS(columns_num[j], 1); // turn on output
  }

  for(int i = 0; i <= 3; i++)
  {
   NRF_GPIOTE->CONFIG[i] &= ~((0x3 << 0) | (0x1F << 8) | (0x3 << 16) | (0x1 << 20)); //
reset peripheral
  }

  for(int i = 0; i <= 3; i++)
  {
   NRF_GPIOTE->CONFIG[i] |= ((0x1 << 0) | (rows[i] << 8) | (0x2 << 16) | (0x0 << 20)); //
sets rows for detecting an input
  }
 }else if(rc == 'c') // Checks which columns have been pressed
 {
  for(int j = 0; j <= 3; j++)
  {
   GPIO_WRITE_BS(columns_num[j], 0); // turn off output
   GPIO_CONFIG_BS(columns_num[j], 0, 0, 1, 0, 0); // input
   GPIO_CONFIG_BS(rows_num[j], 1, 1, 0, 0, 0); // output
   GPIO_WRITE_BS(rows_num[j], 1); // turn on output
  }

  for(int i = 0; i <= 3; i++)
```

```
    {
     NRF_GPIOTE->CONFIG[i] &= ~((0x3 << 0) | (0x1F << 8) | (0x3 << 16) | (0x1 << 20)); //
reset peripheral
    }

    for(int i = 0; i <= 3; i++)
    {s
     NRF_GPIOTE->CONFIG[i] |= ((0x1 << 0) | (columns[i] << 8) | (0x1 << 16) | (0x0 << 20));
// sets columns for detecting an input
    }
  }
}

int Keypad_check() // Detects what button is pressed and records the value
{
  int trigger = 0;
  int ans = 0;

  while(trigger != 1)
  {
    for(int i = 0; i <= 3; i++)
    {
     if(NRF_GPIOTE->EVENTS_IN[i] == 1)
     {
       trigger = 1;
       NRF_GPIOTE->EVENTS_IN[i] = 0;
       ans = i;
       nrf_delay_ms(10);
       break;
     }
    }
  }
  for(int i = 0; i <= 3; i++)
  {
    NRF_GPIOTE->EVENTS_IN[i] = 0;
  }
  return ans;
}

int Keypad() // Controls the whole keypad mechanism and returns what key/button was pressed
on the keypad
{
  int r = 0;
  int c = 0;
  int keys[4][4] = {{41, 100, 0, 101}, {31, 9, 8, 7}, {21, 6, 5, 4}, {11, 3, 2, 1}}; // 11 to 41 is F1
to F4, and 101 and 100 are enter and stop, respectively
```

```
  inputChecker_set('c'); // set columns to receive inputs and generate event
  c = Keypad_check();

  inputChecker_set('r'); // set rows to receive inputs and generate event
  r = Keypad_check();

  return keys[r][c];
}
```

Appendix C.3: LCD Code

```
#include <stdio.h>
#include "nrf.h"
#include "nrf51.h"
#include "boards.h"
#include "app_util_platform.h"
#include "app_error.h"
#include "nrf_drv_twi.h"
#include "nrf_delay.h"

// LCD DEFINES //

// commands
#define LCD_CLEARDISPLAY 0x01
#define LCD_RETURNHOME 0x02
#define LCD_ENTRYMODESET 0x04
#define LCD_DISPLAYCONTROL 0x08
#define LCD_CURSORSHIFT 0x10
#define LCD_FUNCTIONSET 0x20
#define LCD_SETCGRAMADDR 0x40
#define LCD_SETDDRAMADDR 0x80

// flags for display entry mode
#define LCD_ENTRYRIGHT 0x00
#define LCD_ENTRYLEFT 0x02
#define LCD_ENTRYSHIFTINCREMENT 0x01
#define LCD_ENTRYSHIFTDECREMENT 0x00

// flags for display on/off control
#define LCD_DISPLAYON 0x04
#define LCD_DISPLAYOFF 0x00
#define LCD_CURSORON 0x02
#define LCD_CURSOROFF 0x00
#define LCD_BLINKON 0x01
#define LCD_BLINKOFF 0x00

// flags for display/cursor shift
#define LCD_DISPLAYMOVE 0x08
#define LCD_CURSORMOVE 0x00
#define LCD_MOVERIGHT 0x04
#define LCD_MOVELEFT 0x00

// flags for function set
#define LCD_8BITMODE 0x10
#define LCD_4BITMODE 0x00
#define LCD_2LINE 0x08
```

```
#define LCD_1LINE 0x00
#define LCD_5x10DOTS 0x04
#define LCD_5x8DOTS 0x00

// flags for backlight control
#define LCD_BACKLIGHT 0x08
#define LCD_NOBACKLIGHT 0x00

#define En 0x4  // Enable bit
#define Rw 0x2  // Read/Write bit
#define Rs 0x1  // Register select bit

// variables
uint8_t LCD_Address = 0x27;
uint8_t _backlightval = LCD_BACKLIGHT;
uint8_t _displayfunction = LCD_4BITMODE | LCD_2LINE | LCD_5x8DOTS;
uint8_t _displaycontrol = LCD_DISPLAYON | LCD_CURSOROFF | LCD_BLINKOFF;
uint8_t _displaymode = LCD_ENTRYLEFT | LCD_ENTRYSHIFTDECREMENT;

int keyNums[10] = {0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39};

// TWI DEFINES //

/* TWI instance ID. */
#define TWI_INSTANCE_ID     0

/* TWI instance. */
static const nrf_drv_twi_t UI_twi = NRF_DRV_TWI_INSTANCE(TWI_INSTANCE_ID);

// LCD IRQ COMMAND //

void UI_twi_init(void)
{
   ret_code_t err_code;

   const nrf_drv_twi_config_t UI_twi_config = {
      .scl = ARDUINO_SCL_PIN,
      .sda = ARDUINO_SDA_PIN,
      .frequency = NRF_TWI_FREQ_100K,
      .interrupt_priority = APP_IRQ_PRIORITY_HIGH,
      .clear_bus_init = false
   };

   err_code = nrf_drv_twi_init(&UI_twi, &UI_twi_config, NULL, NULL);
   APP_ERROR_CHECK(err_code);
```

```
    nrf_drv_twi_enable(&UI_twi);
}

// LCD COMMANDS //

void LCD_transmit(uint8_t input) // Transmits bits over TWI
{
    uint8_t data = (input | _backlightval);
    nrf_drv_twi_tx(&UI_twi, LCD_Address, &data, sizeof(data), false);
}

void LCD_pulseEnable(uint8_t pulse) //1 is enable and 0 is disable
{
    LCD_transmit(pulse | En);
    nrf_delay_us(1);

    LCD_transmit(pulse & ~En);
    nrf_delay_us(50);
}

void LCD_write4bits(uint8_t value) // Sends data over 4 bits
{
    LCD_transmit(value);
    LCD_pulseEnable(value);
}

void LCD_send(uint8_t value, uint8_t mode)
{
    uint8_t highnib = value & 0xf0;
    uint8_t lownib = (value << 4) & 0xf0;

    LCD_write4bits((highnib) | mode);
    LCD_write4bits((lownib) | mode);
}

void LCD_command(uint8_t command)
{
    LCD_send(command, 0);
}

void LCD_write(uint8_t chara)
{
    LCD_send(chara, 1);
}

void LCD_display()
```

```
{
  _displaycontrol |= LCD_DISPLAYON;
  LCD_command(LCD_DISPLAYCONTROL | _displaycontrol);
}

void LCD_clear()
{
  LCD_command(LCD_CLEARDISPLAY);
  nrf_delay_us(2000);
}

void LCD_home()
{
  LCD_command(LCD_RETURNHOME);
  nrf_delay_us(2000);
}

void backlight()
{
  _backlightval = LCD_BACKLIGHT;
  LCD_transmit(0);
}

void LCD_setCursor(uint8_t col, uint8_t row)
{
  int row_offsets[] = { 0x00, 0x40, 0x14, 0x54 };
  if (row > 4)
  {
    row = 4 - 1;
  }
  LCD_command(LCD_SETDDRAMADDR | (col + row_offsets[row]));
}

void LCD_init() // Initializes LCD
{
  UI_twi_init();

  nrf_delay_ms(50);

  LCD_transmit(0x8);
  nrf_delay_ms(1000);

  LCD_write4bits(0x03 << 4);
  nrf_delay_ms(5);

  LCD_write4bits(0x03 << 4);
```

```
    nrf_delay_ms(5);

    LCD_write4bits(0x03 << 4);
    nrf_delay_us(150);

    LCD_write4bits(0x02 << 4);

    LCD_command(LCD_FUNCTIONSET | _displayfunction);

    LCD_display();

    LCD_clear();

    LCD_command(LCD_ENTRYMODESET | _displaymode);

    LCD_home();

    nrf_drv_twi_disable(&UI_twi);
}

void UI_MainMenu() // Initializes the main menu UI
{
    nrf_drv_twi_enable(&UI_twi);

    LCD_setCursor(0, 0);

    // F1)
    LCD_write(0x46); // F
    LCD_write(0x31); // 1
    LCD_write(0x29); // )
    LCD_write(0x20); // space

    // Input
    LCD_write(0x49); // I
    LCD_write(0x6E); // n
    LCD_write(0x70); // p
    LCD_write(0x75); // u
    LCD_write(0x74); // t
    LCD_write(0x20); // space

    // Pill
    LCD_write(0x50); // P
    LCD_write(0x69); // i
    LCD_write(0x6C); // l
    LCD_write(0x6C); // l
    LCD_write(0x20); // space
```

```
    // Data
    LCD_write(0x44); // D
    LCD_write(0x61); // a
    LCD_write(0x74); // t
    LCD_write(0x61); // a
    LCD_write(0x20); // space

    LCD_setCursor(0, 2);

    // F2)
    LCD_write(0x46); // F
    LCD_write(0x32); // 2
    LCD_write(0x29); // )
    LCD_write(0x20); // space

    // Start
    LCD_write(0x53); // S
    LCD_write(0x74); // t
    LCD_write(0x61); // a
    LCD_write(0x72); // r
    LCD_write(0x74); // t
    LCD_write(0x20); // space

    // Sorting
    LCD_write(0x53); // S
    LCD_write(0x6F); // o
    LCD_write(0x72); // r
    LCD_write(0x74); // t
    LCD_write(0x69); // i
    LCD_write(0x6E); // n
    LCD_write(0x67); // g
}

int UI_mainMenuSelect(int option) // Facilitates the option selection for the main menu
{
    if (option == 11)
    {
        LCD_clear();

        LCD_setCursor(0, 0);

        LCD_write(0x50); // P
        LCD_write(0x6C); // l
        LCD_write(0x65); // e
        LCD_write(0x61); // a
```

```
    LCD_write(0x73); // s
    LCD_write(0x65); // e
    LCD_write(0x20); // space

    LCD_write(0x46); // F
    LCD_write(0x6F); // o
    LCD_write(0x6C); // l
    LCD_write(0x6C); // l
    LCD_write(0x6F); // o
    LCD_write(0x77); // w

    LCD_setCursor(0, 2);
    LCD_write(0x54); // T
    LCD_write(0x68); // h
    LCD_write(0x65); // e
    LCD_write(0x20); // space

    LCD_write(0x49); // I
    LCD_write(0x6E); // n
    LCD_write(0x73); // s
    LCD_write(0x74); // t
    LCD_write(0x72); // r
    LCD_write(0x75); // u
    LCD_write(0x63); // c
    LCD_write(0x74); // t
    LCD_write(0x69); // i
    LCD_write(0x6F); // o
    LCD_write(0x6E); // n
    LCD_write(0x73); // s

    nrf_delay_ms(1000);

    return 0;
}
else if (option == 21)
{
    LCD_clear();

    LCD_setCursor(0, 0);

    LCD_write(0x4E); // N
    LCD_write(0x6F); // o
    LCD_write(0x77); // w
    LCD_write(0x20); // space

    LCD_setCursor(0, 2);
```

```
      LCD_write(0x53); // S
      LCD_write(0x6F); // o
      LCD_write(0x72); // r
      LCD_write(0x74); // t
      LCD_write(0x69); // i
      LCD_write(0x6E); // n
      LCD_write(0x67); // g

      nrf_delay_ms(1000);

      return 1;
    }
}

void UI_inputMenuPill(int slice) // Controls the UI for inputting pill data per slice
{
 LCD_clear();

 LCD_setCursor(0, 0);

 switch(slice)
 {
  case 0:
    // Input
    LCD_write(0x49); // I
    LCD_write(0x6E); // n
    LCD_write(0x70); // p
    LCD_write(0x75); // u
    LCD_write(0x74); // t
    LCD_write(0x20); // space

    // Dosage
    LCD_write(0x44); // D
    LCD_write(0x6F); // o
    LCD_write(0x73); // s
    LCD_write(0x61); // a
    LCD_write(0x67); // g
    LCD_write(0x65); // e
    LCD_write(0x20); // space

    // For
    LCD_write(0x46); // F
    LCD_write(0x6F); // o
    LCD_write(0x72); // r
```

```
  LCD_setCursor(0, 1);

  // Pill
  LCD_write(0x50); // P
  LCD_write(0x69); // i
  LCD_write(0x6C); // l
  LCD_write(0x6C); // l
  LCD_write(0x20); // space

  // One:
  LCD_write(0x4F); // O
  LCD_write(0x6E); // n
  LCD_write(0x65); // e
  LCD_write(0x3A); // :

