# MTE 121/MTE 100 Final Report
# Pillbot

Group 8-10
Uday Roy - 21026894
Aly Ahmed - 21003462
Ilan Rajpar - 20990873
Bishoy Abdelnour – 21010222

## Contents

Table of Tables ........ i

Table of Figures ........ ii

Summary ........ ii

Introduction ........ 1

Scope ........ 1

Constraints and Criteria ........ 4

Mechanical Design and Implementation ........ 6

New Additions ........ 12

New Modifications Made ........ 17

Software Design and Implementation ........ 20

Verification ........ 32

Project Plan ........ 33

Conclusion ........ 35

Recommendations ........ 36

Mechanical Design ........ 36

Software and Industry Recommendations ........ 37

Bibliography ........ 38

Appendices ........ 39

Appendix A – Checklist Used on Demo Day ........ 39

Appendix B - Source Code ........ 39

## Table of Tables

Table 1: Function Testing ........ 26

Table 2: Initial Project Plan ........ 33

## Table of Figures

Figure 1: Sketch of Initial Design ........ 1
Figure 2: Final Design ........ 2
Figure 3: Pill Container Idea ........ 7
Figure 4: Pill Container Sketch ........ 7
Figure 5: Funnel Versus Ramp ........ 8
Figure 6: Front View of First Dispenser Design ........ 9
Figure 7: Back View of First Dispenser Design ........ 9
Figure 8: Close View of First Dispenser Design ........ 10
Figure 9: Ramp Slot ........ 11
Figure 10: Pills Jammed in Funnel Design ........ 11
Figure 11: Front View of Pill Level Detector ........ 13
Figure 12: Back View of Pill Level Detector ........ 14
Figure 13: Pill Detecter TETRIX Frame ........ 16
Figure 14: Modifications Made to Pill Level Detector ........ 17
Figure 15: New Single File Dispensers ........ 17
Figure 16: Pills Inside Single Pill Dispenser ........ 18
Figure 17: Hot Glued Single File Tube ........ 19
Figure 18: Flowchart Page 1 ........ 20
Figure 19: Flowchart Page 2 ........ 21

## Summary

The main idea of the project is to build an automated pill dispenser that reads information from a file and dispenses a certain number of pills at different times throughout the day. There have been several designs of the dispenser to accommodate the requirements of the project. The final design includes three pill containers and three motors with trapdoors connected to them. The trapdoors are what allow the pills to be dispensed onto a ramp leading to a cup for the user. There is also a linear slider with a color sensor attached to it responsible for detecting the level of the pills to know if any of the containers are running out of pills. The program used to run this pill dispenser consists of eight different functions with different parameters and return types. These functions are what ensure that the pill dispenser is functional and responsive to the user. The only issue with the pill dispenser is the accuracy of the pill dispensing as the mechanical design used to dispense the pills does not guarantee the dispensing of a single pill. This means that the number of pills dispensed will almost never be equal to the number of pills meant to be dispensed as per the text file. The conclusion of the project would be the redesign of the mechanical part to ensure that a singular pill is dispensed every time and the complete pill dispenser is functional enough to broadcasted onto an industrial scope.

## Introduction

There has been a vast increase in diseases and medications as the field of medicine advances. Therefore, it has become increasingly difficult to keep up with the number of pills one has to take to stay healthy and alive. It is also especially difficult for those of old age to keep track of their daily medications as the older they get, the more they must take at different times. Therefore, the group aimed to address this issue by designing a robot that would take this tedious task and make it much simpler. The robotic system the group hopes to develop will accurately and conveniently determine the correct amount of each pill specified by the user beforehand and dispense that amount at given times in the span of a day. This will vastly aid those of older age as it will remove the trouble of organizing, counting, and remembering the different types and quantities of pills every day.

## Scope

Before the final form of the pill dispenser was decided upon, there were different initial designs. This section will go over the different initial designs and how they evolved into the final design, then the dispenser in its final form will be explored in more detail.

This right here is a sketch of the groups first idea for the pill dispensing mechanism:

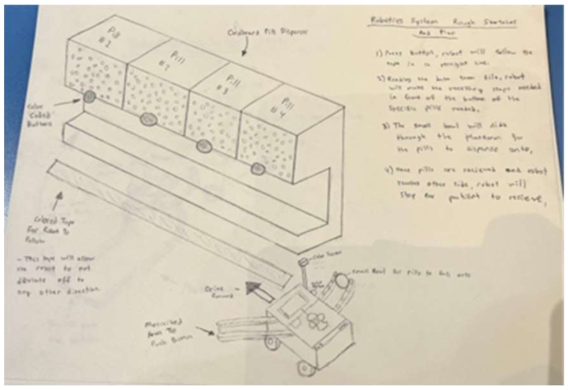

Figure 1: Sketch of Initial Design

It consisted of a structure containing multiple containers, each of which contained a different type of pill. On the front of each container was a button that dispensed exactly one pill. Underneath each container would have been a sticker of a specific color. The robot would have then been mechanically modified so that it has an arm that stretches from its side to click the button, a cup holder, and a color sensor mounted to its side. It then would move in a straight line underneath the containers and scan the color to determine which type of pill needs to be dispensed. The arm would then rise, press the button the number of times necessary to dispense a pill, and then repeat that for each of the containers. This idea was used as it was determined that using motors to control the drop of pills would work much better and faster than a non-motorized mechanism. Therefore, since the dispensing mechanism was being motorized, the need for a robot that moves back and forth to collect pills became obsolete. A mechanism that checks if the pill containers need to be refilled was also added to make the project meet the requirements.

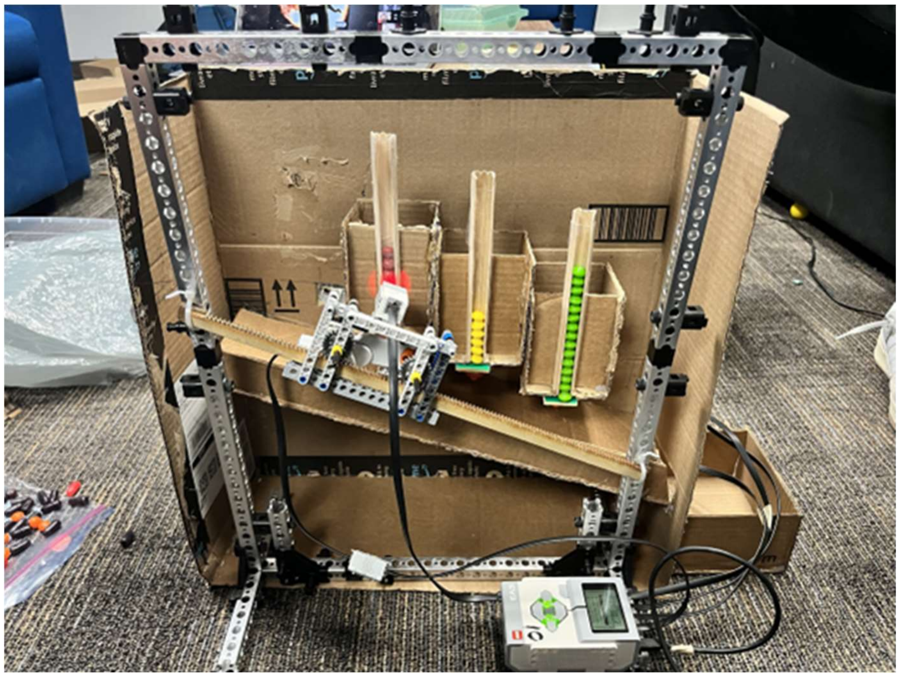
*Figure 2: Final Design*

The pill dispenser that was made for the project performs a multitude of steps and functions to be able to dispense the correct dosage of pills for the user at the correct times during the day, with all the user needing to do is create a text file containing the number of pills of each type of pill for a specific time of the day, and the time intervals between these medicine intakes. Additionally, the user must press the enter button on the robot at the start of the day, when it is time for their first intake of medicine. Now, the functions and steps that the robot will perform specifically will be explained in detail, as well as any changes that were made to these functions.

First, the robot stores into a set of four parallel arrays the amount of each of the four pills the user will take at a particular medicine intake as well as the time intervals between the medicine intakes throughout the day. The initial plan was to use the time of day to indicate exactly the time of the medicine intake and the time that the dispenser should dispense pills for that specific intake, however, there was not a known method of using a clock-in code in RobotC. Additionally, using a timer would fulfill the requirement of using a timer, so it was decided to dispense medicine at separate time intervals relative to other medicine dispenses. It would, though, be preferable to use the clock, so that if the user wakes up late and does not press the button at the correct time of day in the morning, the medicine is still dispensed at the correct time.

There are three motors mounted underneath three separate pill containers. Attached to these motors is a flat surface that when aligned with the hole at the bottom of the container, prevents any pills from falling out. The motor will then rotate this flat surface, which is referred to as a trap door, ideally, allowing a singular pill to drop. The motor quickly rotates clockwise, then anticlockwise, at maximum speed, and at a particular angle that was settled on being 20 degrees. This allows a singular pill to drop through the trap door. There was, though, another issue that was discovered in the trap door. It was that the flat surface would not perfectly align with the hole at the bottom of the container, this was a result of the motor slightly drifting each time it was opened and closed. Drift is when the trap settles at a slightly larger angle than it is supposed to. This occurs because the motor is moving at maximum speed, so when the command for the motor to stop is sent using information from the motor encoder, by the time the motor decelerates and stops, it will have settled at a slightly larger angle than the angle specified in the code. To get around this, a PIDF loop was used. This will be explained in more detail in the *Software Design and Implementation.*

For the correct number of pills to dispense at a particular time, a function will be called which will reference the PIDF loop. In these functions, a for loop is embedded, which will call the dispense function that contains the PIDF loop the number of times each pill needs to be dispensed. Again, the specifics around the functions will be explained in more detail in *Software Design and Implementation.*

In addition, a huge problem that was faced in the dispensing process was that pills would get clogged inside the hole at the bottom of the container. A tremendously lengthy process that consisted of testing different types of containers, different types of pills with different shapes and sizes, adding funnels in many different types and forms, changing the hole sizes, and brainstorming and innovating different mechanisms that would solve this problem consumed an exceedingly long amount of time and effort, and resulted in the creation of a container that would allow for just one singular pill to drop at a time, as well as prevent pills from getting clogged up in the hole as they were getting dispensed. All these different container designs will be explored further in *Mechanical Design and Implementation*.

An additional change that was also made was the reduction of the number of pill containers, and thereby the reduction of the number of types of pills a user can use, from four to three. This was due to technical difficulties faced involving the motor multiplexer. The initial idea would have involved five motors, four for each pill type, and one for the linear sliding system that carries the color sensor (this will be discussed more later), which would have called for a motor multiplexer, as the EV3 only allows for four motors to be connected to it. Because of the lack of experience using a multiplexer, the decision was made to reduce the number of containers to three. This decision also turned out to be beneficial in other ways that were not expected. It allowed for the linear sliding system to require a smaller range of motion, which massively helped, as the rack that the linear sliding system was mounted on was slightly too short for the sensor to be able to scan each of the four containers.

The next step in the pill dispensing process is the pills dropping from the container onto a ramp that is mounted under the containers, which then routes the pills into a cup. Now, this ramp had to be mounted on an angle in order to ensure that the pills slid and did not just lay static on the ramp. For that purpose, the containers that contain the pills were mounted onto the backboard on an angle in a “stair-case” formation so that the distance between the bottom of all the containers and the ramp was equal, preventing the possibility of pills bouncing out due to the ramp being too far from the bottom of the second or third containers. However, mounting the containers did cause some trouble. Had the containers been mounted horizontally, the color sensor that scans the containers would only have needed to move horizontally. Due to the way the containers were mounted, the color sensor needed to move at an angle parallel to the containers. Creating a linear sliding system on an angle was much more challenging than making a horizontal linear sliding system.

Once the pills are dispensed for that interval, the robot sounds an alert and waits for the user to enter the button to make sure that they have taken the medicine. After that, the robot waits till it’s time to take the next dosage of medicine.

Finally, after all, dosages are dispensed for the day, the color sensor scans each of the containers and checks the level of pills inside each container. In order to execute this, the color sensor stops in front of each container. Each container has a transparent plastic cover at the front that allows the light of the pill inside of that container to transmit through. If the color of the pill is detected, nothing happens, however, if the specific color of that pill is not detected, a message is displayed on the screen of the EV3 that instructs the user to refill that container, and the robot sounds a noise. The robot then waits until the user presses the enter button to stop sounding the alert. After that, the robot shuts down. To allow the color sensor to move across, a linear sliding system was constructed. This consists of a long wooden rack that goes across between two mounted bars. The rack was custom designed for Lego gear that came with the EV3 kit and then laser cut in the woodshop.

The motor connected to this system rotates the gears, allowing the system to move along the rack. This part of the dispenser was the most technical part. It required lots of brainstorming and many different ideas to construct. The initial idea was to use a belt drive, but that was determined to be much more challenging and much less practical as it would have been less stable. In the end, this design was picked as it was simple, stable, and easier to construct. The exact details of how this system works will be explained in *Mechanical Design and Implementation.*

After this final task is complete and the user has pressed the button to stop the alert to refill pills, the robot waits ten seconds then the program ends, and the robot shuts down.

