Final Report - Skittle Sorter

Group # 468

Jessica Ramseyer

Brandon Tu

Ali Arshad

Omar Abu Abah

Contents

Table of Figures iii
Table of Tables iii
Report Summary iv
Introduction iv
Background information v
Scope vi
Main functionality vi
Start-up: vi
User Specifications: vi
Sorting Process: vi
Skittle Transport and Motors: vi
Shutdown: vii
Changes: vii
Constraints and Criteria viii
Mechanical design and Implementation ix
Design Description ix
Mechanical design xii
Criteria and general design decision xiv
Four disk design choice xiv
Skittle candy choice xiv
Skittle pack choice xv
Mechanical design decisions xv
Sorting disks xv
Leading arm xv
Three colours with a scrap bin xv
Single colour sensor decision xv
Problems xvi
Putting in spacers between disks xvi
Colour sensor reading through acrylic xvii
..... xviii
Shaker piece needed to be changed in order to function properly xviii

Software design and Implementation xix
Overall Program Design xix
Functions xx
Data Storage xxii
Trade-offs of Software Design Decisions xxii
Testing Process: xxii
Resolution of Problems: xxiv
Verification xxiv
Project Plan xxvi
Conclusion xxvii
Recommendations xxviii
References xxix
Appendix A: xxx
Appendix B: xxxv

Table of Figures

Figure 1: The final assembled robot x
Figure 2: Skittle guiding piece xi
Figure 3: Leading Arm xii
Figure 4: Assembly pieces of the four-disk design xiii
Figure 5: Disk system xvi
Figure 6 Disk System with Spacers xvi
Figure 7: Colour sensor and acrylic sheet xvii
Figure 8: Colour sensor without acrylic sheet. xvii
Figure 9: Shaker piece in lower position xviii
Figure 10: Raised shaker piece xviii
Figure 11: Flowchart for colourReading function xxxv
Figure 12: Flowchart for userInput function xxxvi
Figure 13: Flowchart for rotate function xxxvii
Figure 14: Flowchart for openChamber function xxxvii
Figure 15: Flowchart for display function xxxix
Figure 16: Flowchart for turn_arm function xl
Figure 17: Flowchart for zeroArm function xli
Figure 18: Flowchart for CalibrateMotors function xlii
Figure 19: Flowchart for skittleCounts function xliii

Table of Tables

Table 1 - Comparison of Mechanical Designs xiv

Table 2 - Function Descriptions.....xx
Table 3 - Average of 15 colour readings for different skittle colours xxii
Table 4 - Possible Colour Readings xxiii
Table 5 - Time to sort skittles and accuracy for different numbers of colour readings taken xxiv

Report Summary

The following final design report will discuss and examine the benefits of the creation of a mechatronics system that is able to sort a sample of skittles based on their colour. The foundational design and action plan of the robot that accomplished this goal will be outlined. The scope of the report will encompass both the problem analysis and description, followed by a description of the exact constraints and requirements that the robot must fulfill.

Followed, the mechanical and software designs will be further explained and analyzed in depth with an explanation of their respective method of implementations. This section will go over all the various mechanical components of the robot such as the chassis design, motor drive design, sensor mount design, and finally a brief explanation of any mechanical trade-offs that we were required to face. For software, the report will go over how tasks were broken up into smaller functions. A description of each function and the overall main program will be provided. Explanations as to how each one contributes to the robot's ability to complete its overall task in an efficient manner will be included.

In addition, the final report will include an overview of the project timeline. The completed tasks will be presented in a visual manner outlining the chronological order in which they were completed, while also relating each task to its respective team members. The plan will go over the timeframe for which the construction process of the hardware of the robot was completed, in addition to the programming of the software on which it will run. Doing so will allow for a good view of the amount of time that was spent on completing the robot and will aid for future project planning and delegation of tasks.

Finally, the report will conclude by going over and verifying all the requirements and constraints which were stated earlier. For any that were not met, potential reasons and obstructions will be stated and explained to understand what could have gone wrong and how it could have been done better. Lastly, the report will end off with potential design changes that could be made to the mechanical components and to the software programming which could conceivably improve the performance of the robot.

Introduction

For an individual with a food allergy, there is a constant danger of ingesting something which may be harmful to the body. One very common food related allergy, which occurs in about 4% of people, is a red food dye allergy [1]. If someone with a red food dye allergy finds themselves eating a bag of skittles, there is a high chance that they may encounter an allergic reaction as a result of the food dye in the candy. As such, this introduces a need for a solution which may help individuals prevent such a thing from occurring. [2]

The goal of our robot is to take an unsorted batch of skittles, differentiate them based on colour, then deliver them to a jar or container based on the colour and the preferences of the user. Sorting candy by

colour allows an individual to remove flavors they may not be fond of to improve enjoyment and can also aid someone with a colour dye allergy. By sorting the candy into different colours, the individual will be able to enjoy their skittles without the worry of an allergic reaction, while also adding a sense of fun within the process.

Stakeholders for such a product include those interested in a system to help them sort candy by colour, whether it be to simply arrange them for amusement purposes, or for preventing potential health hazards caused by food dye allergies. Individuals in either of these two groups would be able to positively gain from such a product.