  LCD_setCursor(5, 2);

  // mg
  LCD_write(0x6D); // m
  LCD_write(0x67); // g

  LCD_setCursor(0, 2);

  // ____
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  break;

case 1:
  // Input
  LCD_write(0x49); // I
  LCD_write(0x6E); // n
  LCD_write(0x70); // p
  LCD_write(0x75); // u
  LCD_write(0x74); // t
  LCD_write(0x20); // space

  // Dosage
  LCD_write(0x44); // D
  LCD_write(0x6F); // o
  LCD_write(0x73); // s
  LCD_write(0x61); // a
  LCD_write(0x67); // g
  LCD_write(0x65); // e
```

```
  LCD_write(0x20); // space

  // For
  LCD_write(0x46); // F
  LCD_write(0x6F); // o
  LCD_write(0x72); // r

  LCD_setCursor(0, 1);

  // Pill
  LCD_write(0x50); // P
  LCD_write(0x69); // i
  LCD_write(0x6C); // l
  LCD_write(0x6C); // l
  LCD_write(0x20); // space

  // Two:
  LCD_write(0x54); // T
  LCD_write(0x77); // w
  LCD_write(0x6F); // o
  LCD_write(0x3A); // :

  LCD_setCursor(5, 2);

  // mg
  LCD_write(0x6D); // m
  LCD_write(0x67); // g

  LCD_setCursor(0, 2);

  // ____
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  break;

case 2:
  // Input
  LCD_write(0x49); // I
  LCD_write(0x6E); // n
  LCD_write(0x70); // p
  LCD_write(0x75); // u
  LCD_write(0x74); // t
  LCD_write(0x20); // space
```

```
// Dosage
LCD_write(0x44); // D
LCD_write(0x6F); // o
LCD_write(0x73); // s
LCD_write(0x61); // a
LCD_write(0x67); // g
LCD_write(0x65); // e
LCD_write(0x20); // space

// For
LCD_write(0x46); // F
LCD_write(0x6F); // o
LCD_write(0x72); // r

LCD_setCursor(0, 1);

// Pill
LCD_write(0x50); // P
LCD_write(0x69); // i
LCD_write(0x6C); // l
LCD_write(0x6C); // l
LCD_write(0x20); // space

// Three:
LCD_write(0x54); // T
LCD_write(0x68); // h
LCD_write(0x72); // r
LCD_write(0x65); // e
LCD_write(0x65); // e
LCD_write(0x3A); // :

LCD_setCursor(5, 2);

// mg
LCD_write(0x6D); // m
LCD_write(0x67); // g

LCD_setCursor(0, 2);

// ____
LCD_write(0x5F); // _
LCD_write(0x5F); // _
LCD_write(0x5F); // _
LCD_write(0x5F); // _
break;
```

```
case 3:
  // Input
  LCD_write(0x49); // I
  LCD_write(0x6E); // n
  LCD_write(0x70); // p
  LCD_write(0x75); // u
  LCD_write(0x74); // t
  LCD_write(0x20); // space

  // Dosage
  LCD_write(0x44); // D
  LCD_write(0x6F); // o
  LCD_write(0x73); // s
  LCD_write(0x61); // a
  LCD_write(0x67); // g
  LCD_write(0x65); // e
  LCD_write(0x20); // space

  // For
  LCD_write(0x46); // F
  LCD_write(0x6F); // o
  LCD_write(0x72); // r

  LCD_setCursor(0, 1);

  // Pill
  LCD_write(0x50); // P
  LCD_write(0x69); // i
  LCD_write(0x6C); // l
  LCD_write(0x6C); // l
  LCD_write(0x20); // space

  // Four:
  LCD_write(0x46); // F
  LCD_write(0x6F); // o
  LCD_write(0x75); // u
  LCD_write(0x72); // r
  LCD_write(0x3A); // :

  LCD_setCursor(5, 2);

  // mg
  LCD_write(0x6D); // m
  LCD_write(0x67); // g

  LCD_setCursor(0, 2);
```

```
  // ____
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  break;

case 4:
  // Input
  LCD_write(0x49); // I
  LCD_write(0x6E); // n
  LCD_write(0x70); // p
  LCD_write(0x75); // u
  LCD_write(0x74); // t
  LCD_write(0x20); // space

  // Dosage
  LCD_write(0x44); // D
  LCD_write(0x6F); // o
  LCD_write(0x73); // s
  LCD_write(0x61); // a
  LCD_write(0x67); // g
  LCD_write(0x65); // e
  LCD_write(0x20); // space

  // For
  LCD_write(0x46); // F
  LCD_write(0x6F); // o
  LCD_write(0x72); // r

  LCD_setCursor(0, 1);

  // Pill
  LCD_write(0x50); // P
  LCD_write(0x69); // i
  LCD_write(0x6C); // l
  LCD_write(0x6C); // l
  LCD_write(0x20); // space

  // Five:
  LCD_write(0x46); // F
  LCD_write(0x69); // i
  LCD_write(0x76); // v
  LCD_write(0x65); // e
  LCD_write(0x3A); // :
```

```
  LCD_setCursor(5, 2);

  // mg
  LCD_write(0x6D); // m
  LCD_write(0x67); // g

  LCD_setCursor(0, 2);

  // ____
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  break;

case 5:
  // Input
  LCD_write(0x49); // I
  LCD_write(0x6E); // n
  LCD_write(0x70); // p
  LCD_write(0x75); // u
  LCD_write(0x74); // t
  LCD_write(0x20); // space

  // Dosage
  LCD_write(0x44); // D
  LCD_write(0x6F); // o
  LCD_write(0x73); // s
  LCD_write(0x61); // a
  LCD_write(0x67); // g
  LCD_write(0x65); // e
  LCD_write(0x20); // space

  // For
  LCD_write(0x46); // F
  LCD_write(0x6F); // o
  LCD_write(0x72); // r

  LCD_setCursor(0, 1);

  // Pill
  LCD_write(0x50); // P
  LCD_write(0x69); // i
  LCD_write(0x6C); // l
  LCD_write(0x6C); // l
```

```
    LCD_write(0x20); // space

    // Six:
    LCD_write(0x53); // S
    LCD_write(0x69); // i
    LCD_write(0x78); // x
    LCD_write(0x3A); // :

    LCD_setCursor(5, 2);

    // mg
    LCD_write(0x6D); // m
    LCD_write(0x67); // g

    LCD_setCursor(0, 2);

    // ____
    LCD_write(0x5F); // _
    LCD_write(0x5F); // _
    LCD_write(0x5F); // _
    LCD_write(0x5F); // _
    break;
  }
}

int UI_inputMenuPillInput(int key, int pos) // Controls the UI for inputting pill data numbers
{
  if(key == 101) // enter (start key)
  {
    return 2;
  }

  if(key == 100) // delete (stop key)
  {
    LCD_setCursor(0, 2);

    // ____
    LCD_write(0x5F); // _
    LCD_write(0x5F); // _
    LCD_write(0x5F); // _
    LCD_write(0x5F); // _

    return 1;
  }

  if(pos > 3)
```

```
{
 return 0;
}

LCD_setCursor(pos, 2);

switch(key)
{
 case 0:
  LCD_write(keyNums[0]);
  break;

 case 1:
  LCD_write(keyNums[1]);
  break;

 case 2:
  LCD_write(keyNums[2]);
  break;

 case 3:
  LCD_write(keyNums[3]);
  break;

 case 4:
  LCD_write(keyNums[4]);
  break;

 case 5:
  LCD_write(keyNums[5]);
  break;

 case 6:
  LCD_write(keyNums[6]);
  break;

 case 7:
  LCD_write(keyNums[7]);
  break;

 case 8:
  LCD_write(keyNums[8]);
  break;

 case 9:
  LCD_write(keyNums[9]);
```

```
    break;
  }

  return 0;
}

void UI_inputMenuDays(int day)// Controls the UI for inputting pill data per pill planner compartment
{
  LCD_clear();

  LCD_setCursor(0, 0);

  switch(day)
  {
    case 0: // Sunday AM
      // Input
      LCD_write(0x49); // I
      LCD_write(0x6E); // n
      LCD_write(0x70); // p
      LCD_write(0x75); // u
      LCD_write(0x74); // t
      LCD_write(0x20); // space

      // Dosage
      LCD_write(0x44); // D
      LCD_write(0x6F); // o
      LCD_write(0x73); // s
      LCD_write(0x61); // a
      LCD_write(0x67); // g
      LCD_write(0x65); // e
      LCD_write(0x20); // space

      // For
      LCD_write(0x46); // F
      LCD_write(0x6F); // o
      LCD_write(0x72); // r

      LCD_setCursor(0, 1);

      // Sunday
      LCD_write(0x53); // S
      LCD_write(0x75); // u
      LCD_write(0x6E); // n
      LCD_write(0x64); // d
      LCD_write(0x61); // a
```

```
  LCD_write(0x79); // y
  LCD_write(0x20); // space

  // AM:
  LCD_write(0x41); // A
  LCD_write(0x4D); // M
  LCD_write(0x3A); // :

  LCD_setCursor(0, 2);

  // ____
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  break;

case 1: // Sunday PM
  // Input
  LCD_write(0x49); // I
  LCD_write(0x6E); // n
  LCD_write(0x70); // p
  LCD_write(0x75); // u
  LCD_write(0x74); // t
  LCD_write(0x20); // space

  // Dosage
  LCD_write(0x44); // D
  LCD_write(0x6F); // o
  LCD_write(0x73); // s
  LCD_write(0x61); // a
  LCD_write(0x67); // g
  LCD_write(0x65); // e
  LCD_write(0x20); // space

  // For
  LCD_write(0x46); // F
  LCD_write(0x6F); // o
  LCD_write(0x72); // r

  LCD_setCursor(0, 1);

  // Sunday
  LCD_write(0x53); // S
  LCD_write(0x75); // u
  LCD_write(0x6E); // n
```

```
  LCD_write(0x64); // d
  LCD_write(0x61); // a
  LCD_write(0x79); // y
  LCD_write(0x20); // space

  // PM:
  LCD_write(0x50); // P
  LCD_write(0x4D); // M
  LCD_write(0x3A); // :

  LCD_setCursor(0, 2);

  // ____
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  break;

case 2: // Monday AM
  // Input
  LCD_write(0x49); // I
  LCD_write(0x6E); // n
  LCD_write(0x70); // p
  LCD_write(0x75); // u
  LCD_write(0x74); // t
  LCD_write(0x20); // space

  // Dosage
  LCD_write(0x44); // D
  LCD_write(0x6F); // o
  LCD_write(0x73); // s
  LCD_write(0x61); // a
  LCD_write(0x67); // g
  LCD_write(0x65); // e
  LCD_write(0x20); // space

  // For
  LCD_write(0x46); // F
  LCD_write(0x6F); // o
  LCD_write(0x72); // r

  LCD_setCursor(0, 1);

  // Monday
  LCD_write(0x4D); // M
```

```
  LCD_write(0x6F); // o
  LCD_write(0x6E); // n
  LCD_write(0x64); // d
  LCD_write(0x61); // a
  LCD_write(0x79); // y
  LCD_write(0x20); // space

  // AM:
  LCD_write(0x41); // A
  LCD_write(0x4D); // M
  LCD_write(0x3A); // :

  LCD_setCursor(0, 2);

  // ____
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  break;

case 3: // Monday PM
  // Input
  LCD_write(0x49); // I
  LCD_write(0x6E); // n
  LCD_write(0x70); // p
  LCD_write(0x75); // u
  LCD_write(0x74); // t
  LCD_write(0x20); // space

  // Dosage
  LCD_write(0x44); // D
  LCD_write(0x6F); // o
  LCD_write(0x73); // s
  LCD_write(0x61); // a
  LCD_write(0x67); // g
  LCD_write(0x65); // e
  LCD_write(0x20); // space

  // For
  LCD_write(0x46); // F
  LCD_write(0x6F); // o
  LCD_write(0x72); // r

  LCD_setCursor(0, 1);
```

```
  // Monday
  LCD_write(0x4D); // M
  LCD_write(0x6F); // o
  LCD_write(0x6E); // n
  LCD_write(0x64); // d
  LCD_write(0x61); // a
  LCD_write(0x79); // y
  LCD_write(0x20); // space

  // PM:
  LCD_write(0x50); // P
  LCD_write(0x4D); // M
  LCD_write(0x3A); // :

  LCD_setCursor(0, 2);

  // ____
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  break;

case 4: // Tuesday AM
  // Input
  LCD_write(0x49); // I
  LCD_write(0x6E); // n
  LCD_write(0x70); // p
  LCD_write(0x75); // u
  LCD_write(0x74); // t
  LCD_write(0x20); // space

  // Dosage
  LCD_write(0x44); // D
  LCD_write(0x6F); // o
  LCD_write(0x73); // s
  LCD_write(0x61); // a
  LCD_write(0x67); // g
  LCD_write(0x65); // e
  LCD_write(0x20); // space

  // For
  LCD_write(0x46); // F
  LCD_write(0x6F); // o
  LCD_write(0x72); // r
```

```
  LCD_setCursor(0, 1);

  // Tuesday
  LCD_write(0x54); // T
  LCD_write(0x75); // u
  LCD_write(0x65); // e
  LCD_write(0x73); // s
  LCD_write(0x64); // d
  LCD_write(0x61); // a
  LCD_write(0x79); // y
  LCD_write(0x20); // space