## Constraints and Criteria

Design 1:
Initially, the robot and the pill dispenser were not connected. The criteria for the robot and pill dispenser were:

- A pill dispenser with 4 different containers and one button for each container
- The pill dispenser only dispenses one pill for each press of the button
- The robot must be able to read the text file, and store the data for the correct amount of each pill at each medication time, and wait the correct amount of time between medication times depending on the time specified in the text file
- The robot must have an arm that can push the dispenser button without moving the entire pill dispenser
- The robotic arm must hold a cup, and be able to move this cup directly below each pill dispenser
- When the button of a dispenser is pushed, a pill must be dropped into the cup, and after all the pills are collected for each medication time, the robot must put the cup in the position for the user to take
- The robot must move sideways to reposition itself based on the pill it needs to collect

This design was never built, but had the following anticipated constraints and difficulties:

- Using cardboard to build a dispenser with reliable and consistent buttons that would only allow for one pill to be dispensed
- Accuracy of motor encoders and motors to move a short distance sideways while ensuring the arm can accurately press each button

- Moving the cup below each dispenser each time a new pill was collected from a different container
- Creating a robotic arm that was able to push each button and hold the cup

Design 2:
When the group was brainstorming methods to build the robot, the constraints above were identified, and the group decided on a more efficient and reliable way to build the Pillbot while maintaining the most important criteria. The robot was made static to reduce the amount of motion, and the group decided to connect the robot to the pill dispensers, using motors to control the dispensing of each pill. The overall goal of this was to dispense pills to the user according to a text file at different times of the day, but had a new criterion:

- Building a cardboard chassis that could hold x number of motors, for x number of pill containers that are used in the prototype
- Make x pill containers, each with an opening that allows pills to flow out of the container without getting stuck
- The motors control a trapdoor that opens and closes quickly, and use this to dispense one pill at a time from each pill container

- Same as before, the robot must be able to read the text file, store the number of each pill to collect for each medication time, and the time interval to wait between each medication time.
- Each of the openings in each container must lead to a common ramp, and at the end of this ramp, there must be a container for the

As the robot was built, the constraints of this were:

- The accuracy of the position of the motor, due to interference between the cardboard trapdoor and the pill container
- The consistency of the pill containers, the pills would often get stuck

Design 2.1:
This is called design 2.1 because new features were added to the second design.

When starting the software design of the project, the group realized that changes needed to be made to the robot design. By making the Pillbot static, the margin of error was reduced, by reducing the number of moving parts. The project was simplified, to the point where the group was barely meeting the requirements for the project. In the ideation phase of the project, there was an idea to create a linear slider color sensor, to detect the pill levels in each container and alert the user when each pill container needs to be refilled. It was decided that this would be added to the project to meet the requirements. An alarm to alert the user of their medication times, and an alarm to alert the user when/which containers needed to be refilled were also added.

In addition to the criteria listed for design 2, this created the following criteria:

- Linear slider must have a color sensor attached to it, and a motor with a motor encoder, and be able to move between each container consistently and check the pill level of each container
- The linear slider must be able to accurately move between the clear front part of each pill container
- The alarm must play after all the pills for each medication time are dispensed, and the alarm should continue playing until the user presses the button

- The linear slider must be set up so that the color sensor detects the same level of each container for pills
- The timer for tracking the time between each pill interval must not begin until the user presses the button to indicate they have collected their pills
- After the user has pressed the button to indicate they have collected their pills after the last medication time, the linear slider must check the pill levels in each container. If the pill levels are below the minimum level in a container, then the robot displays a message to the user to refill that specific container and plays an alarm until the user presses the button

By adding these new features and criteria, the project had the following additional constraints, in addition to those constraints mentioned in design 2.

- Accuracy of the color sensor in reading the different colors of the skittles used as pills
- Accuracy of the motor encoder of the linear slider's motor in moving the linear slider to the correct pill container

- Stability of the frame to hold the linear slider sensor and not damage or bend the walls of the cardboard chassis

The criteria for the linear slider drove how the linear slider and its frame was built. A rack and spur were used to hold the linear slider itself, as the group decided this would be the most reliable and controllable way to meet the criteria of moving the color sensor to the different containers. The criteria for a stable linear slider led the group to build a frame made of the TETRIX kit metal extrusions to hold the linear slider.

Before building the chassis, a certain number of pills to be used in the prototype had to be decided on. This impacted the width of the chassis, the number of containers, and the number of motors required. Initially, the group planned to use 4 different containers, but due to a faulty motor that was unable to meet the criteria, 3 motors were available, and thus 3 pill containers were used.

The criteria for the trapdoor dispensing mechanism guided many aspects of the project. The dispenser needed to dispense one pill at a time, and this directed the design of the trapdoor mechanism. It also guided the physical design of the chassis, including the ramp to the collecting cup, and the cutouts of the cardboard chassis for the motors. Constraints with the accuracy of the motor encoder, and the constraints on the build quality of the pill containers caused problems with the accuracy and implementation of the dispensing mechanism.

The criteria for the user to interact with the EV3 brick, to press the buttons and read the message if a specific container needs to be refilled. This criterion led the group to mount the brick on the top of the linear slider's frame to allow for this.

The time criteria were not important in the physical design and implementation as this was as simple as changing a constant in the text file. The medication time alert and refill alert alarms did not significantly guide the physical design as they were functions that were added to the code and did not change anything to the physical design.

## Mechanical Design and Implementation

As seen in Figure 1: Sketch of Initial DesignFigure 1: Sketch of Initial Design, the whole physical system revolved around the idea of creating a suitable cardboard dispenser that will allow the team to do what they aimed for. When the group thought of helping people by handling their pills and essentially giving them the correct dosage of pills at the specific time needed, a dispenser was what came into mind. A dispenser allows the user to store their pills in a specific container where the robot would initially pick up the different pills from their assorted place. A colored tape would also be taped parallel to the dispenser so the robot could follow a straight path with the color sensor to have no discrepancies in its course.

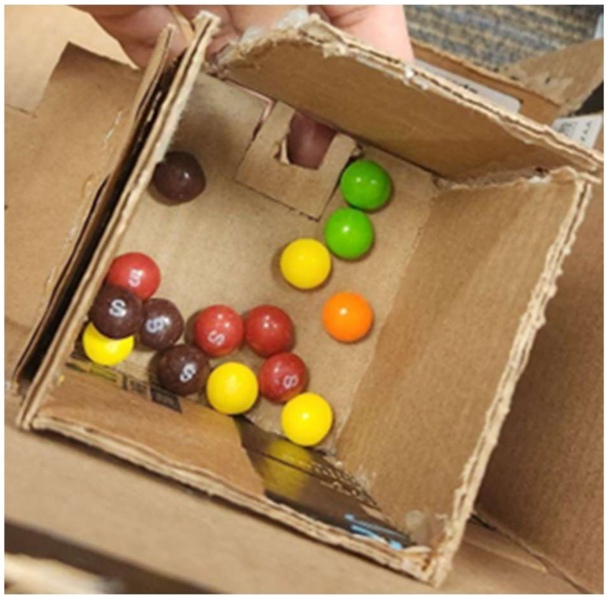

*Figure 3: Pill Container Idea*

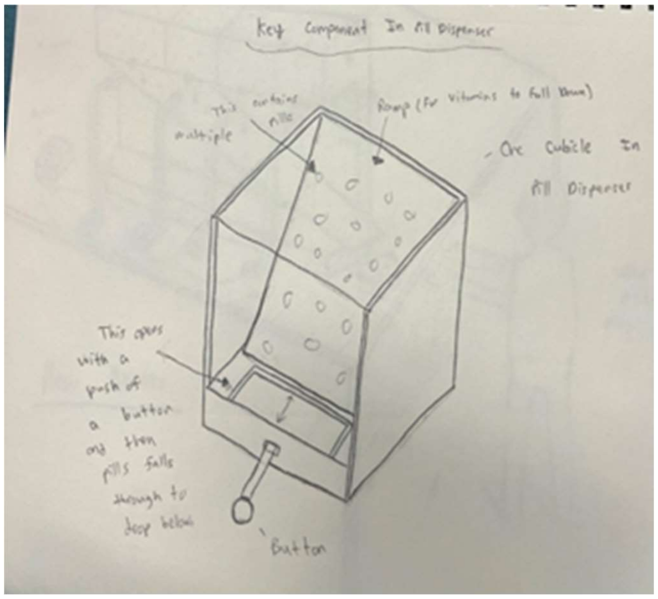
Key Component In Pill Dispenser
This contains multiple pills
Ramp (For vitamins to fall down)
One Cubicle In Pill Dispenser
This opens with a push of a button and then pills falls through to drop below
Button

*Figure 4: Pill Container Sketch*

The only method the team thought of when brainstorming the idea of the robot picking up pills was with a simple push on the button of a container, and one pill will drop through the slot. With a spring attached to the popsicle stick that acts as the button, when pushed the slot opens, and when released the slot closes as shown below on the left. A ramp would have been added inside the container so the pills would have fallen toward the slot. One major concern the team had with this potential design was the inaccuracy it may have when dispensing the correct number of pills. Designed with a spring mechanism, dispensing one singular pill at a time almost seemed highly unlikely as all the pills would be falling towards the slot, resulting in more than one pill being dispensed. One possible solution to fixing that was redesigning the slot to be the size of one pill as shown below on the right.

Another important factor and manipulative variable to consider was the pill size whereas the responding variable was the slot size. Depending on the size of the pill, the slot size would have to be changed accordingly to only allow one pill to drop. The shape of the pill was agreed upon by the group to be either spherical, ovular, or bean-shaped to represent the same physics of a real pill

Another fault behind the engineering of the conceptual drawing on the left was the chance of the very last few pills possibly getting stuck on the sides of the slot as there would be nothing there to push the pills off the side. To stop that from ever happening, a funnel design would be added to the inside of the container where there are ramps coming from both sides as shown below. These were all taken into consideration when brainstorming the procedure behind building the physical system. When moving onto the actual building process, that is when the team ran into serious problems which will be looked further upon in detail down below.

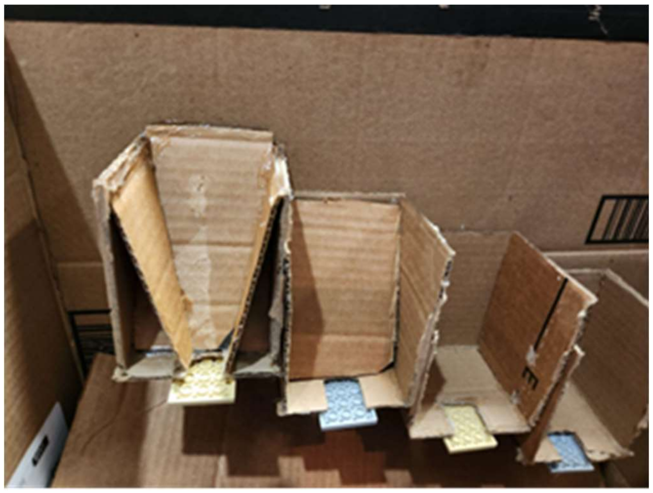

*Figure 5: Funnel Versus Ramp*

In the team's brainstorming process, the team planned to build a robot that simply lifts its arm up enough to push the button of the container so that the slot opens just enough for one pill to fall down onto the container carried by the other arm of the robot. It was a rather simple design build that the group was contemplating to build since it lacked the aspect of major modifications to the main robot.

From the first stage of brainstorming, the group noticed that the entire project as a whole was rather too "simple" and lacked the modifications and engineering needed to display a highly developed project. More changes needed to be done to be more efficient and effective for the user.

Onto the second stage of the physical system where the group made major changes from the initial idea of the pill dispenser. Three important engineering decisions were made:

i) Motorizing the dispensing mechanism.
ii) Collecting ramp.
iii) Dynamic to static.

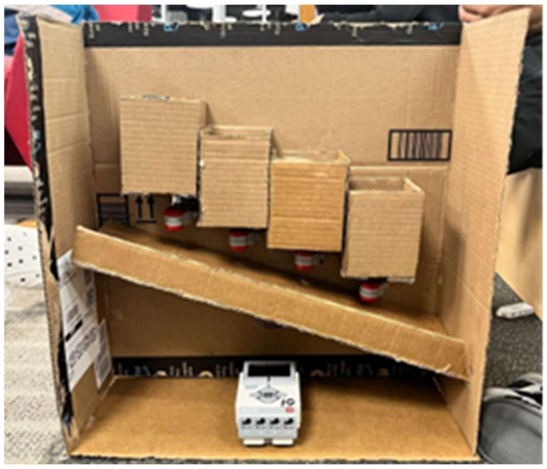
*Figure 6: Front View of First Dispenser Design*

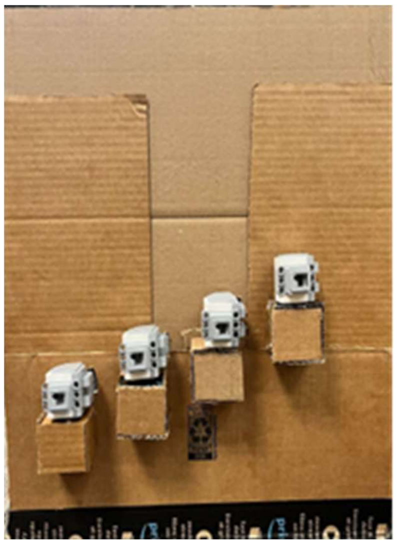
*Figure 7: Back View of First Dispenser Design*

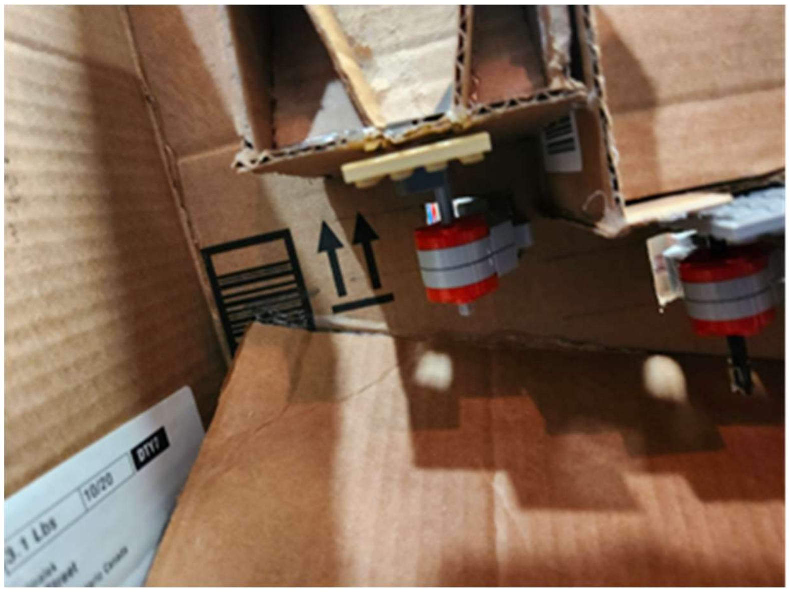

Figure 8: Close View of First Dispenser Design

Switching from a spring contraption slot to a motorized dispenser enhances the precision and accuracy of dropping one pill at a time. The team was unsure about the force required by the robot arm to be able to open the spring-engineered button consistently, so implementing the EV3 motors was a good solution to solve that problem. The motors are taped upon small cardboard cubes that slide through a cut-out slot from behind and get placed right under each container so the trapdoor can rotate freely to let a pill through. It is a much more efficient design as it gets the job done quickly for the user and allows the team to successfully redesign the body of the starter robot, fulfilling one of the expectations the project was supposed to meet.