The following paragraph is an excerpt from [2].

In addition to solving the initial problem introduced, working on this project will also help us reach our goal of getting experience working with and learning how to create mechatronic systems which use different sensors and motors simultaneously to solve the daily problems we face throughout our lives. Thanks to the hands-on component of this assignment, this project will likely also help give us a much deeper breadth of understanding of the design theories we have been learning throughout our MTE lectures, and the programming techniques taught in GENE 121.

Background information

The following paragraph is an excerpt from [2].

To best understand the nature of the problem at hand, there is some key information which must first be understood. In the case of our robot project, the given problem at hand involved the creation of a mechatronics system which could successfully differentiate objects (in this case skittles) based on colour, and then sort them into various containers of a user's choice. As such, a general understanding of how colour sorters function would be useful.

As stated above, the first part of a colour sorter mechanism, involves using a colour sensor to determine the colour of a given single object. To do this it is crucial that the given object is placed at a correct orientation such that there is a large enough face in the field of the sensor for it to scan.

The following paragraph is an excerpt from [2].

Secondly, it must be ensured that the mechanism which lets the object pass through into view of the colour sensor only allows for a single object (or skittle) through. Although at the surface this may sound simple, when working with objects as small as skittles, designing a mechanism to filter through only one skittle from a batch can easily become the hardest task; thus, different filter mechanisms were considered throughout the designing phase of the robot [3].

Scope

Main functionality

Start-up:

- Once the program has been started and the sensors have been configured, the motor shaking the skittle the motor with control of the leading arm are put into their proper starting orientation
- The piece that shakes the skittle and moves in order to drop it down to the leading arm is configured so that it fully blocks a skittle from falling through. This is determined from when the reading of the colour sensor changes from blue to white.
- The leading arm is rotated until it hits the touch sensor in order to put have a standardized starting position

User Specifications:

- Control buttons of the robot correspond with jars labelled 1 through 3.
- For each colour of skittle (i.e. Yellow, brown, blue) the screen displays a message on the screen reading “What jar would you like (insert colour) skittles to go to?”
- The user can then push the control button corresponding with the jar they would like the skittles of that colour to go to.
- This is repeated for all 3 colours of skittles.

Sorting Process:

The design of the sorting process did not change. The following section has been taken from [2].

- Skittles are collected one by one by the spinning sorting disk and dropped into another chamber where they can be read by the colour sensor.
- The colour sensor takes a reading of the colour of the skittle 15 times and the most common reading is determined.
- If this most common reading occurred 7 times or more it can be assumed to be an accurate reading. If not, it will be assumed to be inaccurate.
- The skittle is then dropped down to the leading arm which directs it to the appropriate jar as determined by its colour reading.
- If the skittle reading has been determined to be inaccurate it is dropped into a fourth jar meant for “junk”.
- This process is repeated until all the skittles have been sorted.

Skittle Transport and Motors:

The operation of the first motor did not change from the initial design. The following description of the functionality of the first motor has been taken from [2].

A motor is used to rotate 2 plates on the same axis simultaneously.

1. The sorting disk at the bottom of the holding chamber.
2. A disk further below that moves the skittle to the location of the colour sensor.

- A static disk exists between the two in order to ensure that only one skittle at a time passes to the colour sensor.
- The two non-static disks are out of alignment. As these two disks rotate back and forth skittles are passed from the top holding chamber into the chamber where the colour sensor is located one by one

The second motor in the robot opens a chamber that allows the skittle to pass from the colour sensor to the leading arm.

- The skittle is shaken and the colour of it is read asynchronously. This motor shakes the skittle back and forth between readings by moving side to side quickly in order to move the skittle into a slightly different position.

The final motor rotates the leading arm into the correct position in order to divert the skittle into the proper jar. The following section has been taken from [2].

- The colour reading of the skittle corresponds to a motor encoder value which marks the location of each jar.
- After delivering a skittle the leading arm stays in place and its current position will be kept track of.
- The way in which the leading arm needs to move in order to get into the proper position is then based on the position of the next skittles jar.

Shutdown:

- If 20 seconds passes in which the majority of colour sensor reading are white (or at least not yellow, blue or brown), it is assumed that all the skittles have been sorted.
- Then the motors turn off
- The program displays to the EV3 screen
    - How many total skittles were sorted
    - How many of each colour were sorted
    - The average time it took to sort one skittle

waiting 20s for a person to read it before ending the program.

Changes:

Five colours of skittles were intended to be sorted but the accuracy of the colour sensor proved to be an obstacle that prevented us from doing that. Upon testing the ability of the colour sensor to distinguish between red, orange, yellow, green, blue, purple and brown, it was found that yellow, blue, and brown skittles were the only ones that the colour sensor was able to detect consistent and correct readings for. Colours such as red, orange, purple and green could not even be picked up, instead giving wide ranges of readings that were not even correct.

The ultrasonic sensor was originally intended to be used to check whether the holding chamber has been filled with skittles in order to determine if sorting can commence. It was decided that this use of the sensor would be inefficient and did not contribute much to the overall performance of the robot. The ultrasonic sensor was instead replaced with the touch sensor which was used to zero the leading arm.