  // AM:
  LCD_write(0x41); // A
  LCD_write(0x4D); // M
  LCD_write(0x3A); // :

  LCD_setCursor(0, 2);

  // ____
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  break;

case 5: // Tuesday PM
  // Input
  LCD_write(0x49); // I
  LCD_write(0x6E); // n
  LCD_write(0x70); // p
  LCD_write(0x75); // u
  LCD_write(0x74); // t
  LCD_write(0x20); // space

  // Dosage
  LCD_write(0x44); // D
  LCD_write(0x6F); // o
  LCD_write(0x73); // s
  LCD_write(0x61); // a
  LCD_write(0x67); // g
  LCD_write(0x65); // e
  LCD_write(0x20); // space

  // For
  LCD_write(0x46); // F
```

```
  LCD_write(0x6F); // o
  LCD_write(0x72); // r

  LCD_setCursor(0, 1);

  // Tuesday
  LCD_write(0x54); // T
  LCD_write(0x75); // u
  LCD_write(0x65); // e
  LCD_write(0x73); // s
  LCD_write(0x64); // d
  LCD_write(0x61); // a
  LCD_write(0x79); // y
  LCD_write(0x20); // space

  // PM:
  LCD_write(0x50); // P
  LCD_write(0x4D); // M
  LCD_write(0x3A); // :

  LCD_setCursor(0, 2);

  // ____
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  break;
  break;

case 6: // Wednesday AM
  // Input
  LCD_write(0x49); // I
  LCD_write(0x6E); // n
  LCD_write(0x70); // p
  LCD_write(0x75); // u
  LCD_write(0x74); // t
  LCD_write(0x20); // space

  // Dosage
  LCD_write(0x44); // D
  LCD_write(0x6F); // o
  LCD_write(0x73); // s
  LCD_write(0x61); // a
  LCD_write(0x67); // g
  LCD_write(0x65); // e
```

```
  LCD_write(0x20); // space

  // For
  LCD_write(0x46); // F
  LCD_write(0x6F); // o
  LCD_write(0x72); // r

  LCD_setCursor(0, 1);

  // Wednesday
  LCD_write(0x57); // W
  LCD_write(0x65); // e
  LCD_write(0x64); // d
  LCD_write(0x6E); // n
  LCD_write(0x65); // e
  LCD_write(0x73); // s
  LCD_write(0x64); // d
  LCD_write(0x61); // a
  LCD_write(0x79); // y
  LCD_write(0x20); // space

  // AM:
  LCD_write(0x41); // A
  LCD_write(0x4D); // M
  LCD_write(0x3A); // :

  LCD_setCursor(0, 2);

  // ____
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  break;

case 7: // Wednesday PM
  // Input
  LCD_write(0x49); // I
  LCD_write(0x6E); // n
  LCD_write(0x70); // p
  LCD_write(0x75); // u
  LCD_write(0x74); // t
  LCD_write(0x20); // space

  // Dosage
  LCD_write(0x44); // D
```

```
  LCD_write(0x6F); // o
  LCD_write(0x73); // s
  LCD_write(0x61); // a
  LCD_write(0x67); // g
  LCD_write(0x65); // e
  LCD_write(0x20); // space

  // For
  LCD_write(0x46); // F
  LCD_write(0x6F); // o
  LCD_write(0x72); // r

  LCD_setCursor(0, 1);

  // Wednesday
  LCD_write(0x57); // W
  LCD_write(0x65); // e
  LCD_write(0x64); // d
  LCD_write(0x6E); // n
  LCD_write(0x65); // e
  LCD_write(0x73); // s
  LCD_write(0x64); // d
  LCD_write(0x61); // a
  LCD_write(0x79); // y
  LCD_write(0x20); // space

  // PM:
  LCD_write(0x50); // P
  LCD_write(0x4D); // M
  LCD_write(0x3A); // :

  LCD_setCursor(0, 2);

  // ____
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  break;

case 8: // Thursday AM
  // Input
  LCD_write(0x49); // I
  LCD_write(0x6E); // n
  LCD_write(0x70); // p
  LCD_write(0x75); // u
```

```
  LCD_write(0x74); // t
  LCD_write(0x20); // space

  // Dosage
  LCD_write(0x44); // D
  LCD_write(0x6F); // o
  LCD_write(0x73); // s
  LCD_write(0x61); // a
  LCD_write(0x67); // g
  LCD_write(0x65); // e
  LCD_write(0x20); // space

  // For
  LCD_write(0x46); // F
  LCD_write(0x6F); // o
  LCD_write(0x72); // r

  LCD_setCursor(0, 1);

  // Thursday
  LCD_write(0x54); // T
  LCD_write(0x68); // h
  LCD_write(0x75); // u
  LCD_write(0x72); // r
  LCD_write(0x73); // s
  LCD_write(0x64); // d
  LCD_write(0x61); // a
  LCD_write(0x79); // y
  LCD_write(0x20); // space

  // AM:
  LCD_write(0x41); // A
  LCD_write(0x4D); // M
  LCD_write(0x3A); // :

  LCD_setCursor(0, 2);

  // ____
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  break;

case 9: // Thursday PM
  // Input
```

```
LCD_write(0x49); // I
LCD_write(0x6E); // n
LCD_write(0x70); // p
LCD_write(0x75); // u
LCD_write(0x74); // t
LCD_write(0x20); // space

// Dosage
LCD_write(0x44); // D
LCD_write(0x6F); // o
LCD_write(0x73); // s
LCD_write(0x61); // a
LCD_write(0x67); // g
LCD_write(0x65); // e
LCD_write(0x20); // space

// For
LCD_write(0x46); // F
LCD_write(0x6F); // o
LCD_write(0x72); // r

LCD_setCursor(0, 1);

// Thursday
LCD_write(0x54); // T
LCD_write(0x68); // h
LCD_write(0x75); // u
LCD_write(0x72); // r
LCD_write(0x73); // s
LCD_write(0x64); // d
LCD_write(0x61); // a
LCD_write(0x79); // y
LCD_write(0x20); // space

// PM:
LCD_write(0x50); // P
LCD_write(0x4D); // M
LCD_write(0x3A); // :

LCD_setCursor(0, 2);

// ____
LCD_write(0x5F); // _
LCD_write(0x5F); // _
LCD_write(0x5F); // _
LCD_write(0x5F); // _
```

```
  break;

case 10: // Friday AM
  // Input
  LCD_write(0x49); // I
  LCD_write(0x6E); // n
  LCD_write(0x70); // p
  LCD_write(0x75); // u
  LCD_write(0x74); // t
  LCD_write(0x20); // space

  // Dosage
  LCD_write(0x44); // D
  LCD_write(0x6F); // o
  LCD_write(0x73); // s
  LCD_write(0x61); // a
  LCD_write(0x67); // g
  LCD_write(0x65); // e
  LCD_write(0x20); // space

  // For
  LCD_write(0x46); // F
  LCD_write(0x6F); // o
  LCD_write(0x72); // r

  LCD_setCursor(0, 1);

  // Friday
  LCD_write(0x46); // F
  LCD_write(0x72); // r
  LCD_write(0x69); // i
  LCD_write(0x64); // d
  LCD_write(0x61); // a
  LCD_write(0x79); // y
  LCD_write(0x20); // space

  // AM:
  LCD_write(0x41); // A
  LCD_write(0x4D); // M
  LCD_write(0x3A); // :

  LCD_setCursor(0, 2);

  // ____
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
```

```
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  break;

case 11: // Friday PM
  // Input
  LCD_write(0x49); // I
  LCD_write(0x6E); // n
  LCD_write(0x70); // p
  LCD_write(0x75); // u
  LCD_write(0x74); // t
  LCD_write(0x20); // space

  // Dosage
  LCD_write(0x44); // D
  LCD_write(0x6F); // o
  LCD_write(0x73); // s
  LCD_write(0x61); // a
  LCD_write(0x67); // g
  LCD_write(0x65); // e
  LCD_write(0x20); // space

  // For
  LCD_write(0x46); // F
  LCD_write(0x6F); // o
  LCD_write(0x72); // r

  LCD_setCursor(0, 1);

  // Friday
  LCD_write(0x46); // F
  LCD_write(0x72); // r
  LCD_write(0x69); // i
  LCD_write(0x64); // d
  LCD_write(0x61); // a
  LCD_write(0x79); // y
  LCD_write(0x20); // space

  // PM:
  LCD_write(0x50); // P
  LCD_write(0x4D); // M
  LCD_write(0x3A); // :

  LCD_setCursor(0, 2);

  // ____
```

```
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  break;

case 12: // Saturday AM
  // Input
  LCD_write(0x49); // I
  LCD_write(0x6E); // n
  LCD_write(0x70); // p
  LCD_write(0x75); // u
  LCD_write(0x74); // t
  LCD_write(0x20); // space

  // Dosage
  LCD_write(0x44); // D
  LCD_write(0x6F); // o
  LCD_write(0x73); // s
  LCD_write(0x61); // a
  LCD_write(0x67); // g
  LCD_write(0x65); // e
  LCD_write(0x20); // space

  // For
  LCD_write(0x46); // F
  LCD_write(0x6F); // o
  LCD_write(0x72); // r

  LCD_setCursor(0, 1);

  // Saturday
  LCD_write(0x53); // S
  LCD_write(0x61); // a
  LCD_write(0x74); // t
  LCD_write(0x75); // u
  LCD_write(0x72); // r
  LCD_write(0x64); // d
  LCD_write(0x61); // a
  LCD_write(0x79); // y
  LCD_write(0x20); // space

  // AM:
  LCD_write(0x41); // A
  LCD_write(0x4D); // M
  LCD_write(0x3A); // :
```

```
  LCD_setCursor(0, 2);

  // ____
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  LCD_write(0x5F); // _
  break;

case 13: // Saturday PM
  // Input
  LCD_write(0x49); // I
  LCD_write(0x6E); // n
  LCD_write(0x70); // p
  LCD_write(0x75); // u
  LCD_write(0x74); // t
  LCD_write(0x20); // space

  // Dosage
  LCD_write(0x44); // D
  LCD_write(0x6F); // o
  LCD_write(0x73); // s
  LCD_write(0x61); // a
  LCD_write(0x67); // g
  LCD_write(0x65); // e
  LCD_write(0x20); // space

  // For
  LCD_write(0x46); // F
  LCD_write(0x6F); // o
  LCD_write(0x72); // r

  LCD_setCursor(0, 1);

  // Saturday
  LCD_write(0x53); // S
  LCD_write(0x61); // a
  LCD_write(0x74); // t
  LCD_write(0x75); // u
  LCD_write(0x72); // r
  LCD_write(0x64); // d
  LCD_write(0x61); // a
  LCD_write(0x79); // y
  LCD_write(0x20); // space
```

```
    // PM:
    LCD_write(0x50); // P
    LCD_write(0x4D); // M
    LCD_write(0x3A); // :

    LCD_setCursor(0, 2);

    // ____
    LCD_write(0x5F); // _
    LCD_write(0x5F); // _
    LCD_write(0x5F); // _
    LCD_write(0x5F); // _
    break;
  }
}

void UI_twiEnable(int option) // Enables and disables UI TWI line
{
  if(option == 1) // Disables
  {
    nrf_drv_twi_disable(&UI_twi);
  }else if(option == 0) // Enables
  {
    nrf_drv_twi_enable(&UI_twi);
  }
```

Appendix D: ULSS Code

```
#include <Wire.h>
#include <Servo.h>

byte I2C_OnOff;

// Sorting system and data
bool startSort = false;
bool endSort = false;
int sortData[6] = {0, 0, 0, 0, 0, 0}; // Amount of whole pills to dispense per slice
int sortDataCut[6] = {0, 0, 0, 0, 0, 0}; // Amount pills to cut and dispense per slice
int hasCutPill[6] = {0, 0, 0, 0, 0, 0}; // Amount of pills already cut per slice

// Define pin connections & motor's steps per revolution
const int dirPin = 2;   //for bottom small pill holder
const int stepPin = 3;  //for bottom small pill holder
const int dirPin2  = 4;  //for top big pill holder
const int stepPin2 = 5;  //for top big pill holder
const int stepsPerRevolution = 800;
const int stepsPerSlice = 127;
int xtra = 0;
int steps = 0;

// Defining PWM for ESC
Servo ESC;
#define MIN_PULSE_LENGTH 1000 // Minimum pulse length in µs
#define MAX_PULSE_LENGTH 2000 // Maximum pulse length in µs
#define SET_PULSE_LENGTH 1700 // sets cutting speed in µs

void setup()
{
  // Disables stepper motors to conserve power and prevent jerking from noise generated by
Arduino
  pinMode(12, OUTPUT);
  digitalWrite(12, LOW);
  pinMode(13, OUTPUT);
  digitalWrite(13, LOW);

  delay(1000);

  Serial.begin(9600);

  ESC.attach(11, MIN_PULSE_LENGTH, MAX_PULSE_LENGTH);

  pinMode(8, OUTPUT); // Controls direction
  pinMode(9, OUTPUT); // Controls power
```