Since the team decided to utilize the EV3 motors for the trapdoor of each container, building a separate motorized robot to just pick up certain pills and convey them to the user seemed unnecessary. Rather it was more optimized to add a feature where the pills can be conveyed without the aid of outside help. A simple design that was added to the main frame of the pill dispenser was a ramp inclined at a twenty-degree angle. Each container is the same height above the ramp, operating directly proportional to the ramp. When a certain number of pills are dropped, instead of a robot catching the falling pills into its' bowl where it would have to be very precisely put into the right position, the ramp is a great feature to perfect the accuracy of the pills falling into the bowl. The open slot at the bottom of the ramp as shown below is where the pills fall, ultimately falling into a bowl for the user to pick up from.

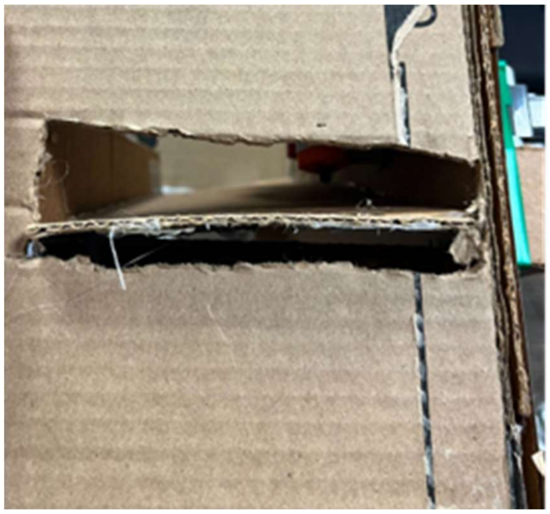
Figure 9: Ramp Slot

The change from dynamic to static was the key to efficiency behind getting the overall object done at a faster pace without the need of implementing unnecessary measures. An EV3 brain that powers the whole dispenser itself rather than relying on the usage of an external robot to do its fundamental job of dispensing pills. With the use of an external robot, the environment that it would have to be in would have to be carefully considered as the team would not want any external factors affecting the robot in some sort of way to prevent it from carrying out its task at hand. However, by implementing the Lego EV3 components into the main physical system itself, exposure to outside factors would not hinder the system's chances of carrying out its task. It also allows the system to be more applicable in real-world situations as the physical system can be placed in most environments and still fulfill its purpose.

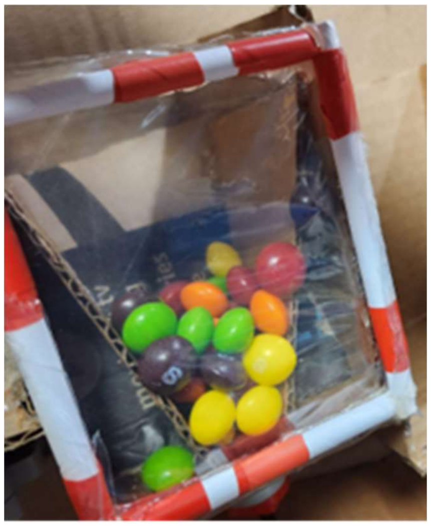
Figure 10: Pills Jammed in Funnel Design

Although progress was being made, the overarching problem of pills getting stuck remained the same. This time it was getting stuck in another way, since all the pills fall towards the slot that only allows the size of one pill to fall through, an accumulation of the pills clumped on top of the slot allows no pills to fall through at all as shown below. Without human intervention, the group had no solution that allows it to shake the container so that one pill can drop. The engineering concept behind the funnel initially made sense but seeing how the pills were getting stuck during the testing stage, the group believed that the size of the skittles was the problem. Therefore, the group wanted to test out other types of candy to see if the hypothesis was correct. The different candies that were tested are:

i) Peanut M&Ms

This was the first candy that the group started testing off with, but the main issue with this type of candy was that the packet came in all sorts of different sizes ranging from the size of a marble to the size of a skittle. The variation of the sizes made it very hard to test with as some of the M&M's could not fit through the open slot at the bottom of the container since it was too large. Since it was causing too many issues the group made the decision to use another type of candy.

ii) Mike and Ike’s

The reason the group wanted to test out these jellybeans was because they were relatively small and represented the actual shape of a real pill. The problem with this candy was when falling to the slot at the bottom of the container, some would fall horizontally down and would get stuck since it was too wide for the width of the slot. The slot was designed to only allow one pill to fall through in a vertical orientation. So to potentially prevent it from getting stuck, the team made the slots wider, allowing Mike and Ikes to fall horizontally and vertically down. However, when testing it out, when the trapdoor would rotate open, too many pills would fall out due to the size of the wide slot. The group was able to prevent the candies from getting stuck but ultimately allowed too many pills to fall rather than only one pill.

iii) Skittles

Having an ovular shape, these skittles all came in the same size and were the smallest candy the group tested out. Changing the slot to only allow one skittle to fall, the only problem with this was an accumulation of the skittles preventing one to fall. This was an engineering problem that only the group could fix, there was no fault in the candy. There was also no problem related to the size of the candy which is why the group went ahead with proceeding with skittles as pills for all future test cases.

Consequently, because of not being able to overcome this challenge, the group had to change their idea of a funnel design and brainstorm another method to drop one pill at a time.

## New Additions

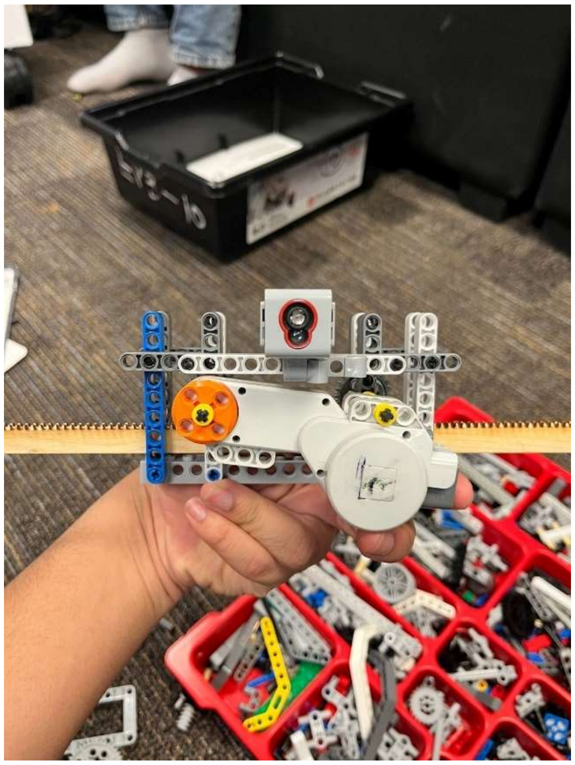

*Figure 11: Front View of Pill Level Detector*

To elevate the project further and make it more convenient for the user, another engineering modification was made to detect the pill level of each container to notify the user when to refill the container with pills. Brainstorming the linear slider pill detector was particularly time-consuming as there were many different possible ways to approach the procedure. From deciding what to use to create the linear slider, and what pieces could be used to mount the color sensor, to perfecting the elevation and angle of the slider. There were many things to consider before building that had to be carefully planned out to avoid future errors as it is a very important feature of the team's project. If the color sensor was not mounted correctly and precisely, the reading would be off, relaying misinformation to the user.

Before building the mechanism, the team had different ideas for potentially creating this pill detector. The various plans were:

i) Conveyor Belt Pill Detector

With this idea, the group had planned to design a conveyor belt connected to two 36-tooth spur gears that rotate the belt itself. One spur gear is placed on an upper elevation near the far left container and the other gear is attached on a lower elevation near the far right container so then the detector can move on an angle from the upper left container to the bottom right container. Both spur gears have an axle go through where one of them would also be connected to an EV3 motor. The only concern was where to place the color sensor as it's the key component of this mechanism. The group had many doubts about resting the sensor itself on top of the belt as it would be very unstable, preventing it from reading the pill level of the container accurately. Also, the weight distribution across the conveyor belt would be very uneven due to the color sensor which could cause the belt to potentially snap by overusing the mechanism. There were many concerns behind this idea which

ii) Color Sensor Attachment with Worm Gears and two 24-tooth spur gears

Moving on from the initial idea of a conveyor belt, the group made a transition to building a mechanism similar to that of a linear slider where the color sensor can just slide across a rack. Initially beginning with this plan of action, the group was thinking of using small lego racks and connecting them together to act as one long rack. However, fortunately, the group was provided with a laser-cut rack long enough for the color sensor to travel from one side of the pill dispenser to the other side. As the color sensor itself cannot slide across the rack, the group had to build a lego attachment where the sensor and motor can be mounted on and can altogether slide across the rack. To be able to translate across the rack, two lego gears had to be used. The team had two choices, using worm gears or spur gears. First starting off with two worm gears, when moving the lego attachment left and right across the rack, there was a noticeable amount of friction that seemed like it may have been potentially damaging the teeth of the laser-cut rack. To prevent this from happening, the group wanted to test out using other types of gear to see which one would give the best result.

iii) Color Sensor Attachment with Two 24-Tooth Spur Gears

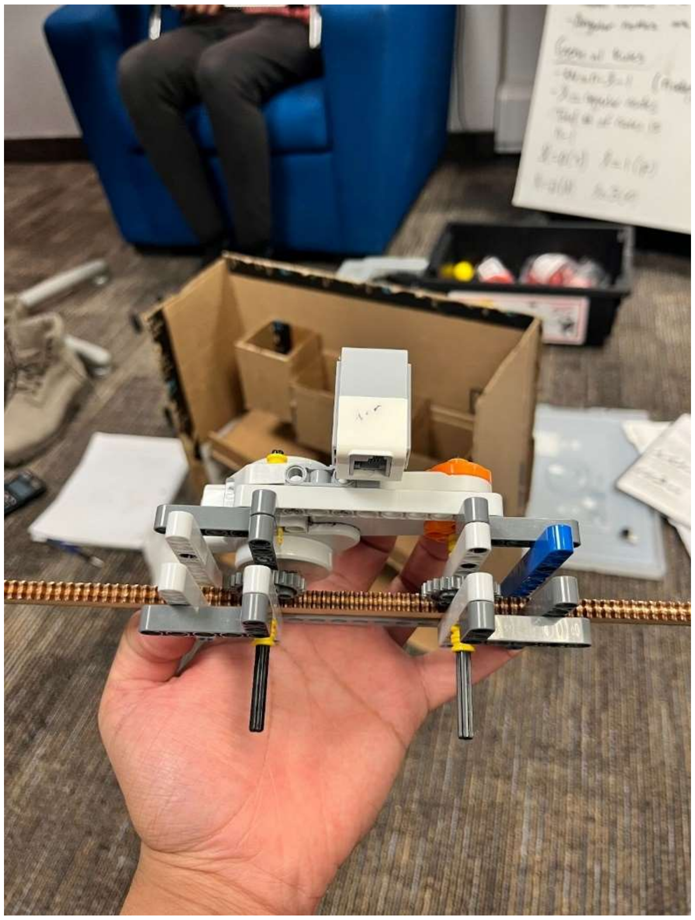
*Figure 12: Back View of Pill Level Detector*

Replacing the two worm gears with two 24-tooth spur gears resulted in a major difference to the overall mechanism. There was a difference in friction where there was not much friction to the point where it damages the tooth of the laser cut rack. Moving the sensor attachment left and right felt smooth and the Lego attachment was able to support the weight of the NXT motor and color sensor.