It was planned for the colour sensor to be taking readings of the skittle while it was being shaken. It was discovered that this was detrimental to the readings that the colour sensor was finding, causing it to consistently give incorrect readings. When the skittle was instead shaken between readings, the increase in accuracy was immediate and quite considerable. It was for this reason that this design change was made.

Constraints and Criteria

Note: These constraints and criteria were taken from the Preliminary Design Document [2].

Project Constraints

Constraint #1: Must be fewer than 10% wrong Skittles in Total

As the robot's main function is to successfully sort skittles by color, it should not be constantly putting Skittles in the wrong containers. A 10% tolerance for error will allow for errors in the color sensor. This is because the position of the skittle coming into the view of the color sensor cannot be controlled from our design [3].

Constraint #2: Must not sort fewer than one Skittle per 3 seconds

To further enhance the enjoyment of skittle consumption, the sorter should operate quickly and efficiently. To achieve this constraint, our robot will move its leading arm the smallest distance possible in order to deliver the skittle to the correct position

Project Criteria

Criteria #1: Candy is Not Damaged

The project's main goal involves enhancing the enjoyment of the Skittle eating process. This would be negatively affected if our sorter produced damaged candies. To assure the candies come out undamaged, we will chamfer the edges of the collection plates. This will remove any sharp edges that could possibly crack or break our skittles as the plates rotate.

Criteria #2: Robot Must Stop by Itself

To increase the independence of the device as well as the convenience for the user, the robot will be built to stop by itself. To achieve this, we will track the time after the color sensor has received a value other than the default wall color. After 20 seconds passes without any input, it can be assumed that there are no more skittles to be sorted.

Constraints and Criteria Overview

From the preliminary report, no changes were made to the constraints or criteria. The only changes made to the constraints occurred prior to the preliminary report, when the robot was originally supposed to sort no fewer than 1 skittle per second. This was mainly changed to allow for more readings per skittle and decrease the amount of skittles sorted into incorrect containers.

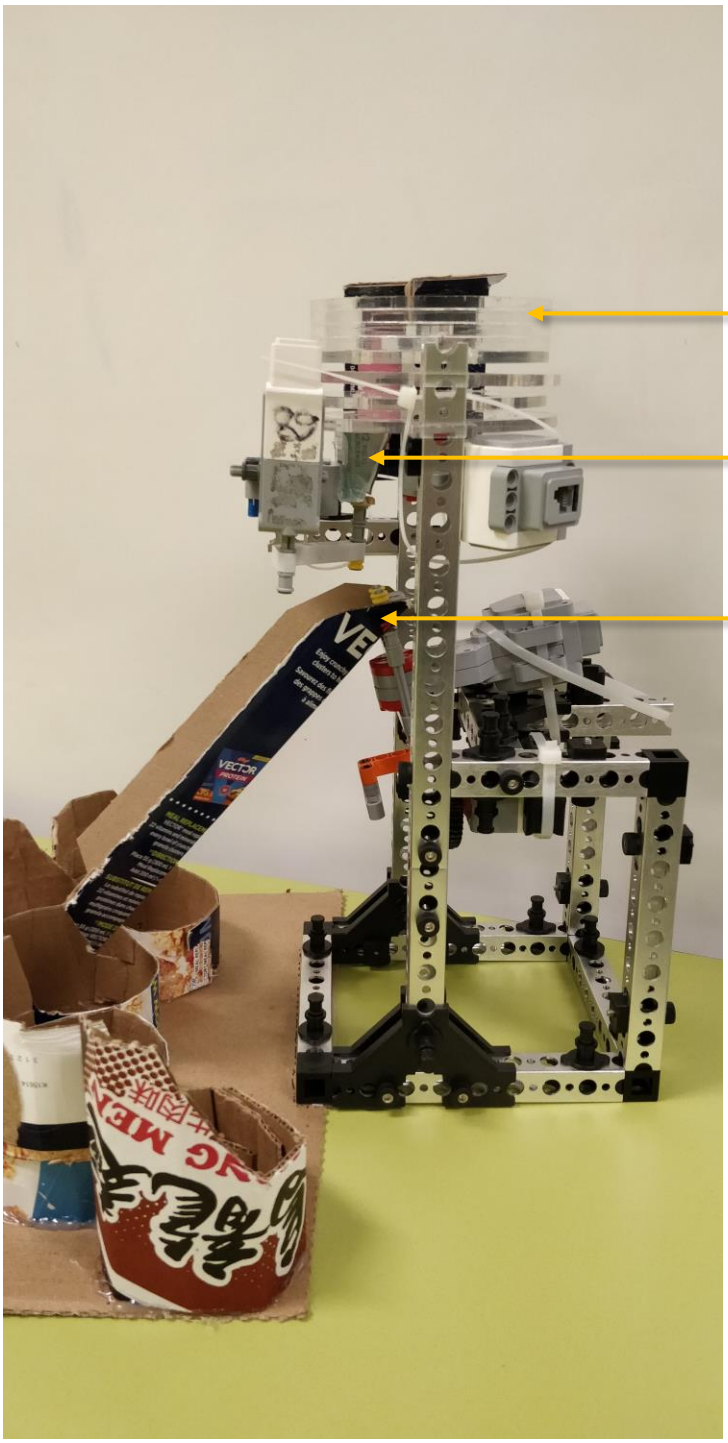
The most valuable constraint was the robot not sorting fewer than 1 skittle per 3 seconds. This set the pace for the entire project, both in terms of the mechanical system as well as the software. The design of the sorter had to accommodate for skittles being dispensed through the machine rapidly. This led to the creation of a rotating arm that would lead the skittle into its respective container before moving onto the next skittle. The software was also majorly affected by this constraint, especially when deciding on how many colour readings of the skittle to take. More readings meant that it took longer for each skittle to be sorted. The constraint also meant we had to track the time of the entire sorting process and count the number of skittles that passed through the machine to calculate the average time to sort one skittle.