```
  // Enable quarter micro stepping
  pinMode(6, OUTPUT);
  digitalWrite(6, HIGH);
  pinMode(7, OUTPUT);
  digitalWrite(7, HIGH);

  pinMode(dirPin,   OUTPUT);
  digitalWrite(dirPin, LOW);
  pinMode(dirPin2,  OUTPUT);
  digitalWrite(dirPin2, LOW);
  pinMode(stepPin,  OUTPUT);
  digitalWrite(stepPin, LOW);
  pinMode(stepPin2, OUTPUT);
  digitalWrite(stepPin2, LOW);

  calibrateCutter();

  digitalWrite(12, HIGH);
  digitalWrite(13, HIGH);

  // Set motor direction clockwise
  digitalWrite(dirPin2, HIGH);

  // calibrates main pill holder
  for(int x = 0; x < 5; x++)
  {
    digitalWrite(stepPin2, HIGH);
    delayMicroseconds(2000);
    digitalWrite(stepPin2, LOW);
    delayMicroseconds(2000);
  }
  delay(1000); // Wait a second

  Wire.begin(1); // Starting TWI for arduino with address 0x1

  Serial.println("Ready to receive");

  Wire.onReceive(ULSS); // Setting up recieving event; this is a function that will be called when
the nRF51 sends data to the Arduino

  Wire.onRequest(confirm); // Seting up request event; this is a function that will be called when
the nRF51 requests data from the Arduino
}

void loop()
```

```
{
 if(startSort == true) // Determines whether to start the sorting or not.
 {
  for(int s = 0; s <= 5; s++)
  {
   // Sorting code //

   if(s == 0) // For position 1
   {
    if(sortData[0] == 0)
    {
     // Do nothing
    }else
    {
     for(int j = 1; j <= sortData[s]; j ++) // Dispenses pills j for a total of sortData[s]
     {
      Serial.println("sorting");
      shakeDrum();
      digitalWrite(13, LOW); // Turns off half-pill stepper to conserve power
      delay(100);
      rotateDrum(0);
      digitalWrite(13, HIGH); // Turns half-pill stepper back on
     }
    }

    if(sortDataCut[0] == 1 && hasCutPill[0] == 0) // Will cut a new pill
    {
     shakeDrum();
     digitalWrite(13, LOW); // Turns off half-pill stepper to conserve power
     delay(100);
     activateCutter(1);
     rotateDrum(1);
     activateCutter(0);
     digitalWrite(13, HIGH); // Turns half-pill stepper back on

     hasCutPill[0] = 1;
    }else if(sortDataCut[0] == 1 && hasCutPill[0] == 1) // Will dispense a cut half
    {
     // Open and close servo door

     hasCutPill[0] = 0;
    }else
    {
     // Do nothing
    }
   }else
```

```
  {
   // Returns both holders to next position
   rotateTopDrumDeg();
   rotateBottomDrumDeg();

   if(sortData[s] == 0)
   {
    // Do nothing
   }else
   {
    for(int j = 1; j <= sortData[s]; j ++) // Dispenses pills j for a total of sortData[s]
    {
     Serial.println("sorting");
     shakeDrum();
     digitalWrite(13, LOW); // Turns off half-pill stepper to conserve power
     delay(100);
     rotateDrum(0);
     digitalWrite(13, HIGH);
    }
   }

   if(sortDataCut[s] == 1 && hasCutPill[s] == 0) // Will dispense a cut half
   {
    shakeDrum();
    digitalWrite(13, LOW); // Turns off half-pill stepper to conserve power
    delay(100);
    activateCutter(1);
    rotateDrum(1);
    activateCutter(0);
    digitalWrite(13, HIGH);

    hasCutPill[s] = 1;
   }else if(sortDataCut[s] == 1 && hasCutPill[s] == 1) // Will dispense a cut half
   {
    // Open and close servo door

    hasCutPill[s] = 0;
   }else
   {
    // Do nothing
   }
  }
}

// Returns both holders back to starting position
rotateBackToHomeTop();
```

```
    rotateBackToHomeBottom();

    startSort = false; // end sorting

    endSort = true; // send confirmation
   }else
   {
    // Do nothing //
   }
}

void ULSS(int nBytes) // Parses data received from master
{
  //Serial.print("Recieved data: ");

  Serial.println("recieved signal");

  int masterData = Wire.read(); // Reads incoming data

  int slice = (masterData & 0b00000111); // Parses slice number
  int pillNum = (masterData & 0b01111000) >> 3; // Parses how many pills to dispense
  int cut = (masterData & 0b10000000) >> 7; // Parses whether to cut pill or not

  sortData[slice] = pillNum;
  sortDataCut[slice] = cut;

  if(slice == 5) // Starts at 0 so 5 is the last slice to be recieved. This then enables the flag
startSort to start the sorting process in the main function.
  {
    startSort = true;
  }

  // Test code to ensure data is being received
  /*Serial.println(masterData, BIN);

  Serial.print("Slice number: ");

  Serial.println(slice, DEC);

  Serial.print("Number of pills to sort: ");

  Serial.println(pillNum, DEC);

  if(cut == 1)
  {
    Serial.println("Cut pill.");
```

```
  }
  else if(cut == 0)
  {
   Serial.println("Don't cut pill.");
  }*/

  while(Wire.available())
  {
   Wire.read();
  }
  return;
}

void confirm() // Sends a confirmation bit to master to tell it sorting is done.
{
  if(endSort == true)
  {
   Wire.write(0x1);

   //Serial.println("Confirmation sent.");

   endSort = false;

   delay(500);

   Wire.write(0x0);
  }else
  {
   Wire.write(0x2);

   endSort = false;
  }
}

void shakeDrum() // Shakes pill holder to dislodge pills
{
  // Set motor direction counter-clockwise
  for(int i = 0; i <= 20; i++)
  {
   digitalWrite(dirPin2, LOW);
   // Spin motor slowly
   for(int x = 0; x < 25; x++)
   {
    digitalWrite(stepPin2, HIGH);
    delayMicroseconds(300);
    digitalWrite(stepPin2, LOW);
```

```
    delayMicroseconds(300);
   }
   delay(150); // Wait a second

   digitalWrite(dirPin2, HIGH);

   // Spin motor slowly
   for(int x = 0; x < 25; x++)
   {
    digitalWrite(stepPin2, HIGH);
    delayMicroseconds(300);
    digitalWrite(stepPin2, LOW);
    delayMicroseconds(300);
   }
   delay(150); // Wait a second
  }
}

void rotateTopDrumDeg() // Rotates main pillholder specified amount of degrees determined by the steps
{
  // Set motor direction counter-clockwise
  digitalWrite(dirPin2, LOW);

  // Spin motor slowly
  for(int x = 0; x < (127+xtra); x++) // Rotates stepper motor 127 steps or 54 degrees. The xtra bit is to ensure the stepper doesnt lag behind and that the slices are aligned properly
  {
   digitalWrite(stepPin2, HIGH);
   delayMicroseconds(2000);
   digitalWrite(stepPin2, LOW);
   delayMicroseconds(2000);
  }
  steps = 127 + xtra + steps; // The amount of steps needed to rotate back to starting position
  xtra = xtra + 1;
  //steps = 127 + xtra + steps;
  delay(1000); // Wait a second
}

void rotateBottomDrumDeg() // Rotates cut half pillholder specified amount of degrees determined by the steps
{
  // Set motor direction clockwise
  digitalWrite(dirPin, LOW);

  // Spin motor slowly
```

```
  for(int x = 0; x < 133; x++) // Rotates stepper motor 133 steps or 60 degrees
  {
   digitalWrite(stepPin, HIGH);
   delayMicroseconds(2000);
   digitalWrite(stepPin, LOW);
   delayMicroseconds(2000);
  }
  delay(1000); // Wait a second
}

void rotateBackToHomeTop() // Rotates main pill holder back to starting position
{
  // Set motor direction clockwise
  digitalWrite(dirPin2, HIGH);

  // Spin motor slowly
  for(int x = 0; x < (steps); x++)
  {
   digitalWrite(stepPin2, HIGH);
   delayMicroseconds(2000);
   digitalWrite(stepPin2, LOW);
   delayMicroseconds(2000);
  }
  xtra = 0;
  steps = 0;
  delay(1000); // Wait a second
}

void rotateBackToHomeBottom() // Rotates main pill holder back to starting position
{
  // Set motor direction clockwise
  digitalWrite(dirPin, HIGH);

  // Spin motor slowly
  for(int x = 0; x < 133*5; x++)
  {
   digitalWrite(stepPin, HIGH);
   delayMicroseconds(2000);
   digitalWrite(stepPin, LOW);
   delayMicroseconds(2000);
  }
  delay(1000); // Wait a second
}

void rotateDrum(int dir) // Rotates pill drum. 0 Rotates clockwise and 1 rotates counter-
clockwise
```

```
{
  if(dir == 0)
  {
    digitalWrite(8, LOW); // clockwise rotation
    delay(100);
    digitalWrite(9, HIGH); // Turn motor on
    delay(2200);
    digitalWrite(9, LOW); // Turn motor off
    delay(100);
  }else if(dir == 1)
  {
    digitalWrite(8, HIGH); // counter-clockwise rotation
    delay(100);
    digitalWrite(9, HIGH); // Turn motor on
    delay(2200);
    digitalWrite(9, LOW); // Turn motor off
    delay(100);
  }
}

void calibrateCutter() // Calibrates cutter motor ESC
{
  // Calibrate
  delay(5000);
  ESC.write(MAX_PULSE_LENGTH);
  delay(2000);
  ESC.write(MIN_PULSE_LENGTH);
  delay(2000);
  /*ESC.writeMicroseconds(MAX_PULSE_LENGTH);
  delay(4000);
  ESC.writeMicroseconds(MIN_PULSE_LENGTH);
  delay(5000);*/
}

void activateCutter(int pwr) // Activate cutter motor. 1 turns cutter motor on and 0 turns it off
{
  // Run cutter
  if(pwr == 1)
  {
    for (int i = MIN_PULSE_LENGTH; i <= SET_PULSE_LENGTH; i += 10) // Gradually
increases speed of the motor until desired speed of 20,000 RPM is reached.
    {
      Serial.print("Pulse length = ");
      Serial.println(i);

      ESC.writeMicroseconds(i);
```

```
      delay(200);
    }
    //delay(4000);
  }else if(pwr == 0)
  {
    ESC.writeMicroseconds(MIN_PULSE_LENGTH);
  }
}
```

Appendix E: LLSS Code

```
/*
set up two arrays for each position
motor1 (x) [Sun Mon Tus Wed Thur Fri Sat]
motor2 (y) [AM PM Final]

15 total postions

final is 1200 steps per revolution  It did 6 forwards at 200 steps per revolution to reach the end

*/

#include "Wire.h"

// Define pin connections & motor's steps per revolution
const int stepsPerRevolution = 200*4;

const int dirPin = 2;   //for z-axis
const int stepPin = 3;  //for z-axis

const int dirPin2  = 4;  //for x-axis
const int stepPin2 = 5;  //for x-axis

// Enables microstepping for the steppers
const int zMicro = 6;
const int xMicro = 7;

// Enables or disables stepper motors
const int zEn = 8;
const int xEn = 9;

int zeroed = 0; // Flag to record when the system has already been zeroed.

int sortNum = 0;
int zPos = 0; // Determines whether the z axis is in the AM or PM position with 0 equalling AM
and 1 equalling PM

bool endSort = false;
bool sort = false;

void setup()
{
  // Disables stepper motors to conserve power and prevent jerking from noise generated by
Arduino
  pinMode(zEn, OUTPUT);
  digitalWrite(zEn, LOW);
```

```
  pinMode(xEn, OUTPUT);
  digitalWrite(xEn, LOW);

  delay(1000);

  Serial.begin(9600);

  // Declare pins as Outputs
  pinMode(stepPin,  OUTPUT);
  pinMode(dirPin,   OUTPUT);
  pinMode(dirPin2,  OUTPUT);
  pinMode(stepPin2, OUTPUT);

  digitalWrite(stepPin, LOW);
  digitalWrite(stepPin2, LOW);
  digitalWrite(dirPin, LOW);
  digitalWrite(dirPin2, LOW);

  pinMode(zMicro,  OUTPUT);
  pinMode(xMicro, OUTPUT);
  digitalWrite(zMicro, HIGH);
  digitalWrite(xMicro, HIGH);

  zeroed = 1; // Enables zeroing to begin

  Wire.begin(2); // Sets up TWI connection with an address of 0x2

  Serial.println("Ready to receive");

  Wire.onReceive(LLSS); // Setting up recieving event; this is a function that will be called when
the nRF51 sends data to the Arduino