Since the pill detector sensor was built, the last stage was firstly mounting the rack at its specific angle and height. The group had many ideas on how to mount the laser cut rack, from potentially hot gluing the sides of the rack to the sides of the cardboard itself. However, the main concern behind that idea was the load and weight that the cardboard would have to sustain. The group believed that the cardboard was not strong enough to hold the linear slider which is why they took no chances in mounting it using that method. Another way the team thought of mounting it was using TETRIX pieces as they are made of heavy-duty aluminum, strong enough to definitely hold the weight of the color sensor attachment. Since the material was durable enough to carry the weight, the team carried on building a frame with the same width and height as the pill dispenser. The TETRIX frame was designed to allow the laser-cut rack to reach from one side of the frame to the other.

Furthermore, after considering what the attachment would be mounted on, now what needed to be considered as the angle and height placement of the color sensor. The angle of the rack placement was very important as placing the rack too high, the color sensor would only be reading the elevation of the upper left container and moving across to a point that is nowhere near the next container. Deciding the angle was all calculated around the placement of the different containers. When building the physical system, the group made the containers aligned at a 20-degree angle. The team, therefore, decided to mount the pill detector mechanism at a 20-degree angle, perfectly aligning the color sensor to the containers so it would be able to read from each container and not miss one. The angle was chosen and now the height had to be considered. If the color sensor was placed too high, the sensor would be taking in the pill level from the top of the container where it needed to be near the bottom of the container. Seeing which height was best fit for the sensor took lots of trial and error, moving it slightly up and down with and sliding the sensor across to see if it reads the bottom of each container consistently, and then marking the exact placement of the rack on the TETRIX frame.

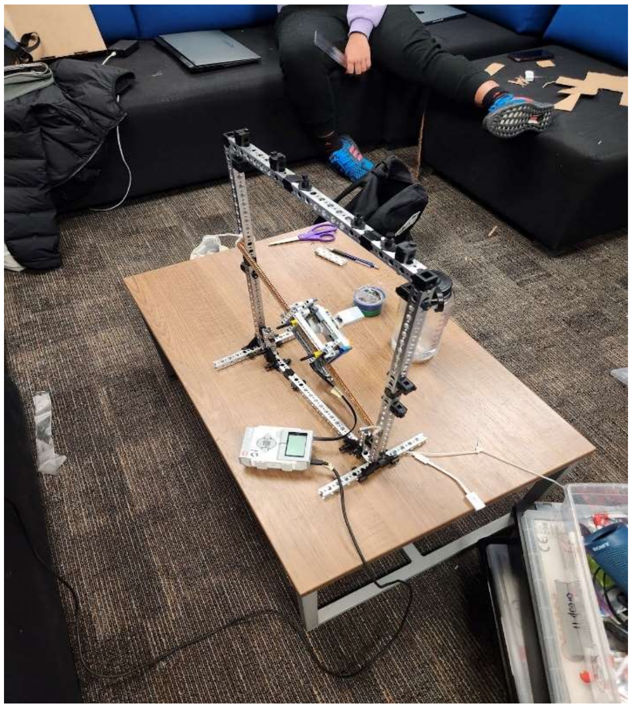

*Figure 13: Pill Detecter TETRIX Frame*

Instead of hot gluing the rack itself to the TETRIX frame, small cardboard pieces were used to glue them together to connect one another. At the end of the mountain process, zip ties were used to apply pressure on both ends to keep it from not falling.

When testing out the color sensor attachment and moving it across from one side to the other, there was a slight problem. When it got to the ends of the laser cut rack, the top and bottom pieces of the lego attachment would prevent it from moving a couple of centimeters more to get the correct reading from the container due to the frame. Since the initial design of the top and bottom pieces were sticking outwards to the sides which press against the sides of the frame when at one end, the group decided to change the pieces to prevent them from clashing against the frame. The group made the color sensor attachment slightly narrower than before (compared to the first photo of the Lego attachment), and changed the pieces from the top L-shaped pieces to the grey connectors as shown below

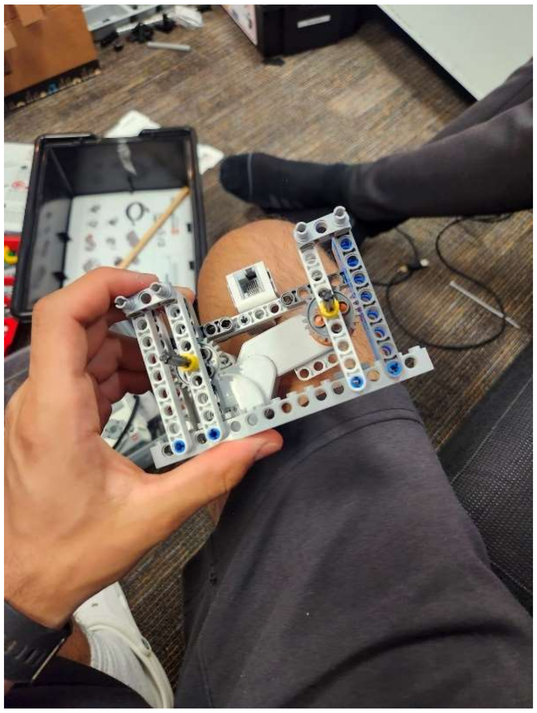

*Figure 14: Modifications Made to Pill Level Detector*

## New Modifications Made

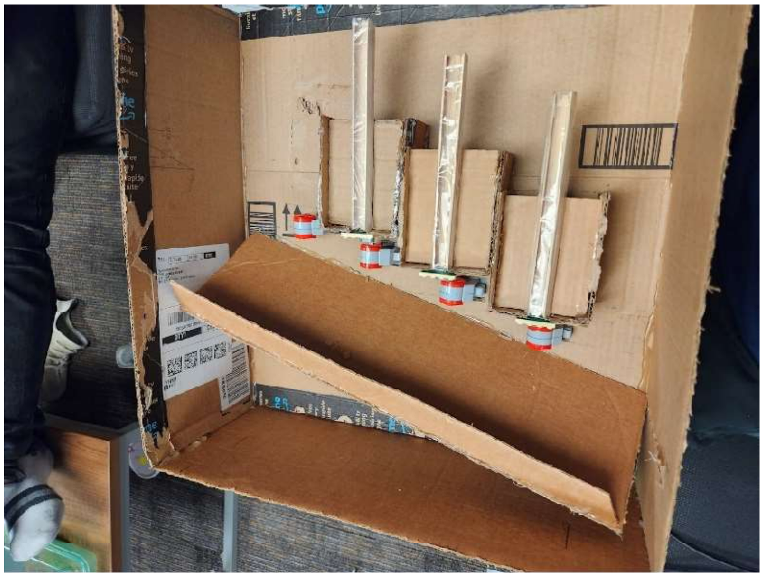

*Figure 15: New Single File Dispensers*

After the pill level detector mechanism was all built, the team decided to go back to redesigning the pill dispenser. The main problem of the pills getting stuck or too many pills getting dispensed remained and at one time, the group could not think of any other possible solution. However, brainstorming during their own free time, the group was able to think of one last resort. The single file dispenser. At first, the only problem some had with this design was the major reduction of pills being used, but there was no other resort the group could possibly think of at that point in time. But the benefits of this design outweighed the cons.

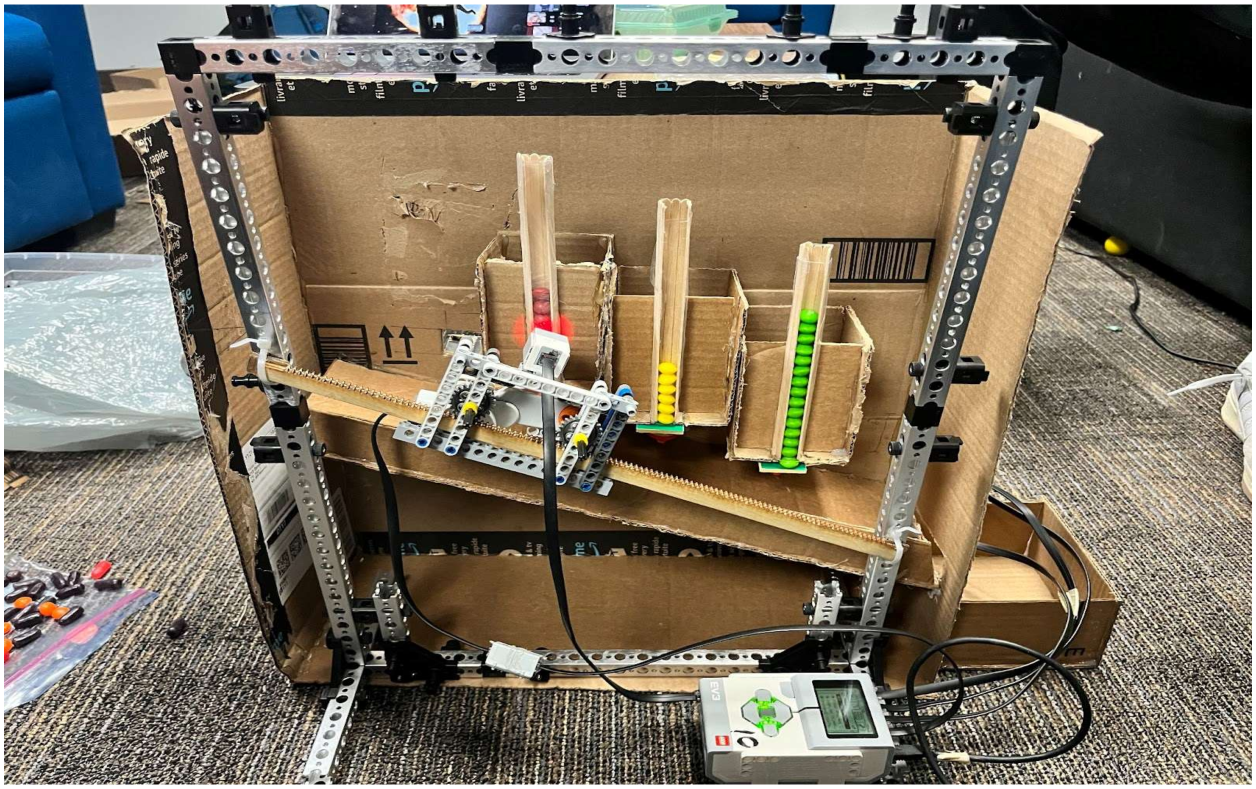

*Figure 16: Pills Inside Single Pill Dispenser*

The ease and smoothness of inserting the pills down the tube allowed no chance of the pills getting stuck. A small plastic sheet was wrapped around the pill tube to allow the color sensor to read through the transparency and see the pill level. A cardboard cover that is opaque will not allow the color sensor to pick up anything which is why the team made the decision to change the material of the front cover.

With the initial design of this pill tube, only popsicle sticks, tape, and plastic sheets were used. However, with those materials being used, the team ran into another problem concerning the tube. When running multiple test runs to see if the pills were at first properly being dispensed, there would be cases where the pills would get stuck around the middle of the tube due to the rigidness of the tube. Using tape was not the best adhesive to use as the tube was still very flimsy causing the sides of the tube to cave in. Since the sides were caving in, some pills would get caught up, and yet once again the pills would get stuck, but there was an easy solution to this.

To prevent the pills from getting stuck, instead of using tape, an alternate solution was to use hot glue instead to make the sides of the tube not cave in. It would also make it more durable.

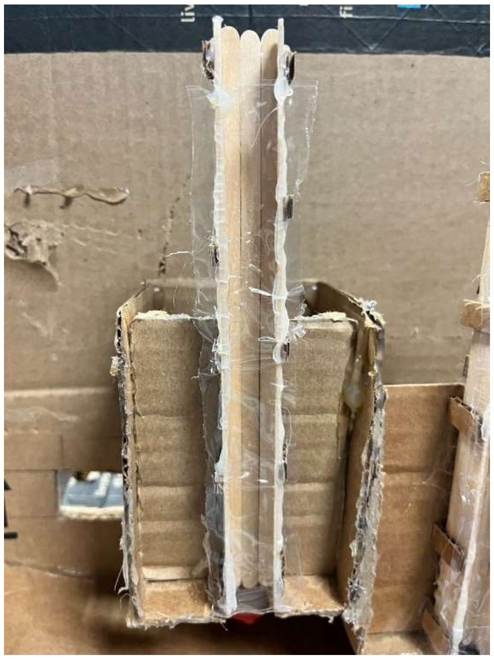

*Figure 17: Hot Glued Single File Tube*

Small cardboard pieces were used to prevent the sides from caving in, this allowed all the pills to be dispensed in a single file order. When running more test runs, the pace that the pills would fall down was intense. This all came down to the proper placement and adjustment of the trapdoor below each tube opening at the very minimum at a fast speed to only allow one pill to drop. Doing all these test runs, there were a few things the group picked up; one was the number of pills the team would put in the tube. The more pills there were in the tube, the more weight there was on the very bottom pill, causing it to fall at a tremendous speed and not quick enough to only allow one pill to fall. When the team would only place a few pills around 4-5 in each container, the trapdoor was quick enough to only allow one pill to drop.

One trade-off that the team had to make near the end of the design process was removing one of the containers from the pill dispenser. The far-left container's motor was not functioning properly when running the last few test cases. Even when left alone to cool down, it would still not be able to open the trapdoor at the same pace as the other motors being used. Another key point to touch on was the team's capability on using the multiplexor. The multiplexor would have allowed the group to use all four containers, but since the team was running into multiple problems behind the physical system itself, they did not want to focus too much on using the multiplexor. The time left needed to revolve around building a solid foundation of a physical system and not using the multiplexor to reduce the stress the team was facing when fixing the pill dispenser.

As seen in Figure 2: Final Design, putting all the components and modifications put together, this was the overall design created by the team. Each component seen in the photo above took lots of brainstorming and testing before finalizing the product.