The least valuable of the criteria was that the skittle had to be undamaged. This turned out to not be a problem, as the skittle was more durable than expected during testing. Even though the candy was being put through multiple drops into the sorting disks and containers, we never noticed any damage to the skittle's surface at all.
The criteria that required the robot must to stop on its own and the constraint to have fewer than 10% inaccuracy were just general guidelines that helped guide our design. They were not majorly impactful or detrimental as we ended up having a very high accuracy with sorting after selecting skittle colours that the colour sensor could read consistently. Similarly, making the robot stop on its own was a simple task of adding a timer that would stop the program once 20 seconds of no input occurred in the colour sensor.

Mechanical design and Implementation

Design Description

The skittle sorter is comprised of three main parts, the sorting disks, the colour reading tube, and the leading arm to sort the skittles into the appropriate containers. The overall design was dependent on the measurements of an average skittle being 8mm wide and 14mm in length.



Four disk design:
Skittles get poured in the top section of the plates, to be sorted through the multiple acrylic disks.

Colour reading tube:
The skittle gets turned onto its side, shaken, and its colour is read, through the colour

Leading arm:
The arm receives a colour and moves accordingly to the colour of the skittle.

Figure 1: The final assembled robot

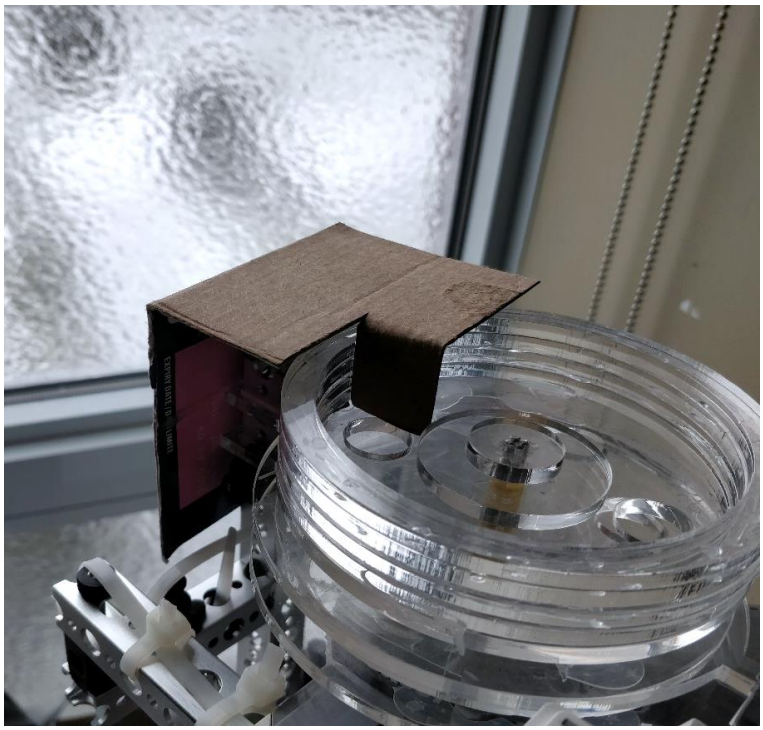


Figure 2: Skittle guiding piece

Firstly, the sorting disks consist of two static disks and two moving disks, which allow only one skittle through at a time. This is possible as the thickness of every disk is that of a skittle. The first disk has two holes which forces skittles down through by constantly rotating. In addition, there is an externally mounted piece that circulates the pieces down the hole so that the skittles don't sit on the top disk without going through. This is shown in Figure 2.

Once the first disk gets a skittle, it is sent down through the first static disk directly underneath it. The static disk then drops the skittle through to the second static disk, after about 30˚, and it is then deposited into the final static disk after a 180˚ turn and into the colour reading tube. The moving disks are offset to be 30˚ apart, and the static disks were offset 90˚ apart as to reduce the risks of jamming in the system. If the disks were not offset, there would been times when the three disks open at once allowing more than three skittles to fall through resulting in a jam.

The colour sorting tube is comprised of a narrowing tube and a shaking disk, which forces the skittle into an upright position. This is so that the colour sensor has access to a greater surface area as to gage a better idea on what colour the skittle is. When the skittle is in the upright position it gets shaken and read a total of fifteen times. For accuracy, the skittle goes through fifteen phases of shaking, stopping between each one to measure the colour. The colour is determined and then passed on to determine which jar the skittle goes into.

Figure 3: Leading Arm

The leading arm motor was mounted at an angle to better manipulate the travel arm position. This angle of the motor also allows for the motor to press the touch sensor and accurately zero its position. The arm has four destinations to turn to, with three being designated colour selections, and the last for unknown readings. The leading arm can be seen in Figure 3.