  Wire.onRequest(confirm); // Seting up request event; this is a function that will be called when
the nRF51 requests data from the Arduino
}

void loop()
{
  if(zeroed == 1) // Zeroes both x and z axis
  {
    zero();
  }
}

void zero() // Zeroes the positions of the x and z axis stepper systems
{
```

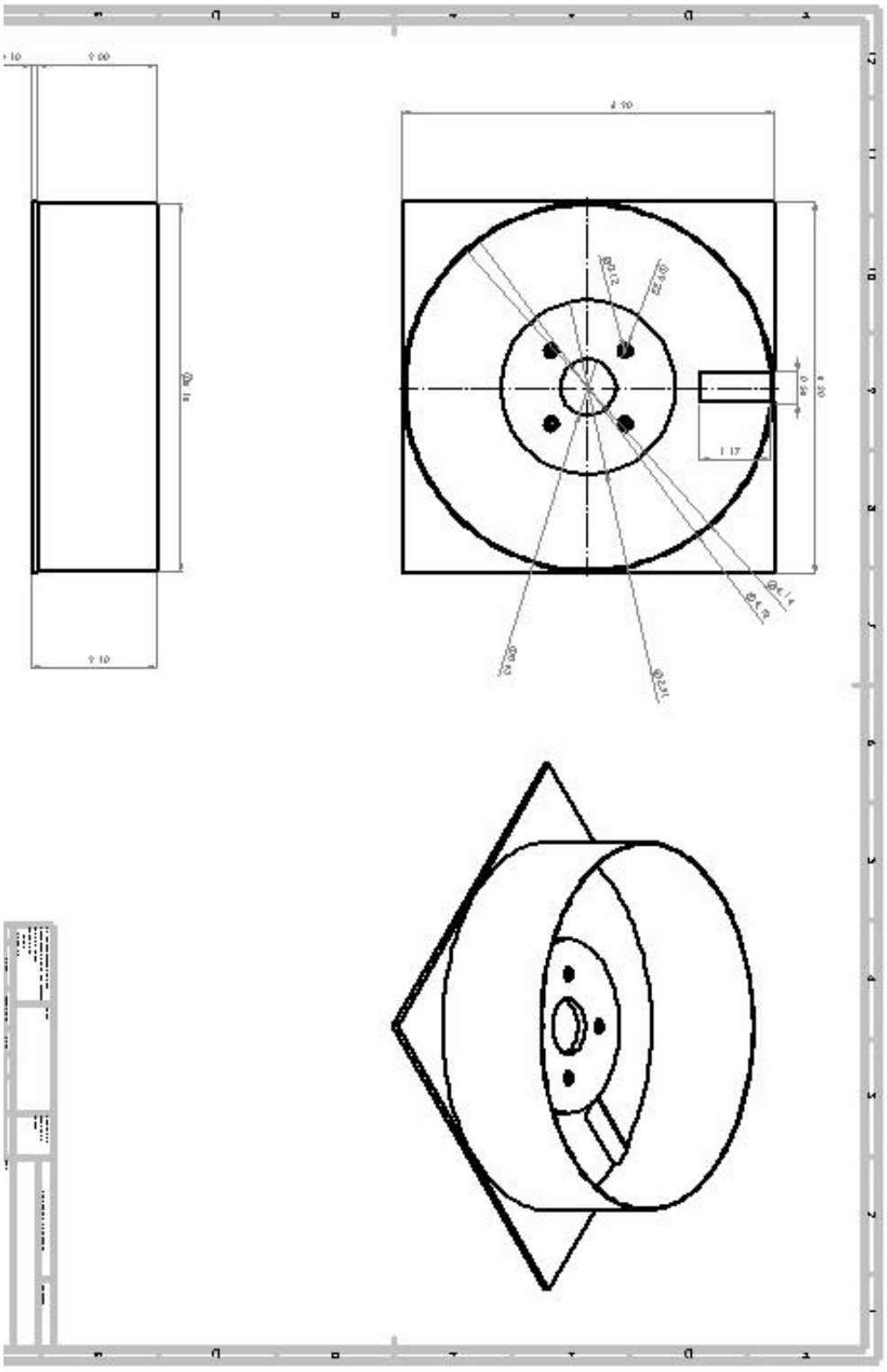
```
  // Activate stepper motors
  digitalWrite(zEn, HIGH);
  digitalWrite(xEn, HIGH);

  delay(100);

  while (analogRead(A1) >= 1000) // While the limit switch is not pressed, the stepper motors
will reverse backwards
  {

      // Set motor direction counterclockwise
      digitalWrite(dirPin, LOW);
      // Spin motor quickly
      for(int x = 0; x < stepsPerRevolution/2; x++)
      {
        digitalWrite(stepPin, HIGH);
        delayMicroseconds(750);
        digitalWrite(stepPin, LOW);
        delayMicroseconds(750);
      }
      delay(1000); // Wait a second
    }

    while (analogRead(A2) >= 1000) // While the limit switch is not pressed, the stepper motors
will reverse backwards
    {

      // Set motor direction counterclockwise
      digitalWrite(dirPin2, LOW);
      // Spin motor quickly
      for(int x = 0; x < stepsPerRevolution/2; x++)
      {
        digitalWrite(stepPin2, HIGH);
        delayMicroseconds(750);
        digitalWrite(stepPin2, LOW);
        delayMicroseconds(750);
      }
      delay(1000); // Wait a second
    }

  // Turns off stepper motors to conserve power.
  digitalWrite(zEn, LOW);
  digitalWrite(xEn, LOW);

  zeroed = 0; // Ends zeroing function.
```

```
}

void forwardX() // Makes x axis move forward to the next compartment
{
  // Activate stepper motors
  digitalWrite(zEn, HIGH);
  digitalWrite(xEn, HIGH);

  // Set motor direction clockwise
  digitalWrite(dirPin2, HIGH);

  // Spin motor slowly
  for(int x = 0; x < 2072; x++)
  {
    digitalWrite(stepPin2, HIGH);
    delayMicroseconds(1000);
    digitalWrite(stepPin2, LOW);
    delayMicroseconds(1000);
  }
  delay(1000); // Wait a second

  // Turns off stepper motors to conserve power.
  digitalWrite(zEn, LOW);
  digitalWrite(xEn, LOW);
}

void moveZ(int dir) // Makes z axis move to and from AM and PM positions, 0 moves to AM
position and 1 moves to PM position
{
  // Activate stepper motors
  digitalWrite(zEn, HIGH);
  digitalWrite(xEn, HIGH);

  if(dir == 0) // z axis is in the PM position and will now move to the AM position
  {
    // Set motor direction counterclockwise
    digitalWrite(dirPin, LOW);
    // Spin motor quickly
    for(int x = 0; x < 2750; x++)
    {
      digitalWrite(stepPin, HIGH);
      delayMicroseconds(750);
      digitalWrite(stepPin, LOW);
      delayMicroseconds(750);
    }
```

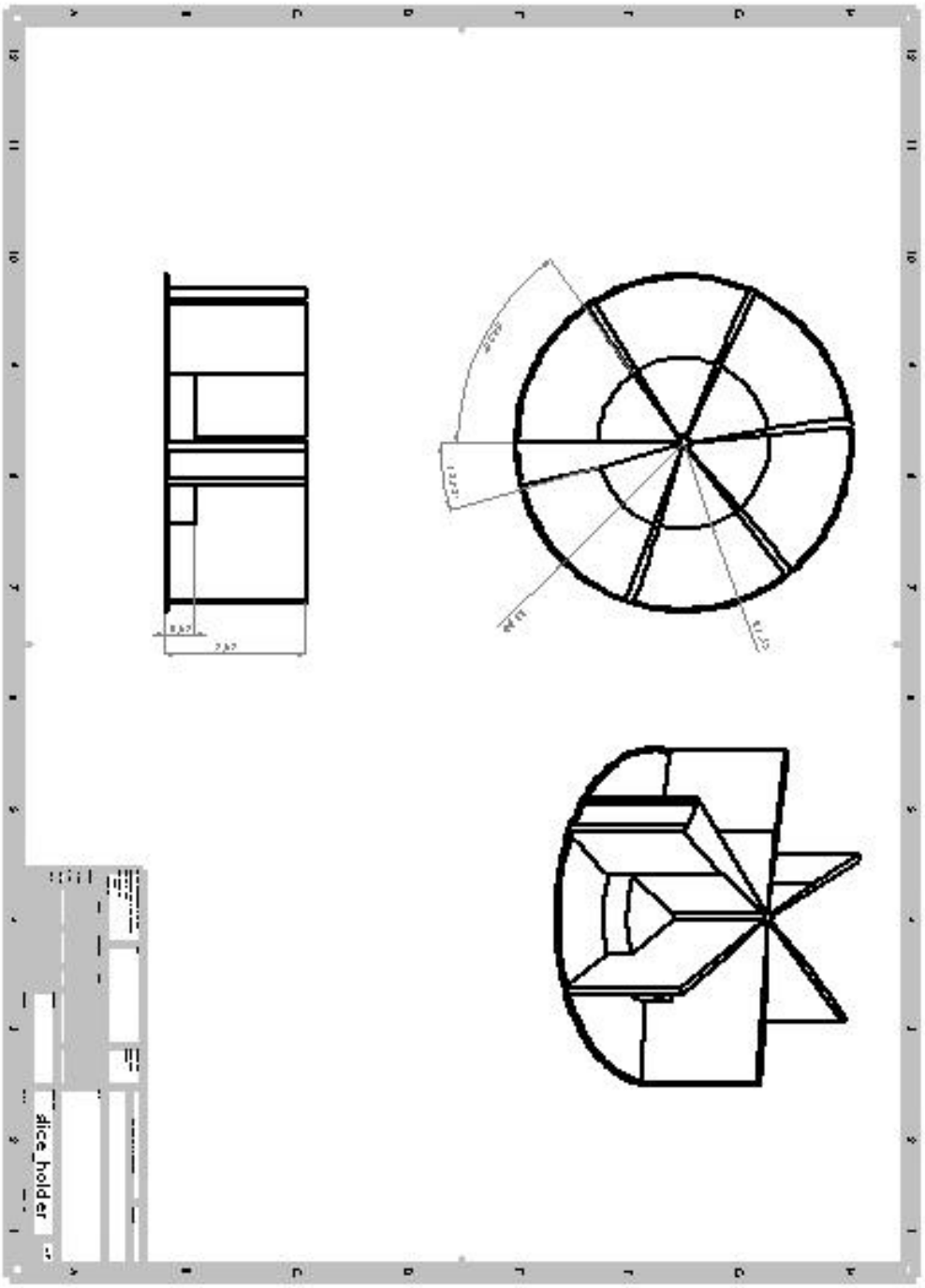
```
   delay(1000); // Wait a second
  }else if(dir == 1) // z axis is in the AM position and will now move to the PM position
  {
   // Set motor direction clockwise
   digitalWrite(dirPin, HIGH);
   // Spin motor quickly
   for(int x = 0; x < 2750; x++)
   {
    digitalWrite(stepPin, HIGH);
    delayMicroseconds(750);
    digitalWrite(stepPin, LOW);
    delayMicroseconds(750);
   }
   delay(1000); // Wait a second
  }

  // Turns off stepper motors to conserve power.
  digitalWrite(zEn, LOW);
  digitalWrite(xEn, LOW);
}

void LLSS(int nBytes) // Moves x and z axis. Positions are premapped and are sequential starting
at Sunday AM position and ending at the Saturday PM position
{
  Serial.println("recieved signal");

  if(sortNum == 0) // Already at first position, no need to move
  {
   // do nothing
  }else
  {
   if((sortNum % 2) == 0 && zPos == 1) // In PM position, so now needs to move to the AM
position and next day
   {
    moveZ(0);
    zPos = 0;

    forwardX();
   }else if((sortNum % 2) == 0) // Needs to move to the next day
   {
    forwardX();
   }else if((sortNum % 2) != 0) // Needs to move to the next time of day
   {
    if(zPos == 0)
    {
     moveZ(1);
```

```
      zPos = 1;
     }else if(zPos == 1)
     {
      moveZ(0);
      zPos = 0;
     }
    }
  }

  sortNum = sortNum + 1;

  endSort = true;

  while(Wire.available())
  {
    Wire.read();
  }
  return;
}

void confirm() // Sends confrimation bit to master to tell it the pill planner is in position
{
  Serial.println("recieved confirm signal");
  if(endSort == true)
  {
    Serial.println("sent signal");

    Wire.write(0x1);

    //Serial.println("Confirmation sent.");

    endSort = false;

  }else
  {
    //Wire.write(0x2);