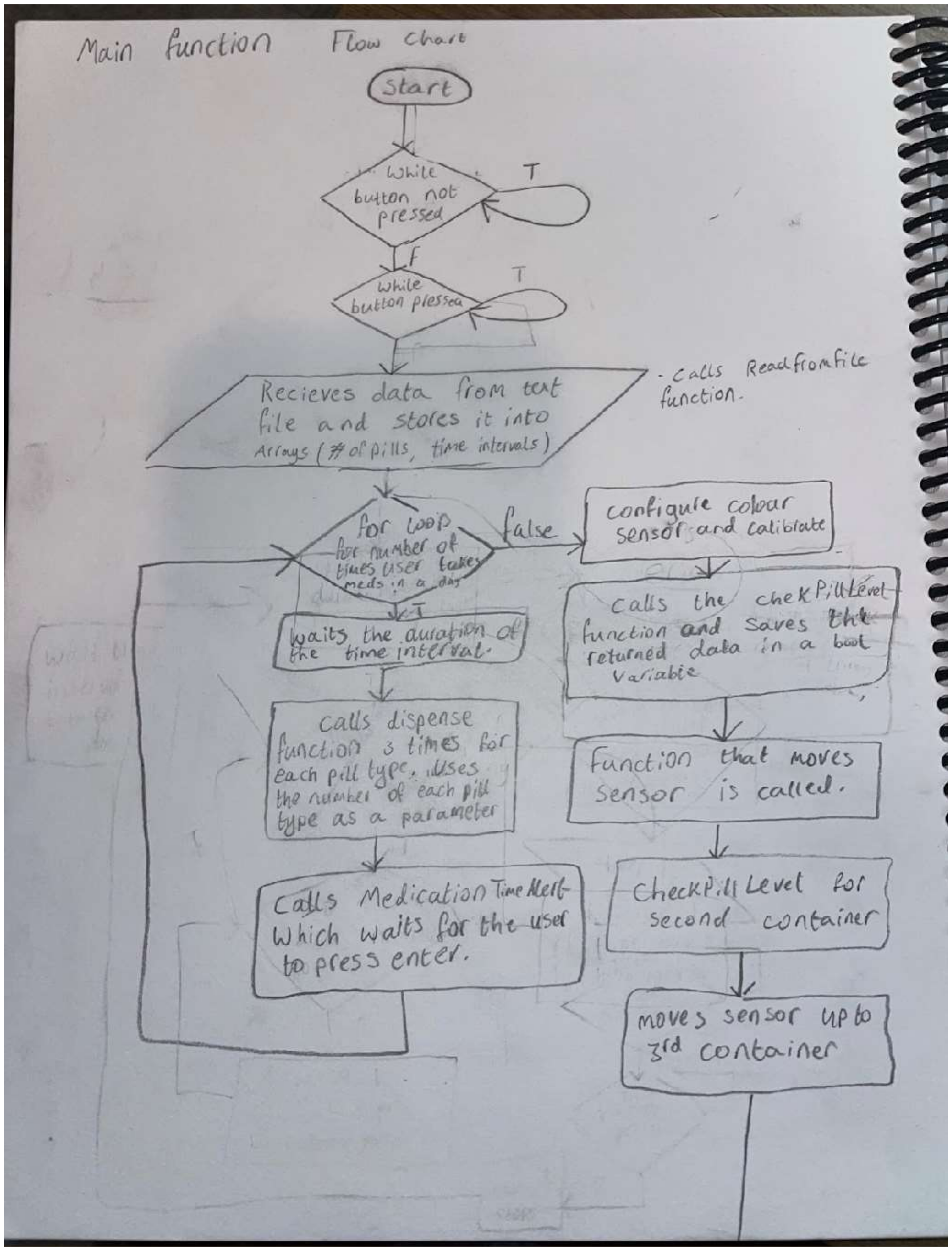
Main function Flow chart
Start
While button not pressed
T
F
While button pressed
T
Recieves data from text file and stores it into Arrays (# of pills, time intervals)
- calls ReadfromFile function.
for loop for number of times user takes meds in a day
false
T
configure colour sensor and calibrate
waits the duration of the time interval.
calls the chekPillLevel function and saves the returned data in a bool variable
calls dispense function 3 times for each pill type, uses the number of each pill type as a parameter
Function that moves sensor is called.
Calls MedicationTimeAlert which waits for the user to press enter.
CheckPillLevel for second container
moves sensor up to 3rd container

*Figure 18: Flowchart Page 1*

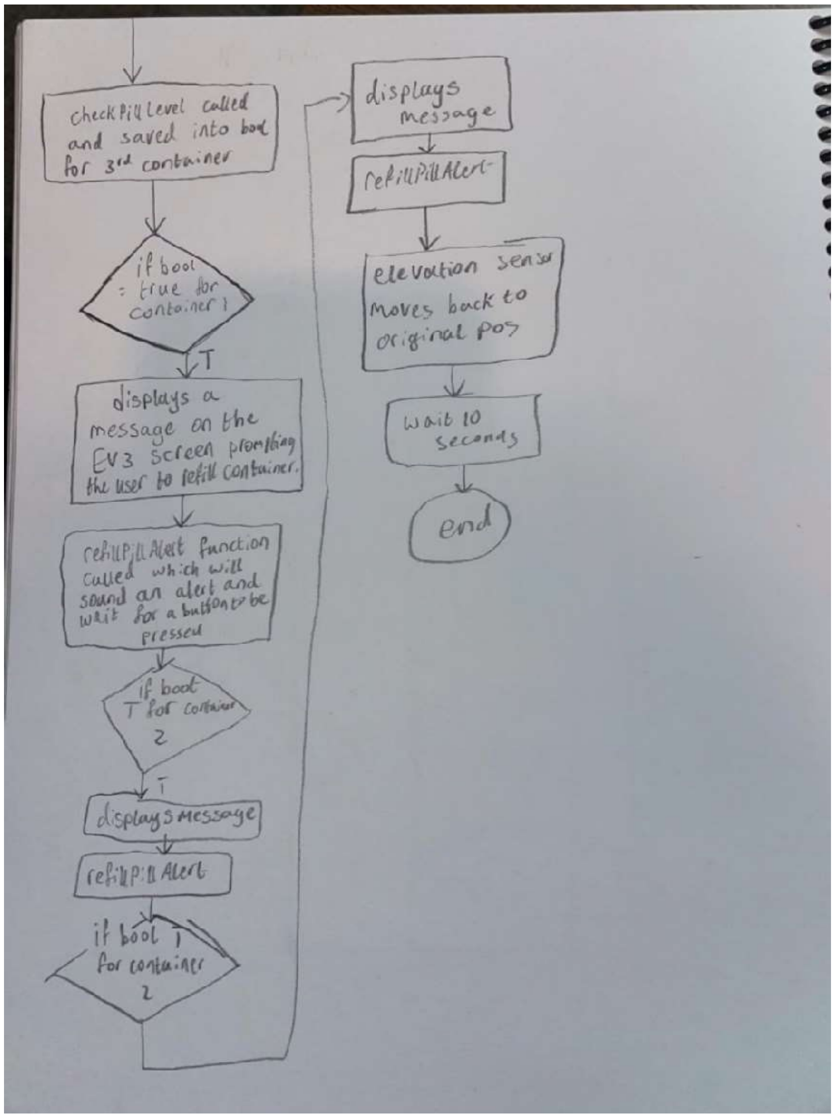
CheckPill Level called and saved into bool for 3rd container
if bool = true for container 1
T
displays a message on the EV3 screen prompting the user to refill container.
refillPillAlert function called which will sound an alert and wait for a button to be pressed
if bool T for container 2
T
displays Message
refillPillAlert
if bool T for container 2
displays message
refillPillAlert
elevation sensor moves back to original pos
wait 10 seconds
end

*Figure 19: Flowchart Page 2*

The functions that perform all these tasks will be explained in detail below, along with other details like how the data was stored, problems encountered and the reasons for any decisions/trade-offs.

- `void ReadFromFile(TFileHandle & fin, int* pill_1, int* pill_2, int* pill_3, float* wait_time_hrs);`
    - The ReadFromFile function has return type void because it was mainly used to read from an input file and store the values from the input file into parallel arrays.
    - The first parameter of the function is pass by reference to be able to read from the file accurately.

  - The function's second, third, and fourth parameters are the parallel arrays that store the consecutive rounds of pills that the robot would dispense. These parameters have an integer return type because there is a set amount of pills to be dispensed and the pills can't be dispensed in halves or any increments less than a whole pill.
  - The final parameter is the fourth parallel array which stores the time intervals between each pill time. This parameter is a float as the wait time between dosage periods could be decimal values as it is measured in hours.

The program began by reading from an input file that was written to store the time intervals between dosage and the number of pills of each pill type. To read from the input file, a "ReadFromFile" function was precisely written to be able to read each value and store them into arrays. The file was split into four lines of values where the first line represented the time intervals, and the second, third, and fourth lines represented the number of pills to be dispensed at each time. The function was coded using a for loop to read each line and store the values into parallel arrays that the EV3 can later call values from to dispense the right amount of pills at the right time.

This function was initially written in another file and took an extensive amount of time to be formatted. In the beginning, there was doubt about how to incorporate arrays as parameters into functions in RobotC. This was the biggest challenge on the very first day of writing the code. After spending endless hours surfing the internet and troubleshooting the function over 50 times, the solution was to get help from the WEEF TAs. First, there were several tests conducted attempting to pass an array as a parameter in a function. To properly test this, a function was written to store initialized variables into parallel arrays using a for loop. After ensuring that the parallel arrays have been functioning correctly as parameters in the function, there was yet another challenge to face: File Input. Due to the lack of experience with File I/O in RobotC, including it as one of the inputs was a tedious task. There were several issues encountered which led to troubleshooting how to get the EV3 to read from the file with efficient code in a singular function. To accurately test file input on RobotC, a beta file was coded and downloaded on the robot. A function was then implemented to read the values from the file using a while loop and output them to the console. After several minor errors and setbacks, the function worked flawlessly as it was possible to output as many values as needed. The final step was to incorporate the function using parallel arrays and the function using File I/O into one detailed, efficient function. The result was none other than the "ReadFromFile" function, which took over three days to complete.

- `float clipToBottom(float powerToMotor);`
  - Receives a number which is the difference between the angle the trap door has rotated and the angle it needs to rotate.
  - Returns the received value if it is greater than 0.1. Returns 0.1 if the received value is less than 0.1.

*Explained with the openAndClose function*

- `void openAndClose(int motor_num, int angle);`
  - Receives the integer representing the motor number (ex: motorA, motorB, …etc)
  - Receives the angle at which the trap door will rotate
  - Does not return any values, only opens, and closes the specified trap door at the specified angle.

One of the problems faced was the drift of the trap door after it opens and closes which causes it to not settle exactly underneath the hole at the bottom of the container. A PIDF loop function was created to counter that drift. Firstly, to open the trap door, a while loop was created that keeps the motor running while the angle at which the motor has traveled is less than the angle it needs to travel. The error, which is the difference between the distance the trap door traveled and the distance it was supposed to travel, is calculated. If that error is found to be more than 0.1, the error is multiplied by a constant named PROPORTIONAL which in this case was a set value of 1/45.0. After that, the number is scaled up so that the motor does not travel at an extremely slow speed. That number is then assigned as motor speed. Therefore, when the error between where the motor is and where the motor is supposed to be getting smaller, the speed assigned to the motor gets smaller and smaller. Therefore, when the command is sent to the motor for it to stop it is traveling at a slower speed closer to 0 and does not need to decelerate as much as it would if it were traveling at a higher speed, allowing it to stop almost instantly after the command is sent, preventing it from drifting. That same exact process is repeated for the trap door when it is returning to its original position. The function that does this is known as the open and close function. This function references the "clipToBottom" function which just ensures that an error less than 0.1 is not used, because when a number that small is scaled up and used as the motor speed, even then, the speed of the motor would be too small.

- `void Dispense(int container, int num_pills);`
  - Receives the number of the container that pills need to be dispensed from (1, 2, or 3).
  - Receives the number of pills that need to be dispensed.
  - Calls the open and close function

This function will use the number of the container received as a parameter to figure out which motor to use to call the openAndClose function. The function will then call the openAndClose the exact times that the pill needs to be dispensed. It will do this by using a for loop that loops exactly the number of times that pill needs to be dispensed.

- `void moveElevationSensordist(int motor_power, float dist_cm);`
  - Receives the distance the color sensor needs to move up/down the rack. Positive distance to move it from left to right. Negative to move it from right to left.
  - Receives the motor power. Positive motor power to move the color sensor from left to right. Negative motor power to move the color sensor from right to left.
  - No return; moves color sensor.

This function controls the movement of the color sensor up and down the wooden rack that goes across the front of all three containers. When the motor that controls the movement of the color sensor up and down this rack rotates a specific number of clicks (motor encoder), the color sensor will move up or down the linear sliding system a specific distance. This specific distance is related to the circumference of the gear used and the number of teeth in the gear. Every time the gear rotates a specific number of clicks/degrees (motor encoder) it will drive the color sensor down the rack a specific distance. That relationship was found and used to determine the angle the motor needs to rotate to get from container 1 to container 2, then container 3, then back to its starting position.

- `bool checkPillLevel(int color);`
  - The checkPillLevel function has a return type of boolean because the purpose behind it is to inform if the pill level is low or not.
  - This function has parameters of integer color so that when the color sensor can detect color it returns a value.