Mechanical design

The majority of the device was assembled using parts from kits, including the Lego EV3 kit and the Tetrix Prime Kit. Some parts however, have been custom made and designed, including those of the spinning disks, scooper, and the leading arm. The Lego and Tetrix kit were used to mechanically support these pieces into position.

Figure 4 shows the pieces of the four-disk assembly. Each part is numbered into a section.

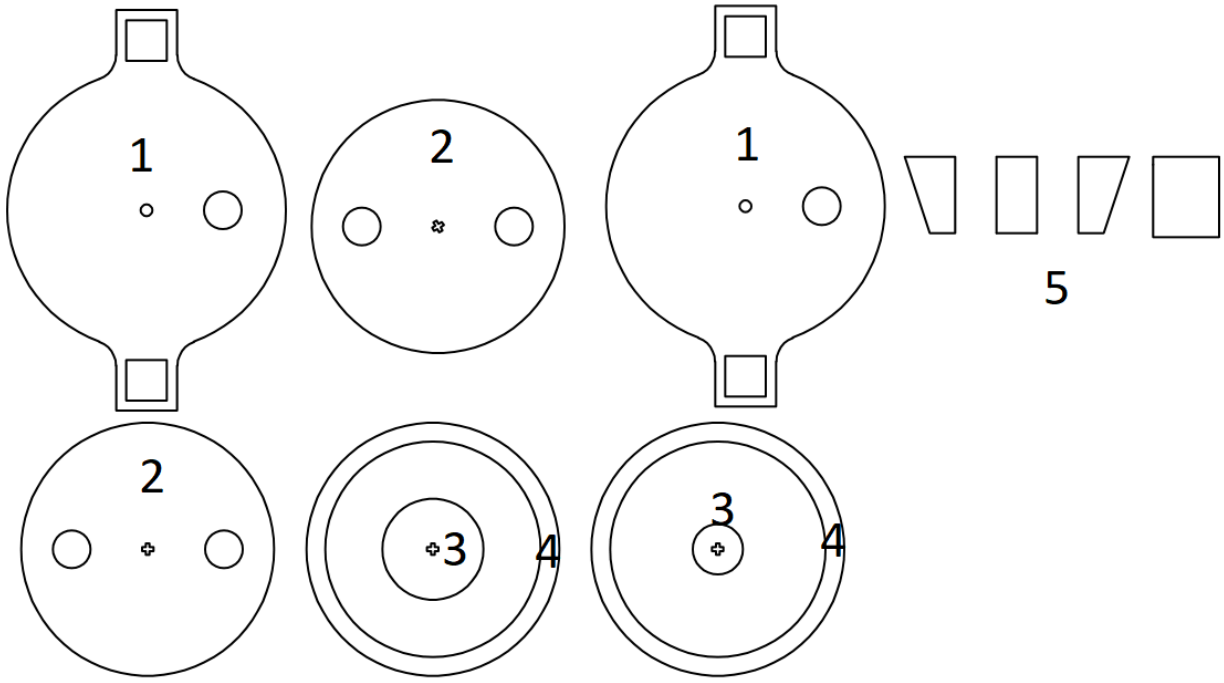


Figure 4: Assembly pieces of the four-disk design

1) These disks are static meaning they do not move and thereby are externally mounted for both a weight support and permanent positional alignment. They are aligned to the other disks concentrically through the hole in the middle which is slightly larger than the outer diameter of the shaft
2) These disks are the moving parts of this assembly. They have a center cut out to fit the exact motor shaft size to decrease the chances of slippage. These disks have two holes as opposed to one, to increase the overall flow throughout the system. Additionally, to avoid a jam, where a static disk and two moving disks open all at the same time, the fit of the shaft of the moving disks is offset by 30 degrees to avoid an open situation, which may cause a jam.
3) These small cut outs are used on top of the disks to push the skittles away from the center and to the sides towards the holes. These are used to avoid having skittles sitting in the top disk without being sorted.
4) The two outer disks were directly mounted onto the top plate to hold the skittles in, preventing them from falling off of the sides.
5) These pieces were used to assemble the colour sorting chamber. When assembled, the bottom opening is the width of a skittle in the upright position and the wider top opening is that of a skittle in the horizontal position.

The Lego EV3 motor shaft ran through all the pieces to keep them concentric and Lego spacers were used between each disk to reduce sliding friction. The Tetrix aluminum shafts were used to support the static disk into position, which had a very tight fit to reduce vibrations that could misalign the disks during operation.

In order, the 3's would go on the shaft first, then a moving disk (2), following that, a static disk (1), and then another moving disk (2). Finally, a static disk (1), with the colour reading tube attached (5).

Criteria and general design decision

Four disk design choice

Deciding which design to pick was based on many factors including speed, reliability, accuracy, and simplicity of the design. The following designs were the ones proposed from the preliminary design document [2].