    //endSort = false;
  }
}
```

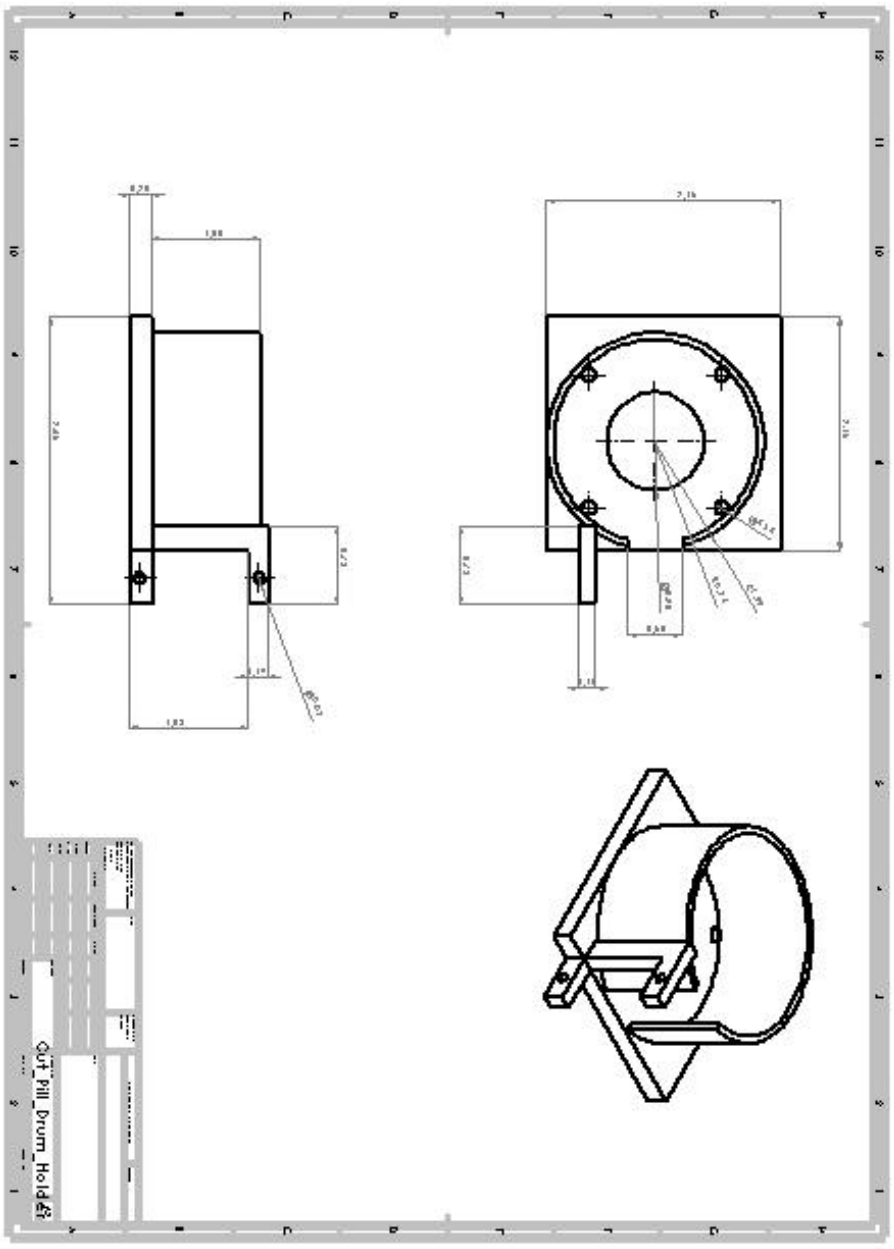
Appendix F: SolidWorks Drawings



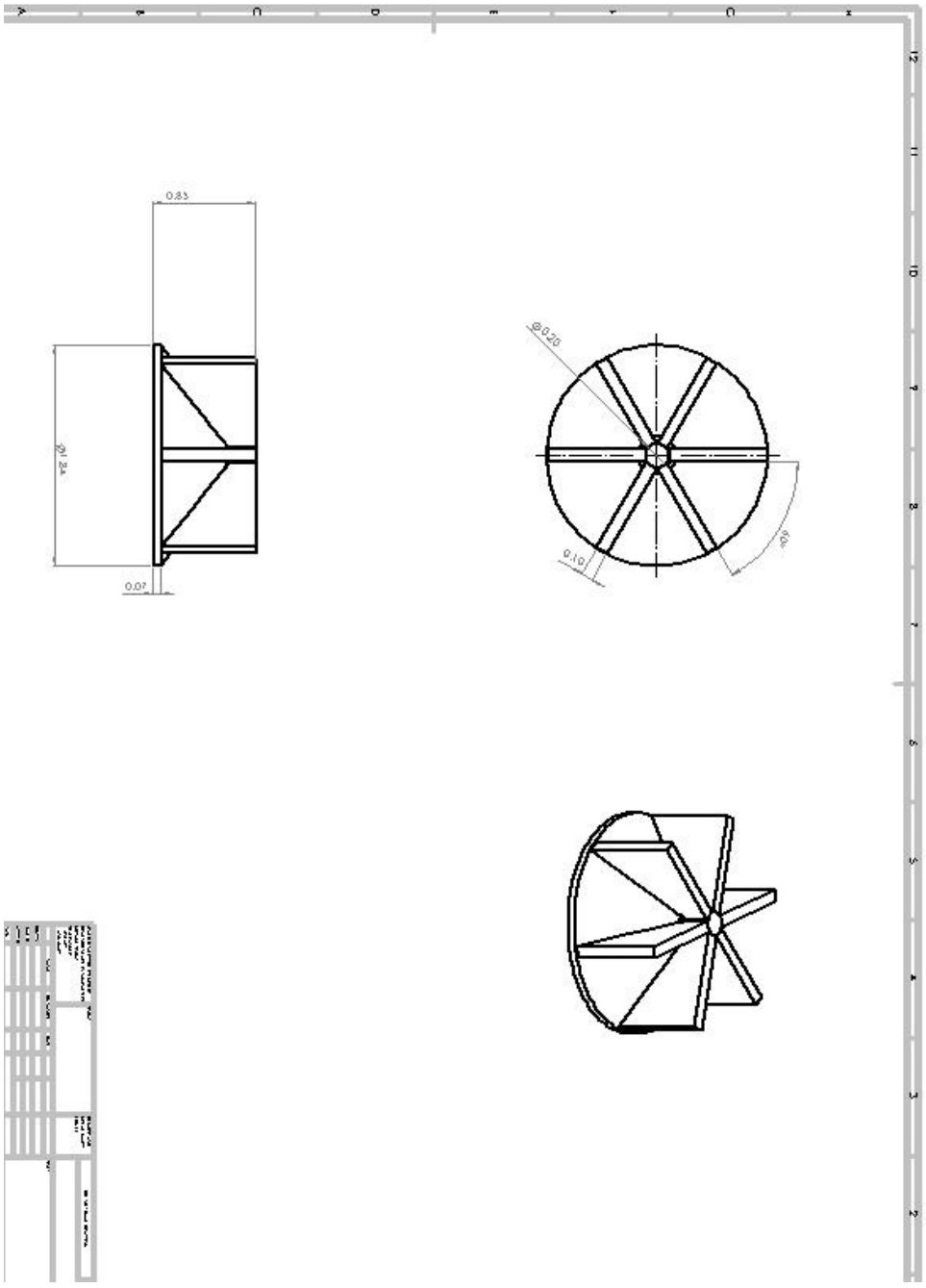
Appendix F.1: Main Pill Holder



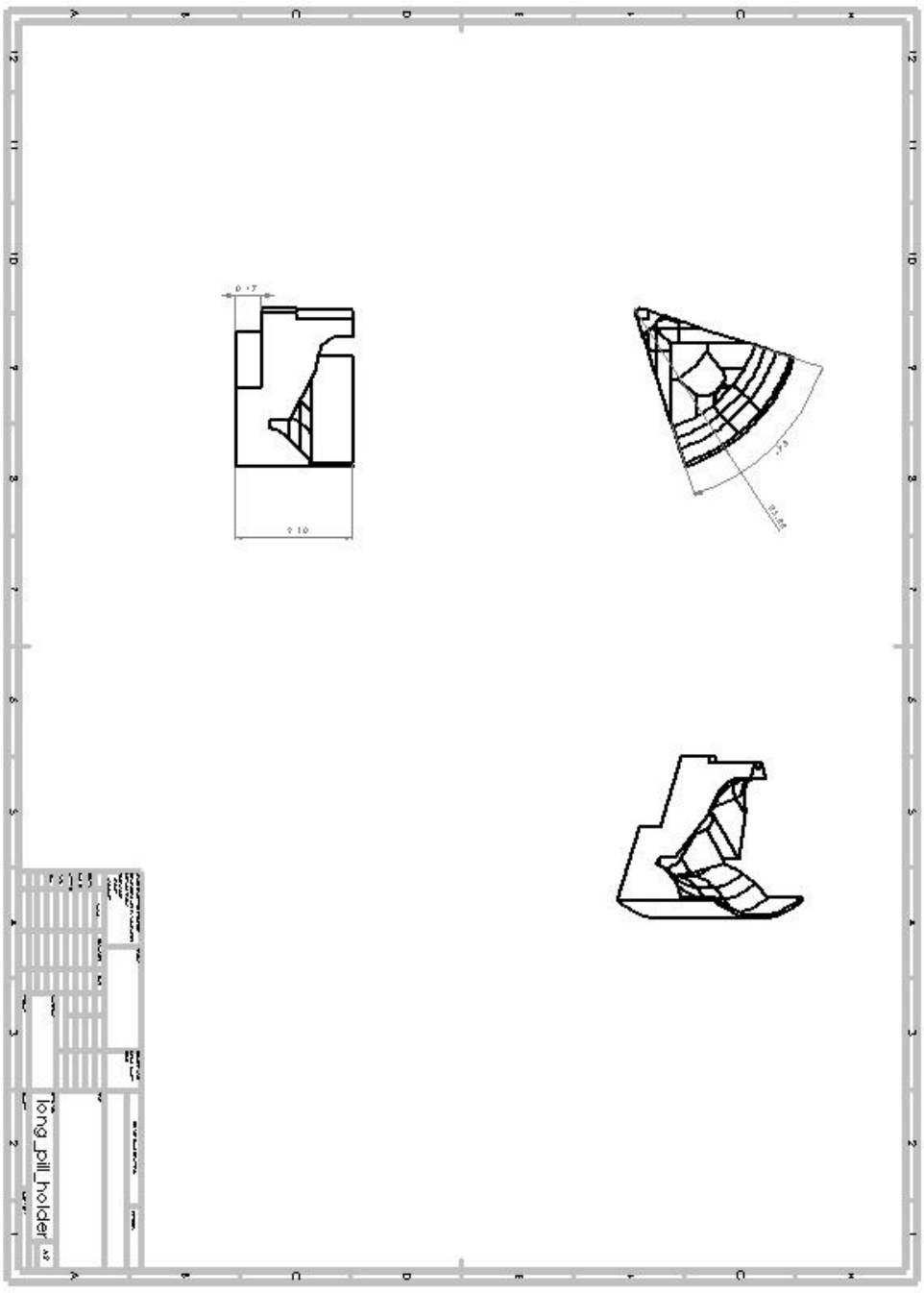
Appendix F.2: Slice Holder



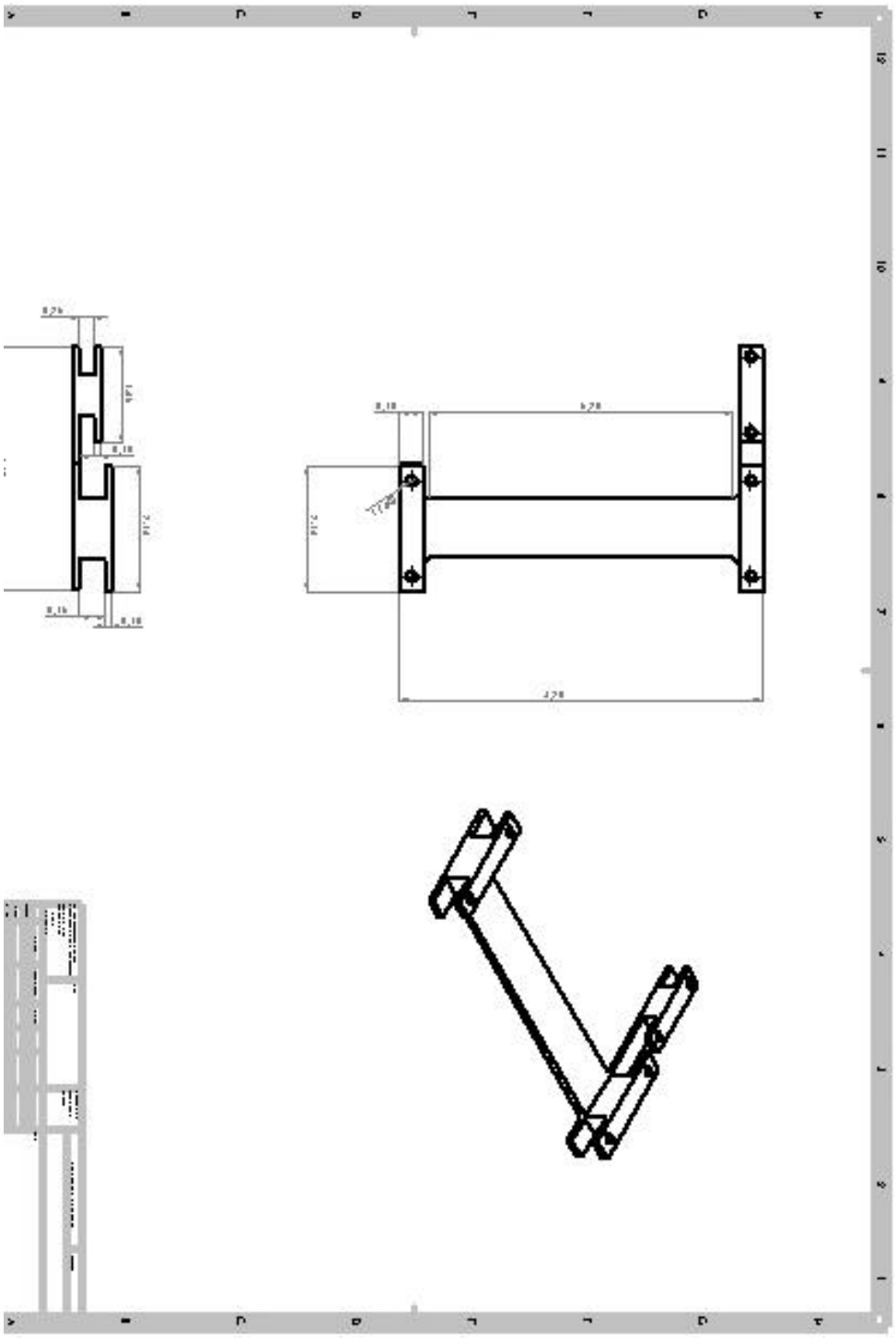
Appendix F.3: Half Pill Holder