    - The main aspect of the function is the if statement that states if the color sensor can read color, return false, and if not then return true. This summarizes the basic aspect of if the color sensor can still detect pills in the container and then return false since the container is still full of pills. And if the color sensor returns true, that means the pill container is empty and the color sensor could not detect any pills.

The checkPillLevel function was split into its own task block to make sure that the color sensor was working properly and that it was able to detect the pills and return the proper value. The problem with this function is its usage of the color sensor, which in most cases is unreliable and faulty. This caused the group to go through different stages of testing the color sensor using this function to make sure that the color sensor would function properly when it's in a proper orientation and distance from the pill container.

To test this function, the group made a beta code to be able to adjust the placement of the color sensor on the linear slider by using the left and right buttons on the EV3. The code would basically move the motor clockwise when the right button was pressed so that the color sensor would move down the linear slider, and counterclockwise to move the color sensor up the linear slider. This allowed the group to place the color sensor in front of each container without using external force on the sensor to minimize any damage. After placing the color sensor in the proper position, the group tested the function to see if the color sensor would return any values. After several trials with different pill colors, orientations, and distances from the pill container, the group was able to place the color sensor a certain way to make sure that it would read if there was any color or not most of the time. This is essentially important to the project as the color sensor would make sure that the user would be alerted if the color sensor would return a true value, which is the next step of the process in “refillPillAlert”. This is why the checkPillLevel has a return type of boolean so that it can later be used to know if the level of pills left is below the required amount.

- `void refillPillAlert(bool pill_level);`
    - The refillPillAlert function has a return type void because its main purpose is to sound an alert to the user to refill a pill container.
    - The parameter is defined to be a boolean because the pill_level would either return true or false depending on the output of another function(refer to Appendix C on line 8).
    - The function mainly uses an if statement that would check if the pill level is true, which means that the pill container is below the required amount, and would play an alarm to alert the user of the almost empty container.
    - The function then requires the user to press the enter button to ensure that the user has been notified of the empty container. The click of the enter button would then clear all the sounds being played.

An important task block that was split up from the rest of the code was the function “refillPillAlert”, as it utilizes the speaker of the robot and a sound file. The main issue of this function was trying to use a sound file that isn’t in the EV3’s list of sound files. Since the robot can only play sounds from a certain file type, it was important to find out how to convert normal mp3 sounds to .rsf, which is the file type required by the robot.

To convert from an mp3 file to an rsf file, a reference video was used as a guide(Reference). The video helped in providing software specific to the EV3 and step by step explanation of how to upload and export the sound file. Using the LEGO MINDSTORMS Education EV3 software, an mp3 file was uploaded to the software using the sound editor. This allowed the user to edit the specific sound by cutting the sound down and increasing or decreasing the volume. Once the sound file is uploaded by the user and downloaded to the software, it automatically gets converted from an mp3 file to an rsf file. The next step is to navigate into the project properties and locate the name of the sound file and make sure that it is a .rsf file. From there, the user would be able to export the file into the computer files and insert it into the "Sounds" file under "EV3 System Files". This ensures that the file would be in the same place as all the other sound files to eliminate any errors when the robot reads through the file. The final step is to download the sound file to the robot using the File Management Utility. The process of adding one's own sound file to the robot is quite extensive and confusing, however, it was deemed useful as it added to the aesthetic of the project and aided the complete project design. When the file was added to the robot successfully, the function was tested to make sure that the sound would play properly when the boolean was fulfilled and would clear when the button was pressed. With every trial, notes were taken to understand what would go wrong with the code and why would the sound loop even though the button was pressed. In some cases, the function would play the sound before checking if the pill level is true or false, and this was due to how the function was referenced in the main code.

- `void MedicationTimeAlert(int time);`
    - Similar to the "refillPillAlert" function, the "MedicationTimeAlert" function serves the same purpose of sounding an alert but for a different cause.
    - The function is mainly used after every time interval to alert the user that the next dosage period is coming.
    - The function is programmed to alert the user and wait till the user presses the enter button to ensure that the user is aware and nearby so that the robot would dispense the next round of pills.

Just like the "refillPillAlert", the "MediationTimeAlert" was split into a different task block to test the functionality of the sound file that was made. However, unlike the "refillPillAlert", for this function, the group decided to manually record the sound of one of the group members and use that sound to alert the user.

The process of converting the file from mp3 to rsf was very similar. At first, the group recorded a voice recording that says, "Please come take your pills, make sure to press the button when you're done". This is to alert the user that the time interval between medications has passed and the next round of pills is due. After recording this voice recording, it was converted into an mp3 file compatible with the LEGO MINDSTORMS Education EV3, and from there it was uploaded and converted into a .rsf file. Then, it was exported into the sound files of the EV3 System Files and downloaded to the robot using the File Management Utility. The function serves as a key aspect of the project as it alerts the user after every time interval. This is important because in most cases, the user would probably forget about medication times or even delay them. The code of this function ensures that the high-pitched, aggravating sound would loop until the enter button was pressed. This was implemented by using a while loop where while the enter button was not pressed, the sound would loop, and once it was pressed all sounds would be cleared. This was tested several times without the main code to make sure that the words were audible to the user since the robot's speaker was muffled. Also, there were several tests to ensure that the sound would loop consistently until the user pressed the enter button.

*Table 1: Function Testing*

| | Tests | Cause | Expected Behavior | Program Functioned Correctly |
|---|---|---|---|---|
| `ReadFromFile` | 1. Initializing arrays<br>2. Reading Text file<br>3. Storing values into parallel arrays<br>4. Outputting Text file<br>5. Outputting values from parallel arrays | 1. Test if possible to pass arrays by reference in parameters<br>2. Make sure EV3 reads the text files accurately<br>3. Being able to store values into parallel arrays using the function<br>4. Outputting the correct values from the text file<br>5. Outputting values from parallel arrays that were stored using text file | The function should read the values from the text file and store them in parallel arrays. | All values from the text file were read in order by the robot and stored in the correct orientation in the parallel arrays. Outputting the parallel arrays and comparing them to the text file made sure the program functioned correctly. |
| `clipToBottom` | 1. Angle the motor would turn<br>2. The margin of error | 1. Make sure the angle the motor turns is accurate every time<br>2. Understand after how many tries would the motor not turn the same exact angle using the openAndClose function | The function is to ensure that the motor doesn't overshoot when it rotates at a specific angle. Therefore, it utilizes a proportional ity constant. | Using the openAndClose function, the angle at which the motor would turn the platform was exactly as much as inputed into the function. |
| `openAndClose` | 1. Angle the motor would turn<br>2. Angle every motor would turn is equal<br>3. The drag after several repetitions of the function | 1. Make sure that the angle at which the motor would turn is the same so only one pill would fall<br>2. Make sure every motor would turn at the same angle to | The function is expected to turn the motor, which the trapdoor is attached to, several times | Function would open and close a specific amount of times, according to how many pills should be dispensed. Each motor opens and closes the required amount of times and dispenses the number |

| | | | | |
|---|---|---|---|---|
| | | minimize errors in dispensing<br>3. Minimize the amount of movement of the platform when the motor moves back and forth several times | according to the Dispense function. The angle at which the motor opens should be almost perfect every time to ensure a singular pill would dispense. The platform was adjusted using a beta code that would move the platform left and right with the press of the left or right buttons on the EV3 to minimize external force on the motor. | of pills correctly with minimal to no error. |
| `Dispense` | 1. The correct number of pills dispensed each time<br>2. The motor was assigned to the proper coincident container<br>3. The function would work efficiently with every round of pills | 1. To make sure that the dispense function is dispensing the right amount of pills by measuring how many times the trapdoor opens and closed<br>2. Taking note of which motor was assigned to each container by measuring the number of times the openAndClose function was called | The Dispense function is supposed to assign each motor to a container and then use a for loop that would call the openAndClose function a certain amount of times for every motor to dispense the supposed | With every round of pills, the Dispense function would call the openAndClose function the right amount of times because the trapdoor would open and close the exact amount of times as per the text file. |

| | | | | |
|---|---|---|---|---|
| | | 3. Make sure that every time a new round of pills was being dispensed, the correct amount of pills were dispensed from the proper motor | number of pills. | |
| `moveElevationSensordist` | 1. The sensor moves the correct distance between each container<br>2. The sensor stops at every pill container | 1. To make sure that the sensor would move the correct distance and not stop too short of the container or after container<br>2. To make sure that the color sensor would stop at every container and be able to detect color so the color sensor has to be in the exact position in front of the pill container | The function is expected to move the color sensor down the linear slider a specific amount for each pill container and stop at each container. After the last container, the color sensor should move back up the linear slider. | When all the pills have been dispensed, the function moves the color sensor down the linear slider a specific distance and pauses in front of each pill container allowing the color sensor to detect the color of the pills. After the color sensor has moved to each container and detected the color of the pills, the moveElevationSensordist moves the color sensor back up the linear slider to its starting position |
| `checkPillLevel` | 1. Check if the color sensor is able to read a color<br>2. Check if the function accurately returns true when there are no pills in the container | 1. To check if the color sensor is placed in the correct position to accurately pick up any color<br>2. To make sure that when the color sensor picks up a color it returns false meaning that there are enough pills in the container and vice versa if there are no pills | The function is expected to have the color sensor detect the color of the pills and if the color sensor is able to detect the color of the pill, then the function would return false. If the color sensor was unable to detect the | As the color sensor moves between each container using the moveElevationSensordist it detects the color in the container and accordingly, it returns either true or false. If the color sensor would return true, meaning that there are no pills left in the container, then the next part of the code will take action which is the refillPillAlert. |

| | | | | |
|---|---|---|---|---|
| | | | color of the pill, then the function should return true. | |
| `refillPillAlert` | 1. Check if the sound in the function is properly playing and audible to the user if the pill level returns true<br>2. Check that the sound loops while the enter button is not pressed and is stopped when the enter button is pressed | 1. To make sure that the sound was properly playing in the function, the group used a basic built-in function called playSoundFile() to test the functionality of the sound. When the sound would play after installing all the files onto the robot, the group altered the volume of the sound and length to make sure it was long enough. Then tested the function to make sure that when the pill level returned true, the sound would play.<br>2. To make sure that the sound loops when the enter button is not pressed, the group placed the playSoundFile() inside a while loop so that while the button is not pressed, the sound loops. When the button is pressed, the sound stops playing, and is completely cleared. | The function works properly when the pill level returns a value of true, and the function enters a while loop where if the button is not pressed then the alarm will be played till someone presses the enter button. | When the color sensor does not detect color in one of the pill containers and the checkPillLevel function returns a boolean value of true, the refillPillAlert will enter a while loop that will alert the user with an alarming sound to inform the user that one of the pill containers needs to be refilled. To stop the sound from infinitely looping, the user would have to press the enter button for the sound to be cleared. |

| | | | | |
|---|---|---|---|---|
| `MedicationTimeALert` | 1. Check if the sound in the function is properly playing and audible to the user<br>2. Check that the sound loops while the enter button is not pressed and is stopped when the enter button is pressed | 1. To make sure the sound was properly playing, the group recorded a phrase to be downloaded into the robot and played when it was time for the user to take their pills.<br>2. After the time interval has passed and the Dispense function has been calledk, the function will play the sound alerting the user that it's time for the next round of pills. This alert will loop until the button is pressed by the user. To test this, the group individually tried the function on its own and made sure that the alert will loop until the enter button is pressed. | The function is supposed to play the phrase "Please come take your pills, make sure to press the button when you're done" when the time interval stated in the text file has passed and all the pills have been dispensed. The phrase will loop until the user approaches the robot and presses the enter button. | The function plays the alert after each time interval has passed and the Dispense function has dispensed all the pills into the cup.The alert loops perfectly as the phrase prompts the user to press the button to ensure that the user has taken the pills. |
| `Main` | 1.Checking that the dispenser waits the correct amount of time before dispensing for the following time that day.<br><br>2. Making sure that the main calls the dispense function the correct number of times based on info received from the files (basically, making sure that the correct number of pills are dispensed. | 1. So that the user takes pills at the correct time of day. Prevents the user from developing any medical complications that could be caused from taking the medicine at the wrong time.<br><br>2. This is also to make sure that the user does not take the wrong amount of pills | 1. That the dispenser waits the correct amount of time as indicated in the text file<br><br>2. The trap door opens the correct amount of times needed to dispense the proper amount of pills. | 1. A stopwatch was used to make sure the robot waited the correct time. It did.<br><br>2. As the code ran, the trap doors were observed, and they did indeed open and shut the correct number of times based on the text file.<br><br>3.The position of the colour sensor was observed and the colour sensor did indeed settle right in front of the pill after |

|  |  |  |  |  |
|---|---|---|---|---|
|  | 3. Checking that the moveElevationSensorDist is called with the correct distance.<br><br>4. Making sure the checkPillLevel is not called too early and that it works properly<br><br>5.Making sure the message that informs the user to refill a container is displayed properly and that the user has enough time to read the message before the program ends. | 3. To check that the function correctly determines if the container needs to be refilled.<br><br>4. If the checkPillLevel is called before the colour sensor is directly in front of the container, that will cause it to return a value of false, causing a prompt for the user to refill the container when it does not have to be refilled. Also making sure that when a value of false is returned, the container calls the medicationTimeAlert function and displays the message as it should and does nothing when a value of true is returned.<br><br>5. Making sure the message that appears on the screen is readable (it does not overlap with any other text) and that it disappears when the user presses the enter button to turn off the refillPillAlert. | 3. The colour sensor stops exactly in front of the container at the correct position<br><br>4. That the checkPillLevel is called at the correct time and therefore returns the correct value.<br><br>5. The message appears on the screen correctly | multiple rounds of testing.<br><br>4. When a container did not need to be refilled, a message was not displayed, and when a container needed to be refilled a message was displayed which indicates that the function worked correctly.<br><br>5. It was observed that the function worked properly. |

See *Appendix* for Demo Task List.