Table 1 - Comparison of Mechanical Designs

	Speed	Reliability	Accuracy	Simplicity
Design #1	Speed at which the sorting is done is very high.	May result in many jams due to the tolerances	Not the most accurate as the skittle would lay directly on the colour sensor giving poor readings	The design was complex as it required a differential setup to power the main axial
Design #2	Acceptable speed, The system had to rotate at least 180˚ to cycle a skittle in.	The system is reliable	The colour sensor is accurate, but it may be hard to shake the skittle.	The overall design is simple, and can be easily made in the allotted time
Design #3	Acceptable speed, the system would only have to rotate 60˚ for a skittle to sort.	The system is reliable, may jam if not timed correctly or stopped mixing	The colour sensor is accurate, and it was easy to shake the skittle for better readings	The design was timely to manufacture as extensive 3D printing was required.

Based on a group decision design #2 was chosen as it did not have as large of barriers as the other two designs, mainly in the manufacturing process. The four-disk design has other advantages such as compatibility, and durability, which were not requirements but were contributing to the design decision.

The four-disk design was in fact easier to manufacture than expected. It only took a single day to cut and assemble the product. The reliability of the system also proved to be quite satisfactory. Even after hundreds of skittles going through, the system did not jam once on its own unless a human interfered with it. The design was also almost three times as fast as our sorting time requirement once again proving itself to be a solid design choice.

Skittle candy choice

The skittle was picked as the candy because of its smooth and strong surface as well as its variety of colours. The candy was chosen based on its ability to be read by the colour sensor. Its size is a major contributing factor in this regard, as a larger candy would be better read by the colour sensor than a smaller candy would. The candy also needed to be easily manipulated as it needed to be rotated into a

vertical orientation. Other candy was considered but were all deemed either too inconsistent with candy size, like the M&M peanut candy, or poor reading from the colour sensor, like the rockets candy. Skittles met these requirements the best and as a result was chosen as the candy to be sorted.

Skittle pack choice

The skittles that were put into the sorter were chosen based on the colour accuracy reading that the sensor saw. Based on tests with the colour sensor and the different skittle packs, the colours were picked from two different packs of skittles. Brown, yellow, and blue skittles were picked as they had the most consistent reading out of all of the colours.

Mechanical design decisions

Sorting disks

The 4-disk design was chosen for the reasons listed above but in addition to those reasons, these disk were easier to mount to the Tetrix prime kit, and easier to precisely fabricate so that skittles fit into it. The disks were accurate enough to have no risk of jamming, even at higher tested speeds. They also had sufficient rigidity during operation, so that if a skittle were to almost get jammed it would push it into the proper position as opposed to jamming or even breaking because of a skittle. The skittles fall through the system at a relatively consistent rate, meaning that while the disks are spinning, they do not end up spinning without a skittle in them.

Leading arm

The leading arm was chosen over the swinging arm from design #3 as it would be easier to manipulate and zero into position. The arm was made of a lighter material so that speed can be achieved leading to a shorter sorting time. The arm zeros out, meaning that the system can start in any configuration and still be able to go into proper working position without making it too much of a hassle for the user.

Three colours with a scrap bin

Out of the skittle packs that we had gotten we were only able to obtain three reliable colour options. These three options would give a consistent reading when a mode is taken from fifteen readings, which was already proving slow, adding any additional colours would cause the system to slow down, as no other colours could provide reliable readings under fifteen colour measurements. So, to keep speed acceptable and options open, the three colours, brown yellow and blue were chosen. A scrap bin was included for readings that are determined to be inaccurate to avoid a skittle ending up in the wrong jar.

Single colour sensor decision

Integration of a single sensor was already proving to be challenging, as the EV3 colour sensor simply lacked accuracy, and speed when reading the colour of a skittle. For instance, the colour green cannot be read as green instead it is read as brown or blue with no readable pattern, and utilizing the RGB mode provided no better accuracy. From these trials the input was limited to three colours, and from there it was deemed unnecessary to include a second colour sensor as it would provide no better accuracy than the first one did. This second reading stage will ultimately lead to a slower sorting process as a skittle has to be read twice. In the end a single colour sensor was used for sorting speed.

Problems

Putting in spacers between disks

Having acrylic disks rubbing against each other was not ideal for sorting skittles as the frictional forces made it difficult to accurately move skittles through the system, without jittering in the disk movement

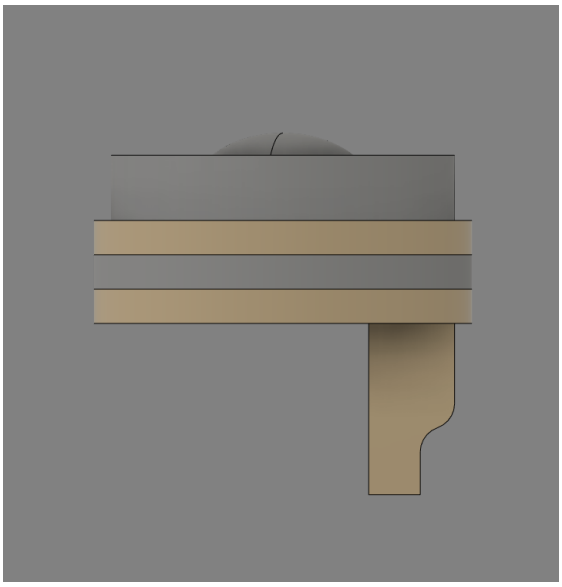


Figure 5: Disk system

Unfortunately, this initial design, as demonstrated in Figure 5, had a lot of friction and was not suited for control movement, not to mention it was causing the motor to heat up.