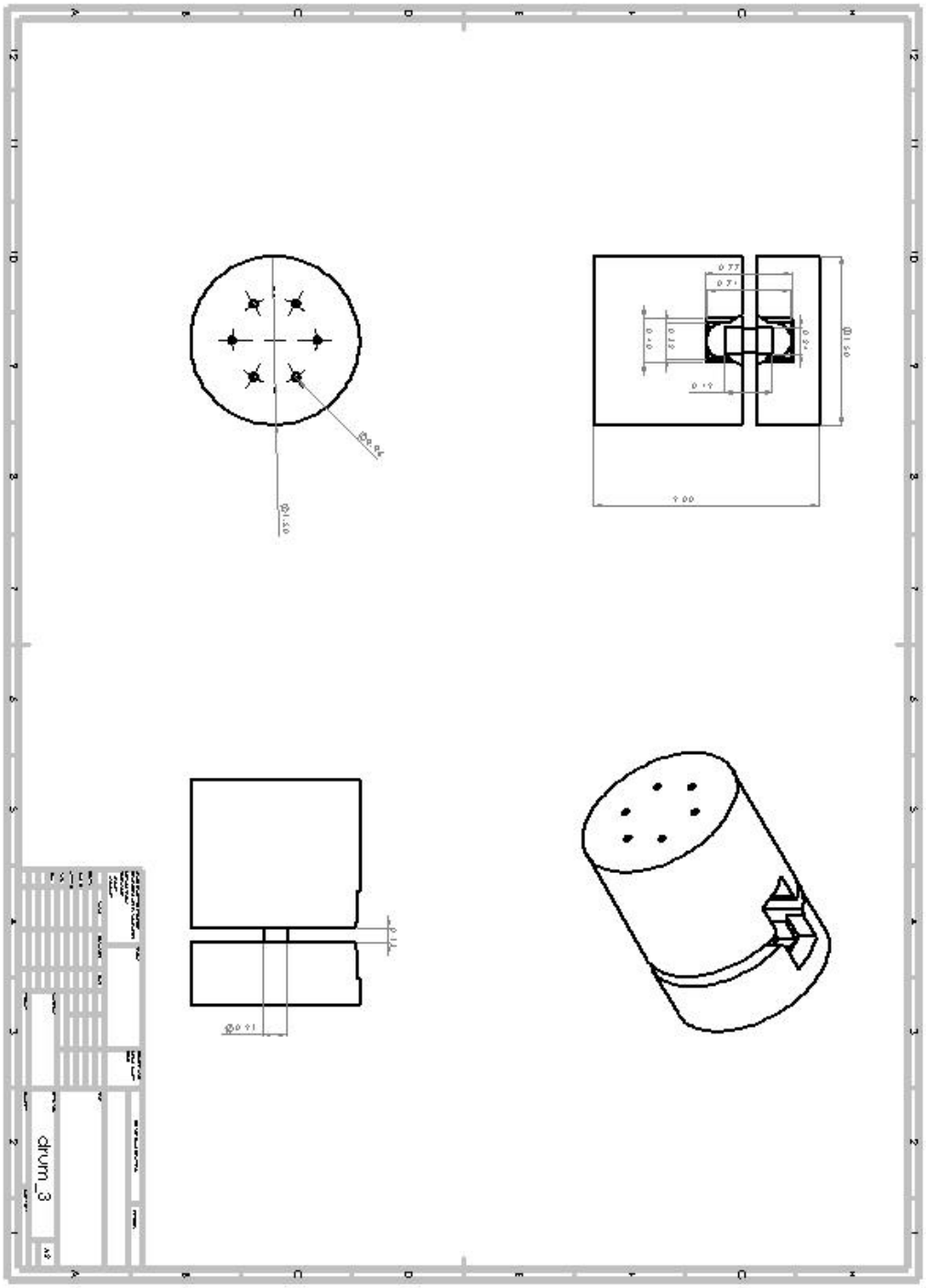
Appendix F.4: Half Pill Slices



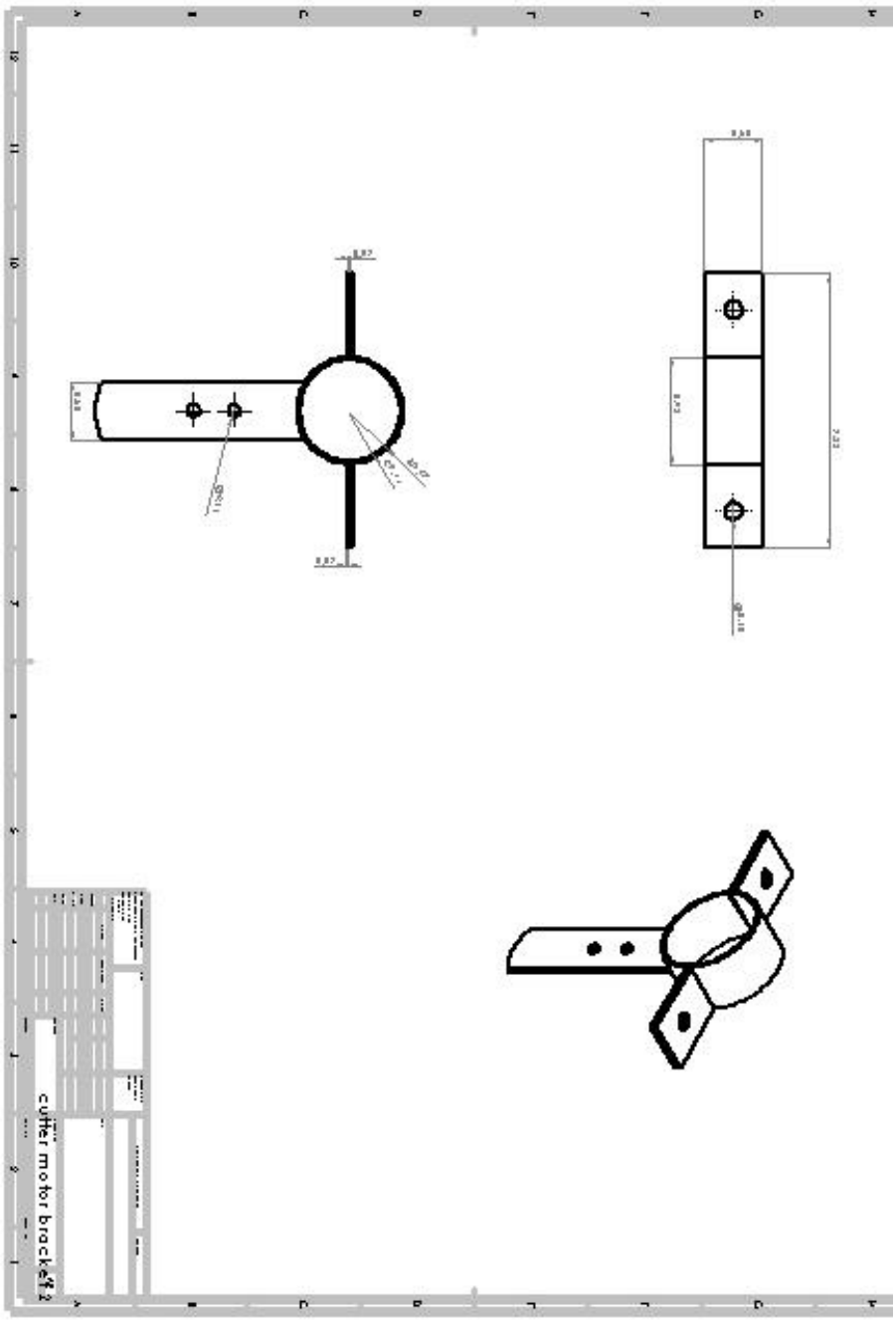
Appendix F.5: Pill Holder Slice



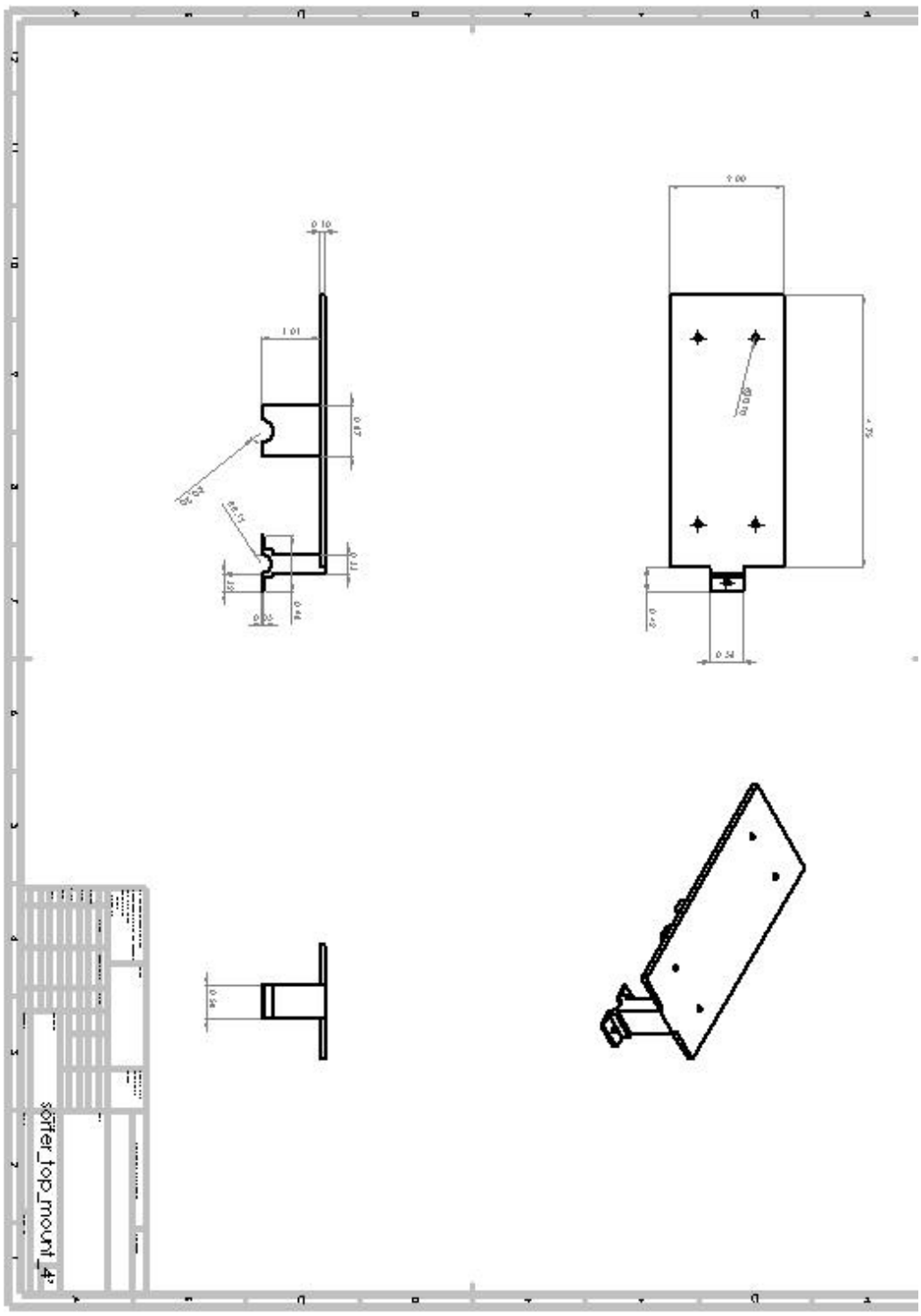
Appendix F.6: LLSS Alignment



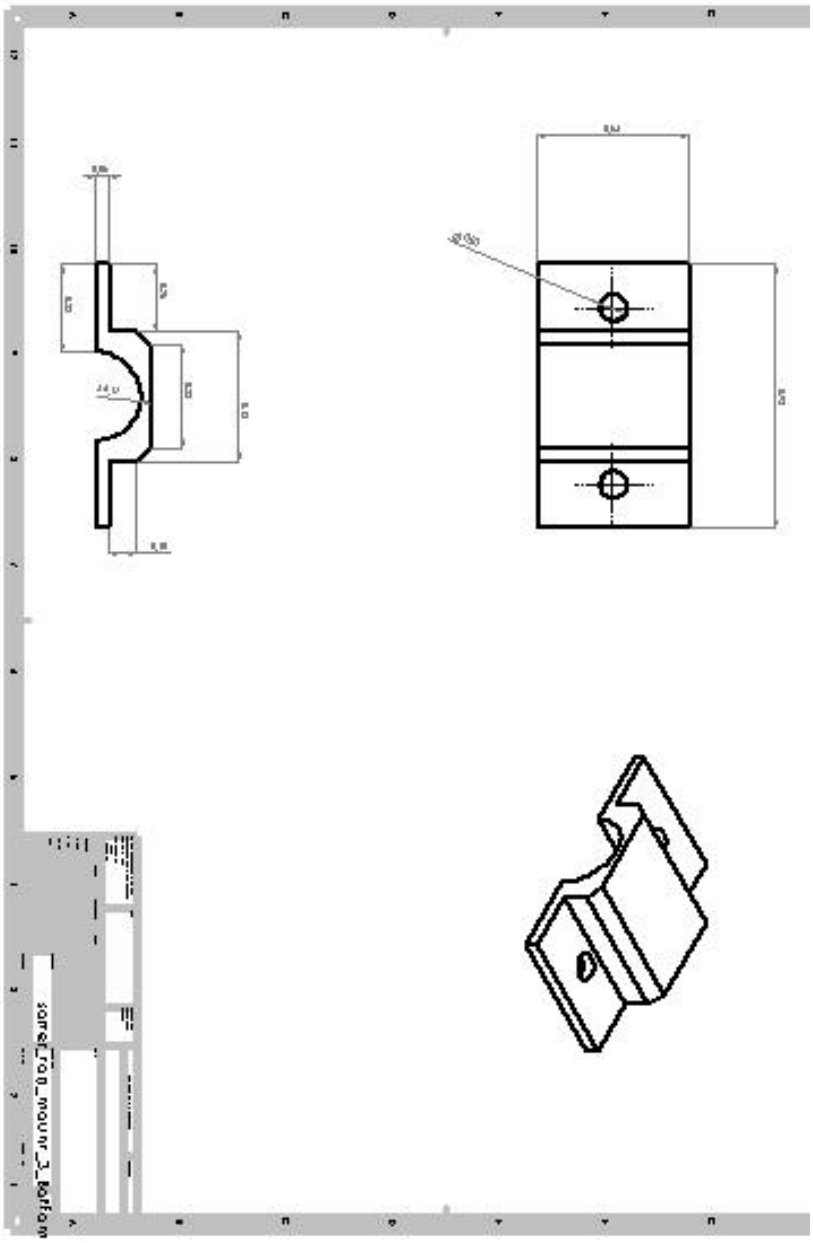
Appendix F.7: Pill Drum



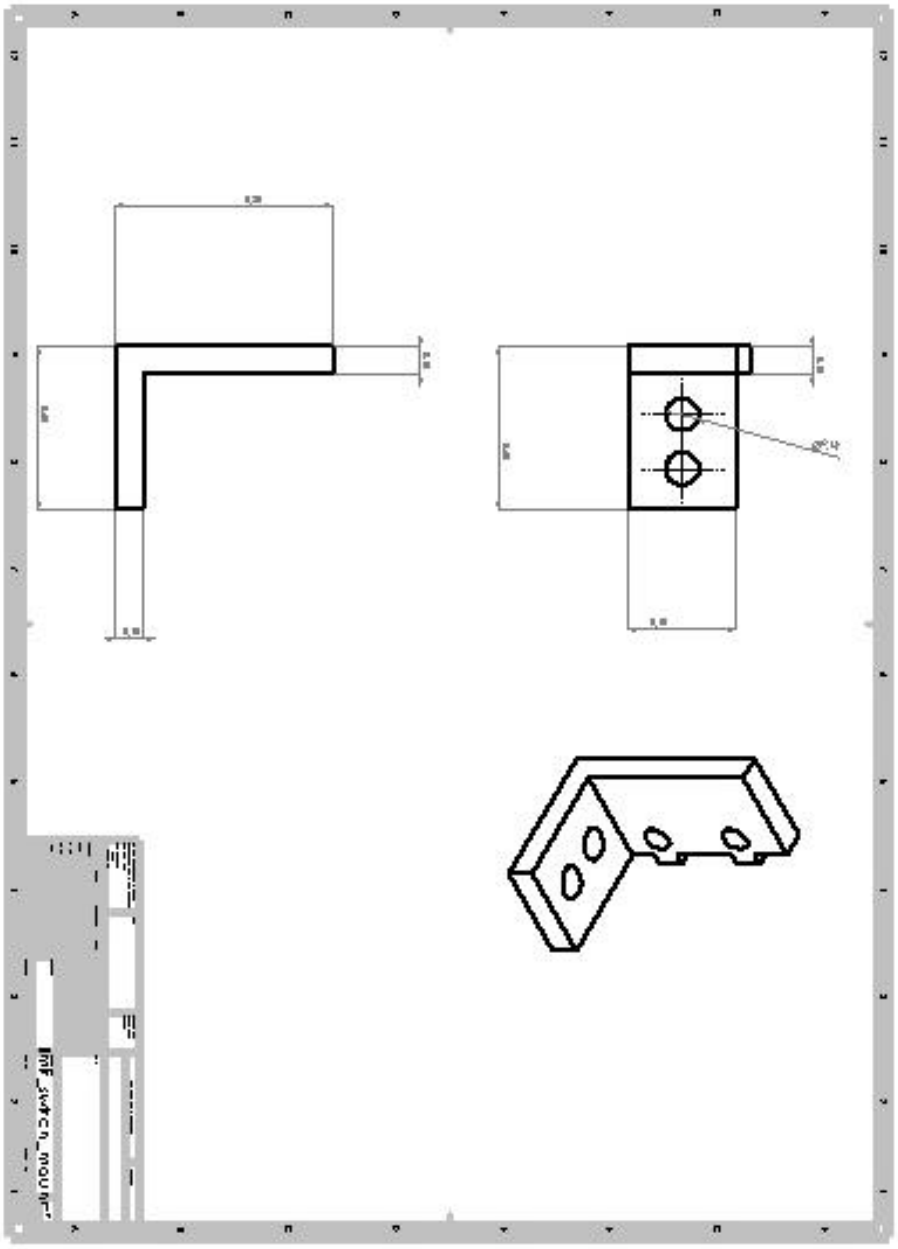