## Verification

For the checklist, criteria, and constraints, see *Appendix*.

During the demo, the group acted as a user, filled the Pillbot with skittles, and pretended to be a user. The robot's actions took place in the order below.

The button was pressed by the user, then the robot began dispensing the pills for the first round. Each trap door was opened and closed to dispense one pill per open and close movement. This was not the case. For one pill container, no pills were dispensed, for another, 3-4 were dispensed, and for another, more than 5 were dispensed for each open and close movement. All the pills dispensed dropped into the ramp and rolled down into the pill container for the user. The robot failed to dispense the correct number of pills, but the robot performed the correct number of open and close movements for each container. The robot then played the collect pill alarm until the user pressed the button.

After the correct amount of time for the interval between the first and second medication time, the robot began dispensing the second round of pills. Like round 1, the correct number of pills were not dispensed, but the correct number of open and close movements were performed for each pill container. The sound alarm then played until the user pressed the button.

The robot waited the correct time for the interval time specified in the text file between rounds 2 and 3, then began dispensing round 3 of pills. One of the containers was empty, due to the robot dispensing too many pills from that container in previous rounds. Another container dispensed no pills because they were stuck in the pill container. Like rounds 1 and 2, the correct number of open and close movements were performed at each pill container. The robot then played the sound alarm until the user pressed the button.

After the user pressed the button, the linear slider moved to the first, second, then third pill dispenser. Two of the three pill dispensers were below the minimum level at the end, so the robot displayed a string to refill one of the pills containers and played the refill pill alert until the user clicked the button. Once the user clicked the button, the robot displayed a string to refill the other container until the user pressed the button. The linear slider then moved to its original position at top of the rack, and the program terminated.

Looking at the order in which events took place, the robot met all the points in the checklist and criteria, except one. The robot was supposed to dispense one pill for each open and close movement, however, it dispensed between 0 and 6 pills for each open and close movement. This clearly indicates that the robot was unable to dispense the correct number of pills

The constraints say that the trap doors have various speeds, the pill containers have folds in the plastic where the pills can get stuck, and the motor can be inaccurate. As a result of these constraints, issues with dispensing the correct number of pills were anticipated by the group. During testing, these issues were identified, but the group was not able to fix these issues through rebuilding the pill containers or modifying the motor speed and angle. It was clear that after facing issues with the motors, cardboard and hot glue interface, and pill containers, the group needed to create a new and more precise design for the pill dispensing mechanism.

The group was not able to fix the issue in time for the demo, but concepts to address this issue were created, as seen in the *recommendations* section. A circular and flat 3D or cardboard piece could be attached to each pill dispenser's motor. This circular piece would have contained holes that are the size of the pill used. The motor would rotate at a slow speed, and when the hole in the flat and circular piece lines up with the hole in the pill container, a pill is dispensed from the pill container. Different motor speeds could be tested to ensure only one pill is dispensed.

An alternate concept is to create a pill dispensing mechanism similar to that of a bouncy ball or bubble gum dispensing machine, where the pill is collected by a spinning gear with large teeth, and this gear spins each time the button is pressed. Once the gear is filled, each time the button is pressed, the gear rotates, and the opening at the bottom of the gear causes only one pill to be dispensed. This is more advanced and would require 3D printing but would likely create a more precise pill dispenser since the dispenser is not relying on the speed of the motor and the timing of an opening in the pill dispenser hole to dispense a pill.

## Project Plan

For making the cardboard chassis, each group member chose a certain aspect of building the chassis to work on. When one person finished their task, they would help someone else to finish theirs, or discuss with the group what they could work on next to make themselves productive. For example, when making the cardboard chassis, Aly measured and drew on the large pieces of cardboard, Bishoy and Uday cut the pieces, and Ilan hot glued them together.

When creating the software, each person was responsible for a different:
Ilan - read from file function, dispense pills function
Uday - open and close function, move elevation sensor function
Aly - check pill level function, refill pill alert function
Bishoy - medication time alert function with custom sound, creating text file

Each person was responsible for coding and testing each of their functions, then later, the functions were combined into the same source code and used to create the main code. The plan was to work together on the main code, however since the group decided to create the linear slider late into the project, the functions needed to be coded after the linear slider itself. Aly and Bishoy focused on putting all the other functions together and creating a working main code, while Uday and Ilan built the linear slider. Then, the linear slider functions were tested individually, then added to the main code that contained all the functions.

When the functions had been put together and a task main was created, there was debugging in the task main to be done, and there was inconsistency with dispensing the correct number of pills. This was not according to plan, however, this was split evenly, where Aly and Bishoy worked to debug the code, and Ilan and Uday worked to rebuild 2 of the pill containers and modify the trapdoors.

*Table 2: Initial Project Plan*

| | Task | Date |
|---|---|---|

| | | |
|---|---|---|
| 1 | Collecting necessary items needed for dispenser | Friday, Nov 4 |
| 2 | Building Cardboard dispenser<br>Create slides | Monday, Nov 7 |
| 3 | Continued - Building Cardboard dispenser<br>Create slides for Formal Presentation<br>Start redesigning EV3 | Wednesday, Nov 9 |
| 4 | Design code and flowchart<br>Finish software design document | Monday, Nov 14 |
| 5 | Testing and tweaking the project,<br>Start Final Report | Wednesday, Nov 16 |
| 6 | Demo day | Friday, Nov 25 |
| 7 | Final Report | Dec 6 |

This was the initial project plan that was made for the informal presentation. The group expected to meet Monday and Wednesday evenings up until the report, and possibly add one or two sessions before the demo day for testing.

There were many revisions to the project plan. The group decided to add the linear slider and the alarms the day before the software design meeting, which was on Tuesday, November 15th. After the software design meeting, the group met multiple times to build the linear slider and could not decide on a physical design to use for the pill-level detector. Initially, a belt was considered, then after a few days, a new solution was found, and the construction of the linear slider began. In the following days, the group began testing the detect pill level functions and tested the accuracy of the encoder on the linear slider. Since the group decided to add the linear slider after the software, more time was spent on the physical part of the robot to account for this addition to the design. Due to this, the group was behind in the software, and in the following week the group had to spend extra time to make up for the time lost from working on the linear slider.

The pill dispensing mechanisms were difficult to build successfully. The constraints of using cardboard, popsicle sticks, hot glue, and plastic bags made the pill dispenser unreliable, and the group ran into problems with the trapdoors. This led to new revisions in the project plan, by adding a session almost every day between demo day and the software design to formulate new ideas for the pill containers and pill dispensers, built the linear slider, and program and test the functions and main code.

The group vastly underestimated the required time for this project when the project plan was initially made. Closer to the demo day, the group met every day of the week to test and redesign the pill dispensing mechanisms and modify the main source code and functions. When the group decided to add the linear slider, time was spent building this, instead of time spent programming as was planned in the initial project plan. This made the group behind schedule and caused the testing to happen later than planned for. The group also found inconsistencies with the pill dispensing mechanism and pills getting stuck in the pill containers. More time was spent to address these issues, and time was also spent on programming in the last week before the demo.

Reflecting on the actual path and timing of this project, the group should have left more time in the initial project plan to fix unexpected setbacks in the physical design or code, and to fix any problems identified from testing. For example, the final project was unable to dispense one pill accurately (as addressed in the *recommendations* section), and with more time this issue could have been fixed. The addition of the linear slider to the project was not planned for in the initial project plan, but this also caused the group to be behind schedule for programming.

## Conclusion

The idea of this project stemmed from the desire to aid the elderly and ease the troubles they go through as they take their daily pills. The initial idea was to create a pill dispenser that would be operated by an automated robot to manually dispense pills using a robotic arm and pressable buttons. The robot would be following a colored tape and pressing the button to dispense a specific number of pills, which would've been determined from a text file, into a cup that would be also held by the robot. This seemed like a solid idea at the start, but as the group started building the dispenser and thinking of how to build the robot, they came to realize the process was slow and difficult. There were several obstacles that would arise such as the amount of force the robotic arm would have to apply to the cardboard button several times in one go. Not to mention the fact of how unreliable cardboard projects could be if not done correctly. This is why the project idea was altered to a whole new mechanism. Instead of having a robot that would manually dispense each pill by moving from container to container, the whole process would be motorized and automated to have the pills dispensed on their own.

The second version of the project plan was the automated pill dispenser. This idea essentially consisted of the same main ideas as the first idea but with a few major changes. The pill dispenser would still read values from a text file, store them into arrays, and dispense them accordingly across the span of a day. However, the second design of the pill dispenser would have motors at every pill container with trapdoors attached to them. These trap doors would open and shut fast enough to allow one pill to drop through. These pills would slide down a ramp and into a cup on the side, readily available to the user. This idea seemed much better as all the group had to focus on was ensuring that the trapdoor was quick enough to be able to dispense just a singular pill to have little to no error. Unfortunately, that was not the case. After several different trials of altering the code to change the angle at which the trapdoor opens, the problem was not with the trapdoor, but with the mechanical design of the pill containers themselves. Every time the trapdoor would open, several pills would drop at once. This was threatening to the whole aspect of the project as the main idea was for the dispenser to be completely accurate in dispensing the correct amount of pills every time. After hours and several different trials of changing the code and the mechanical design, the group was unable to fix that issue and accepted it as a failure.

After redesigning the whole project, the group came to realize that automating the dispensing highly simplified the project. Therefore, the linear slider was added to the project to increase complexity as well as add new functionality to the whole design. The linear slider is made of a laser-cut spur rack and attached to it is a color sensor. The purpose of the linear slider is to allow the color sensor to move between each container and detect if there is color or not. If there is no color in the pill container, that means that the pills are below the required amount and the user would be alerted to refill that specific container.

The program of the project was formatted to efficiently fulfill the purpose of the project and complete the requirements. There are eight functions, each of which has its own parameters and uses. To start off the program, there is a function to read from a text file that was formatted to include the time intervals of medication times throughout the day(scaled down to a few minutes for the purpose of the demo). This function reads the time intervals and number of pills for each container and stores them into parallel arrays. Then there is a function(clipToBottom) to ensure that the power to the motor does not get too small so that the motor would not overshoot in the open and close function. The open and close function uses a proportional constant alongside the clipToBottom function to open and close the trapdoor while keeping the angle at which the trapdoor turns constant. The dispense function is primarily used to assign motors to the pill containers and dispense a certain amount of pills according to the parallel arrays derived from the text files. Then, there is a function to move the linear slider between each pill container and allow the color sensor to detect if there is a color in the pill container. The value returned from the color sensor is then incorporated into another function, which returns a boolean value to indicate whether or not there are enough pills in the container or not. The final two functions are both alerts used to alert the user. The first function alerts the user when the pill container is below the required amount by playing a loud blaring alarm, which requires the user to press the enter button to clear. The second function is used after the time interval between the consecutive round of pills has passed and the pills have been dispensed. It plays a phrase telling the user to come take the pills and press the enter button to clear the alert.

Comparing the final project design constraints and criteria to the initial specified constraints and criteria, the main aspect of the project remains the same with just minor differences. The initial list of constraints/criteria includes aspects of the initial design which include the cardboard buttons, robotic arm, and placement accuracy of the cup. In the final design of the project, these constraints were eliminated since the dispenser was automized. Instead, new constraints included the accuracy of the color sensor, the accuracy of the motor encoder, and the stability of the frame that was holding the linear slider. In the end, the project was able to meet the criteria as it dispensed pills at 3 different times with the time intervals being accurate and the alerts being accurately placed. The only criterion the final design did not meet was the number of pills dispensed each time. More pills were dispensed than set in the text file, which resulted in failed accuracy of pill dispensing. To prevent this failure, the final product could've utilized a 3D-printed container that was tailored to the size of the pill to ensure that a singular pill would fall into the container and then be dispensed. This would've been a much more accurate method of dispensing the pills instead of using the trapdoors. By increasing the scope of the project, the final design could be altered so that it is tailored to be more user-friendly, efficient, and accurate when viewed from an industrial standpoint so that it fulfills the requirements and is fully functional to the user.

# Recommendations

## Mechanical Design

For the group's software, changes would be made to move conditions from the main code to the functions. For example, in the alarm functions, a while loop was used in task main, which called the function to play sound, then once the user pressed the button, the code would exit the while loop, stop calling the function, and hence stop playing the sound. There were multiple instances of this in the main code, where while loops could have been moved inside the function to increase efficiency and cleanliness.

At the end of the day, when the robot had dispensed the last round of pills for the last medication time, the linear slider would move to each container and check if there were pills at the minimum level. Then, after checking the last pill container, it would display an alert to the user to refill the first container it noticed was below the minimum level and play the refill pill alert until the user pressed the enter button. If there was more than one pill that the sensor detected had no pills at the minimum level, then after the user pressed the button to acknowledge the refill alert for the first container, a new alert and string would be displayed. This was accurate; however, it was inefficient. To make this more efficient, a string to indicate all the pill containers that needed to be refilled and only one pill alert should have been used instead of an individual message and pill alert for each container.