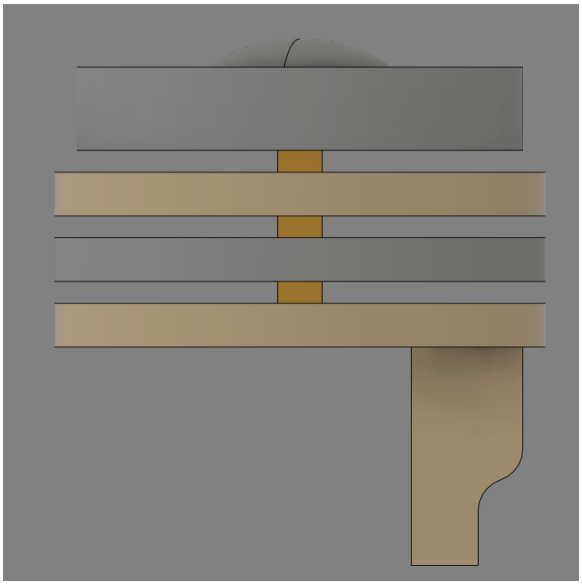


Figure 6 Disk System with Spacers

This design, as demonstrated in Figure 6 had spacers that reduced the friction to an immeasurable amount. This improved the accuracy of the motor movement and overall speed.

Colour sensor reading through acrylic

Initially the sensor was reading the colour of the skittle through the acrylic sheet. This was a cause for colour reading errors, as the light coming off the colour sensor would get refracted and reflected in a way that would give false readings to the EV3. This was discovered when a group member noticed a vast increase in the accuracy of colour readings the sensor provided after removing the acrylic in front of it.

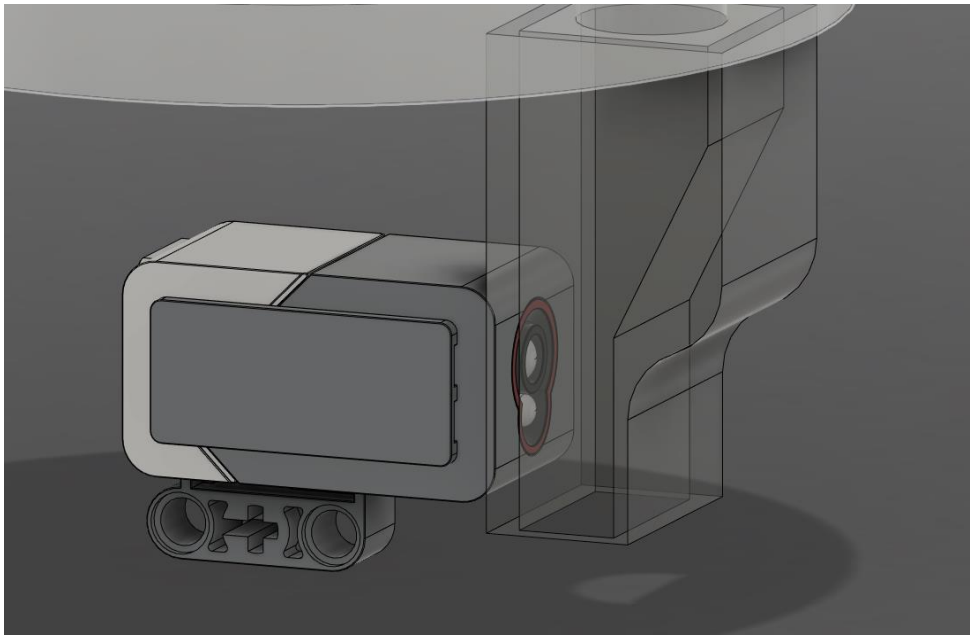


Figure 7: Colour sensor and acrylic sheet

The colour sensor was reading the colour through the acrylic sheet as demonstrated in Figure 7. This was a major error, as it could only work when the sensor was perfectly parallel with the sheet.