Appendix F.8: Cutter Motor Mount



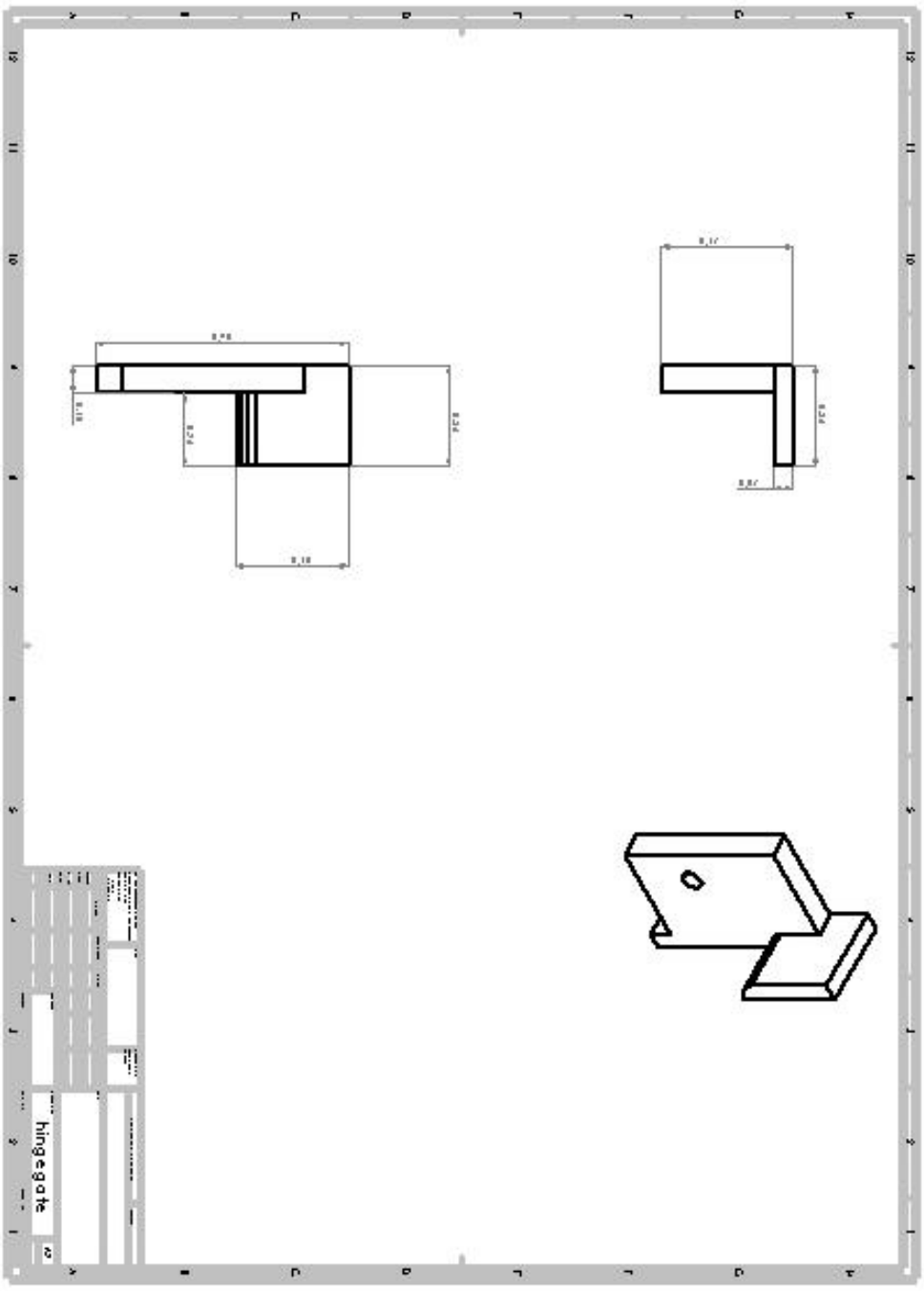
Appendix F.9: LLSS Plate Couple



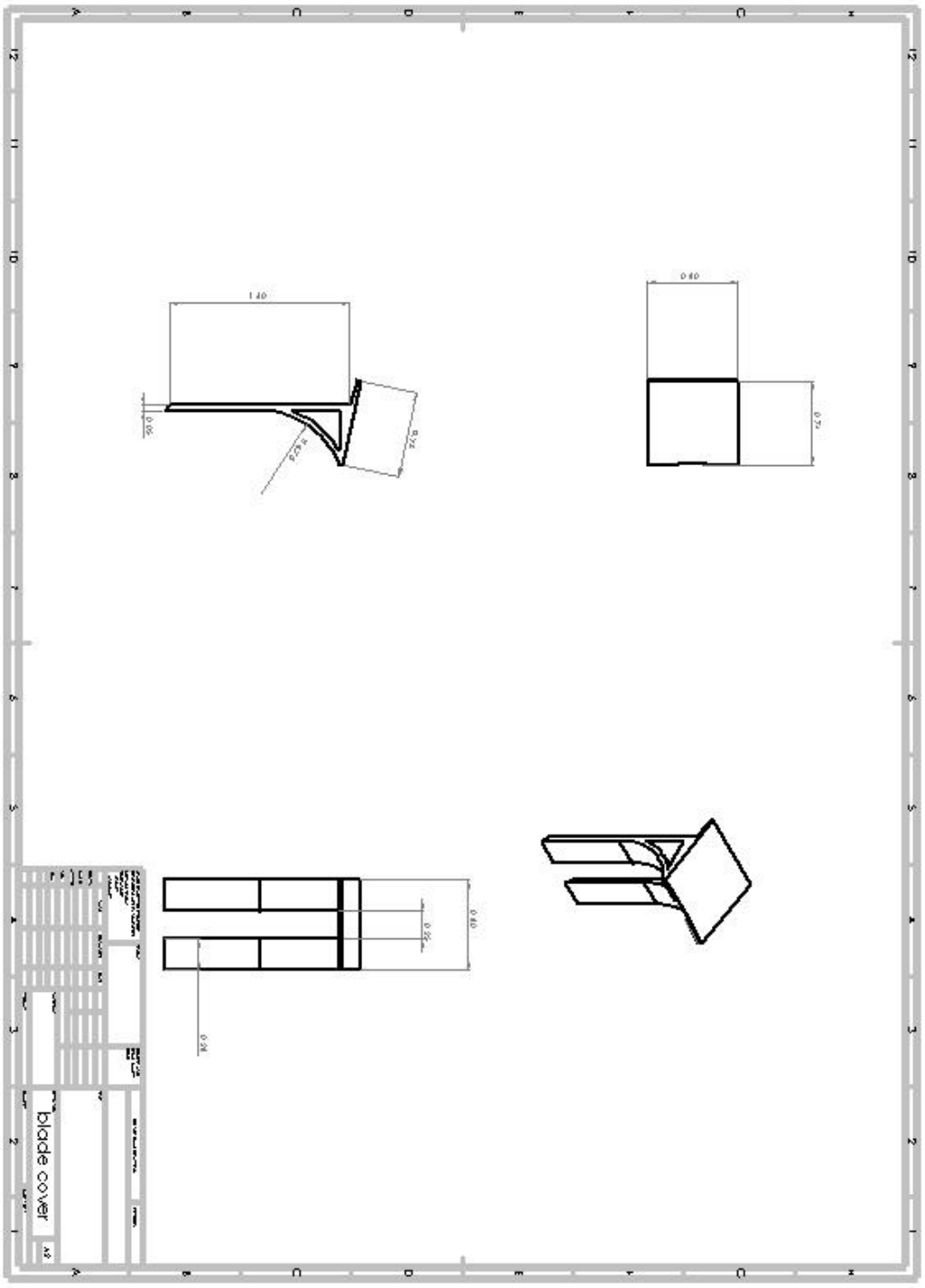
Appendix F.10: LLSS Plate Couple Rail Mount



Appendix F.11: Limit Switch Mount



Appendix F.12: Half Pill Servo Gate



Appendix F.13: Drum Cover

Appendix G: Wiring Diagram