The group failed to meet the criteria for the pill dispenser, because they were not able to dispense one pill at a time. To fix this, there are two concepts that can be attempted:

Instead of using a single rectangular piece that covers the whole of the container, a potential solution is a circular piece with holes in it that turns at a speed to allow only one pill to fall through. The holes could need to be customized to the size of the candy used, and could be built out of lego, cardboard, or 3D printed. A 3D printed would be the best option because the hole size would need to be accurate for the pill dispenser to be accurate. This piece could then be attached to the motor, and by testing different motor speeds, the best speed could be found so that the opening is below the hole in the container long enough for only one skittle to fall through. This could be successful but could also run into problems with pills getting stuck.

Another concept the group could use to solve the issues with the pill dispenser is to create a dispensing mechanism that works similar to a bubble gum machine dispenser. The pill would fall directly into a frame that holds only one pill at a time, then when the motor turns, the frame rotates, and one pill falls out of the frame, while another frame is filled with a pill. This could be 3D printed, and this mechanism would increase the precision and consistency of the pill dispensing mechanism, which is necessary for dispensing a single pill.

The motors of the Pillbot were mounted to the chassis with tape, and this was not a secure method.The dispensing mechanism used a rectangular piece which covered the trapdoor and moved at a high speed to expose and cover the opening. During testing, the group noticed the motor would wiggle against the cardboard it was taped to. When using a cardboard chassis, two options are to use glue or tape. A plastic or wood chassis could have allowed for a more secure method for mounting the motors, such as creating a fitted metal frame for the motors and drilling this frame into the wood chassis.

The pill containers were made using Ziploc bags, hot glue, and popsicle sticks. The skittles would rub against the hot glue and get stuck, and some parts of the plastic had stretched pockets, leaving undesired extra room in the single file container. Ideally, more precise and fitted material could be used for the containers, such as 3D-printed plastic. The clear front part of the containers could be made from acrylic glass, or some type of net instead of plastic to prevent the skittles from getting caught in stretched plastic pockets.

The linear slider frame was not attached to the cardboard chassis. There were markings on the cardboard chassis to indicate the correct placement of the frame, but if the frame was slightly misplaced, the color sensor of the linear slider bumped into the containers. This would cause damage to the linear slider or containers if left unattended. To increase reliability, the linear slider frame could be connected to the cardboard chassis. Using a quick and unsecure method such as tape would not suffice, rather, another frame made out of aluminum extrusions could be created for the base of the cardboard chassis. This frame could then be connected to the linear slider frame.

## Software and Industry Recommendations

If the Pillbot were to be used in industry, the ability to edit the medication times, the number of each pill per medication time, and wait times between each medication time would need to be more convenient to edit for the user. Instead of editing the text file, downloading it to the EV3, then going into the RobotC program and deleting the old file, another program could be created to rewrite the text file, where the user could connect their robot to their computer, and through some application or by running another program and inputting their desired changes, they could change the text file.

If the Pillbot was used by consumers such as seniors, caretakers, and doctors, the criteria and constraints would change. For example, there would need to be many more than 3 pill containers, and therefore more motors. A high number of pill containers would create high demand for the linear slider to move between each pill, therefore a new system to check the pill levels, such as individual color sensors, or using mirrors within the pill containers could be implemented. Using one linear slider to check the pill level for more than 10 pills could force the chassis of the dispenser to be long and inconvenient for practical use. Other sensors could be used to check the pill levels, such as a weight sensor. This could reduce errors in color sensors and provide flexibility to use different pills with different sizes and weights. If a weight sensor was used, then the software could have constants that store the weight of one pill in a specific container. The minimum level of pills could be detected by using a boolean function to detect if the weight in a certain container is above a certain level, rather than using color sensors. Then, if the pill used in a certain container was changed, this constant could be modified.

Since different pill sizes could cause the pills to get stuck in the pill dispensing mechanism, the pill dispenser would need to be able to handle various pill sizes. To address this, interchangeable hole and container templates could be used, with a large base container and smaller frames that could be added on to be used with smaller pills. If a 3D printed dispensing piece were used, a large base piece could be used, and smaller templates could be 3D printed. This would allow for quick customization to allow for a container and pill dispensing mechanism to work with a different size pill, provided that the pill is smaller than the largest template size.

To make the software accommodate more pills and pill containers, more parallel arrays would need to be used, and increase the number of containers to check for the pill level checker. If parallel arrays become an unrealistic method of storing data from the file, object-oriented programming could be used. For example, if there are 40 different pills, it may be unrealistic to use 40 different parallel arrays. In this situation, a new method of reading and storing the data from the text file could be devised. If a linear slider was still used where there are multiple pills, the constants for the number of encoders counts between each container would need to be checked.

For the physical system, to make the project more robust and industry-ready, other materials could be used instead of cardboard, such as plastic, wood, or metal. The Pillbot prototype was unnecessarily large considering it only dispensed 3 different pills, so the size of the project would need to be reduced for it to be consumer friendly. If used in industry, the Pillbot should be modified to consider human factors design, as a user would interact with the Pillbot multiple times a day, and designing the user to be more practical, robust, and aesthetic would elevate the user experience.

## Bibliography

Builderdude35. (2016, January 7). *How to Make Custom Sound Files in EV3-G*. Retrieved from Youtube: https://www.youtube.com/watch?v=kv_ib_x5IT8&feature=youtu.be

Hadi, D. (2017, September 24). *How to build a high peformance LOGO slider*. Retrieved from Youtube: https://www.youtube.com/watch?v=J249acUk7tY

hundredvisionsguy. (2016, June 1). *RobotC and EV3 Sensors and Loops Part 1*. Retrieved from Youtube: https://www.youtube.com/watch?v=kjoKC0uWtTo

## Appendices

### Appendix A – Checklist Used on Demo Day

Checklist:

- Enter button is pressed to start the program
- The robot reads the pill quantities for each pill time, and the time intervals from the text file
- The robot starts to dispense pills based on info from the text file
- After dispensing a round of pills, sound alert plays until user presses button
- After each round of pills, the robot waits the amount of time specified in the text file before dispensing the next round of pills
- After button is pressed after the last round, linear slider checks pill levels for each container
- Displays messages on screen for the pill containers that need to be refilled.
- Sounds alert plays until the user presses the button
- Waits 10 seconds
- Program ends

Criteria:

- Robot waits for user to press button before dispensing first round
- Robot dispenses the correct number of pills
    - Round 1: 1 of red1, 3 of yellow, 2 of red2
    - Round 2: 3 of red1, 3 of yellow, 3 of red2
    - Round 3: 2 of red1, 1 of yellow, 3 of red2
- Robot waits the correct amount of time
    - Between Round 1 and 2: 30 seconds
    - Between Round 2 and 3: 20 seconds
- Robot plays the customized sound if the user does not press the enter button
- After all the pill times, the tells the user which pill containers are at a low level and need to be refilled.

Constraints:

- Color sensor can only detect red and yellow skittles consistently
- Color sensor can be inaccurate, leading to incorrect refill pill alerts
- Clear plastic parts of the container have folds, so pills can get stuck
- Trapdoor speed varies with cardboard, so inconsistent number of pills dispensed
- The motor can be inaccurate due to high speeds and short movements, leading to motor drift

### Appendix B - Source Code

```
#include "PC_FileIO.c"
```

```
void ReadFromFile(TFileHandle & fin, int* pill_1, int* pill_2, int* pill_3,
float* wait_time_hrs);
float clipToBottom(float powerToMotor);
void openAndClose(int motor_num, int angle);
void Dispense(int container, int num_pills);
void moveElevationSensordist(int motor_power, float dist_cm);
bool checkPillLevel(int colour);
void refillPillAlert(bool pill_level);
void MedicationTimeAlert(int time);

const int PROPORTIONAL = 1.0/45.0;
const int ANGLE = 20;
const int NUM_ELEMENTS = 3;
task main()
{


            TFileHandle fin;
            bool fileOkay = openReadPC(fin, "pills_to_dispense.txt");

            displayString(2, "Press enter to begin");
            //Wait till Enter button is pressed
            while(!getButtonPress(buttonEnter))
                  {}
            while(getButtonPress(buttonEnter))
                  {}


            //File Input

            //declaring the arrays
            //Also, Populating the arrays manually:
            int pill_1[NUM_ELEMENTS];
            int pill_2[NUM_ELEMENTS];
            int pill_3[NUM_ELEMENTS];
            float wait_time_hrs[NUM_ELEMENTS];

            //Populates Arrays with info from files

            ReadFromFile(fin, pill_1, pill_2, pill_3, wait_time_hrs);




            //For loops to dispense pills

            for(int times = 0; times < 3; times++)
            {
                  float time_in_Hrs = wait_time_hrs[times];  //We will
multiply num hours by 10 seconds just for the purposes of the demo
                  time1[T1] = 0;
                  while(time1[T1] < time_in_Hrs * 10000)
                  {}
                  Dispense(1, pill_1[times]);
                  Dispense(2, pill_2[times]);
                  Dispense(3, pill_3[times]);
                  MedicationTimeAlert(time_in_Hrs);
                  //In this function it should wait until a button is
pressed                                                               for
the beeping sound to stop.
            }

            //booleans for refilling the containers
            bool refill_container_1 = false;
            bool refill_container_2 = false;
            bool refill_container_3 = false;

            //configuring the color sensor
            SensorType[S1] = sensorEV3_Color;
            SensorMode[S1] = modeEV3Color_Color;
            wait1Msec(100);
            //Moving the colour sensor up and down to check the level of
pills
            refill_container_1 = checkPillLevel((colorRed));
            moveElevationSensordist(20, 13);
            refill_container_2 = checkPillLevel((colorYellow));
            wait1Msec(3000);
            moveElevationSensordist(20, 12);
            wait1Msec(3000);
            refill_container_3 = checkPillLevel((colorRed));;

            //Displaying messages and sounding alerts for containers that
need to be refilled
            if(refill_container_1)
            {
                  displayString(5, "refill pills in container 1");
                  refillPillAlert(refill_container_1);
            }

            if(refill_container_2)
            {
                  displayString(10, "refill pills in container 2");
                  refillPillAlert(refill_container_2);
            }
            if(refill_container_3)
            {
                  displayString(15, "refill pills in container 3");
                  refillPillAlert(refill_container_3);
            }

            moveElevationSensordist(-20, -25);



            wait1Msec(10000);

}

void ReadFromFile(TFileHandle & fin, int* pill_1, int* pill_2, int* pill_3,
float* wait_time_hrs)
{
int time_temp1 = 0, time_temp2 = 0, time_temp3 = 0;

readIntPC(fin, time_temp1);
readIntPC(fin, time_temp2);
readIntPC(fin, time_temp3);

wait_time_hrs[0] = time_temp1;
wait_time_hrs[1] = time_temp2;
wait_time_hrs[2] = time_temp3;

for(int count = 0; count<3; count++)
      {

      int temp1 = 0, temp2 = 0, temp3 = 0;
      readIntPC(fin, temp1);
      readIntPC(fin, temp2);
      readIntPC(fin, temp3);

      pill_1[count] = temp1;
      pill_2[count] = temp2;
      pill_3[count] = temp3;

      }
}


//function for pidf
float clipToBottom(float powerToMotor)
{
      if(powerToMotor < 0.1)
      {
            return 0.1;
      }
      else
      {
            return powerToMotor;
      }
}

//Open and close function
void openAndClose(int motor_num, int angle)
{
      nMotorEncoder(motor_num) = 0;
      while (abs(nMotorEncoder[motor_num]) < angle)
      {
            float error = abs(angle - nMotorEncoder[motor_num]);

            float motorPower = clipToBottom(error*PROPORTIONAL)*1000;

            motor[motor_num] = motorPower;
      }
      Motor[motor_num] = 0;
      nMotorEncoder(motor_num) = 0;
      while (abs(nMotorEncoder[motor_num])< angle)
      {
            float error = abs(angle - nMotorEncoder[motor_num]);

            float motorPower = clipToBottom(error*PROPORTIONAL)*100 + 100;

            motor[motor_num] = -motorPower;
      }
Motor[motor_num] = 0;
}

//dispense function
void Dispense(int container, int num_pills)
{
      int motor_name = 0;
      if(container == 1)
      {
            motor_name = motorA;
      }
      if(container == 2)
      {
            motor_name = motorB;
      }
      if(container == 3)
      {
            motor_name = motorC;
      }

      for(int num_time = 0; num_time < num_pills; num_time++)
      {
            openAndClose(motor_name, ANGLE);
            wait1Msec(500);
      }
}

//function that moves elevation sensor specific distance
void moveElevationSensordist(int motor_power, float dist_cm)
{
nMotorEncoder[motorD] = 0;
motor[motorD] = -motor_power;
float const RADIUS = 1.75;
int const ENC_LIMIT = dist_cm*180/(PI*RADIUS);
if(dist_cm > 0)
{
      while (abs(nMotorEncoder[motorD]) < ENC_LIMIT)
      {}
      motor[motorD] = 0;
}
else if( dist_cm < 0 )
{
      while (abs(nMotorEncoder[motorD]) < abs(ENC_LIMIT))
      {}
      motor[motorD] = 0;
}
return;
}

//Check pill level function
bool checkPillLevel(int colour)
{
      if(SensorValue[S1] == colour)
      {
            return false;
      }
      else
      {
            return true;
      }
}

void refillPillAlert(bool pill_level)
{

      if(pill_level == true)
      {
            while(!(getButtonPress(buttonEnter)))
            {
                  playSoundFile("alarm");
            }
            while(getButtonPress(buttonEnter))
            {}
            clearSounds();
        }
        return;
}

void MedicationTimeAlert(int time)
{


        while(!getButtonPress(buttonEnter))
        {
            playSoundFile("pills_reminder_2");
        }
        while(getButtonPress(buttonEnter))
        {}
        clearSounds();


}
```