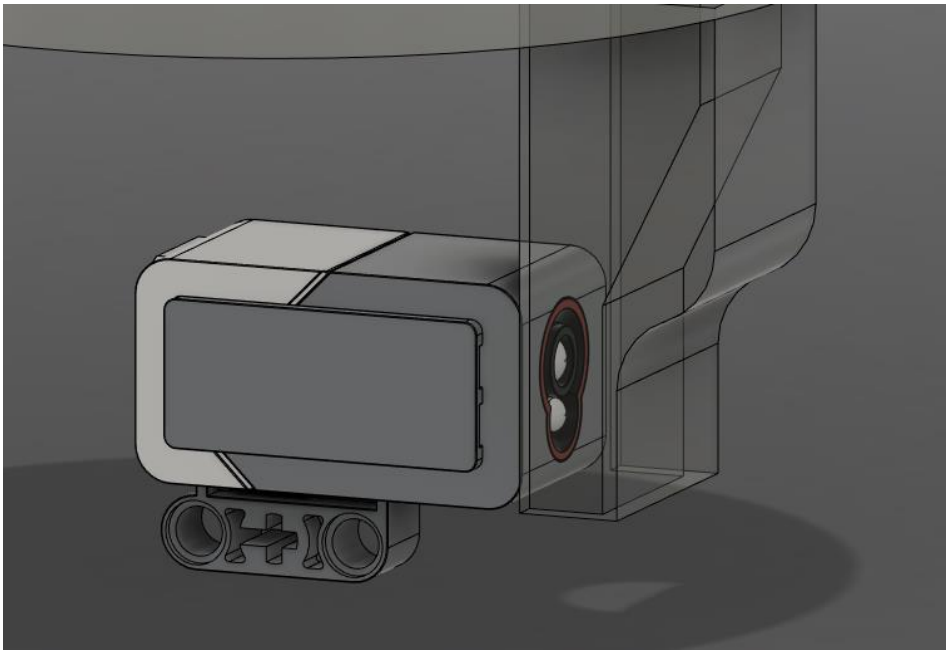


Figure 8: Colour sensor without acrylic sheet.

As demonstrated in Figure 8, the front acrylic sheet was removed. This gave the sensor direct access to the skittle to get a better reading. The inside of the chamber was also lined with white paper (not shown) to eliminate light reflection from the other sheets.

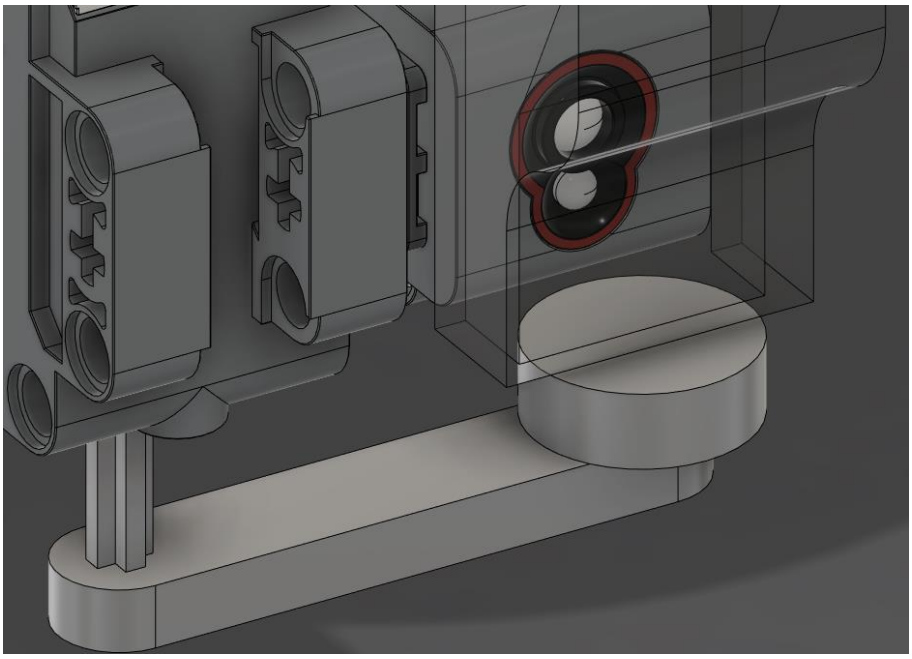


Figure 9: Shaker piece in lower position

Shaker piece needed to be changed in order to function properly

Initially, the shaker piece was sitting too low as demonstrated in Figure 9, and the sensor could not be moved downwards. This resulted in inconsistencies with shaking and colour readings as the skittle was sitting too low.

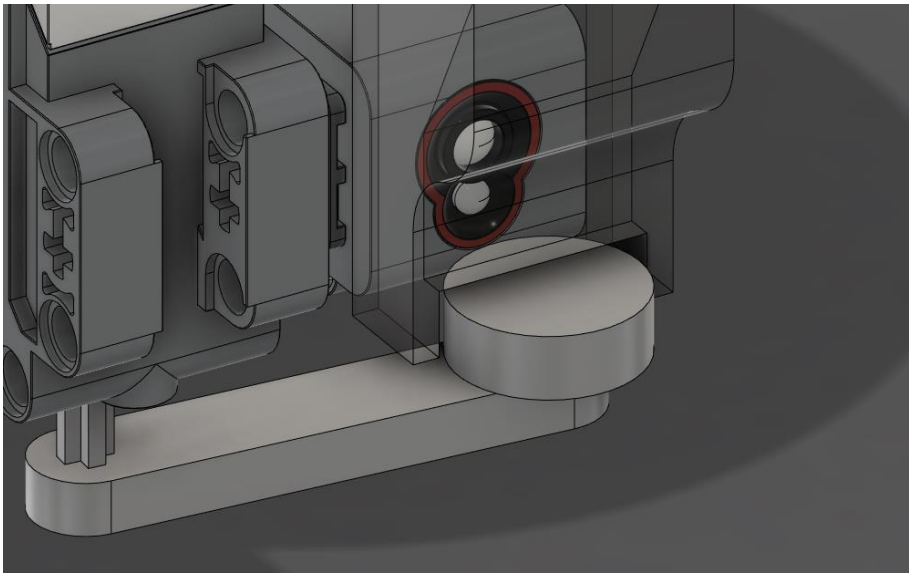
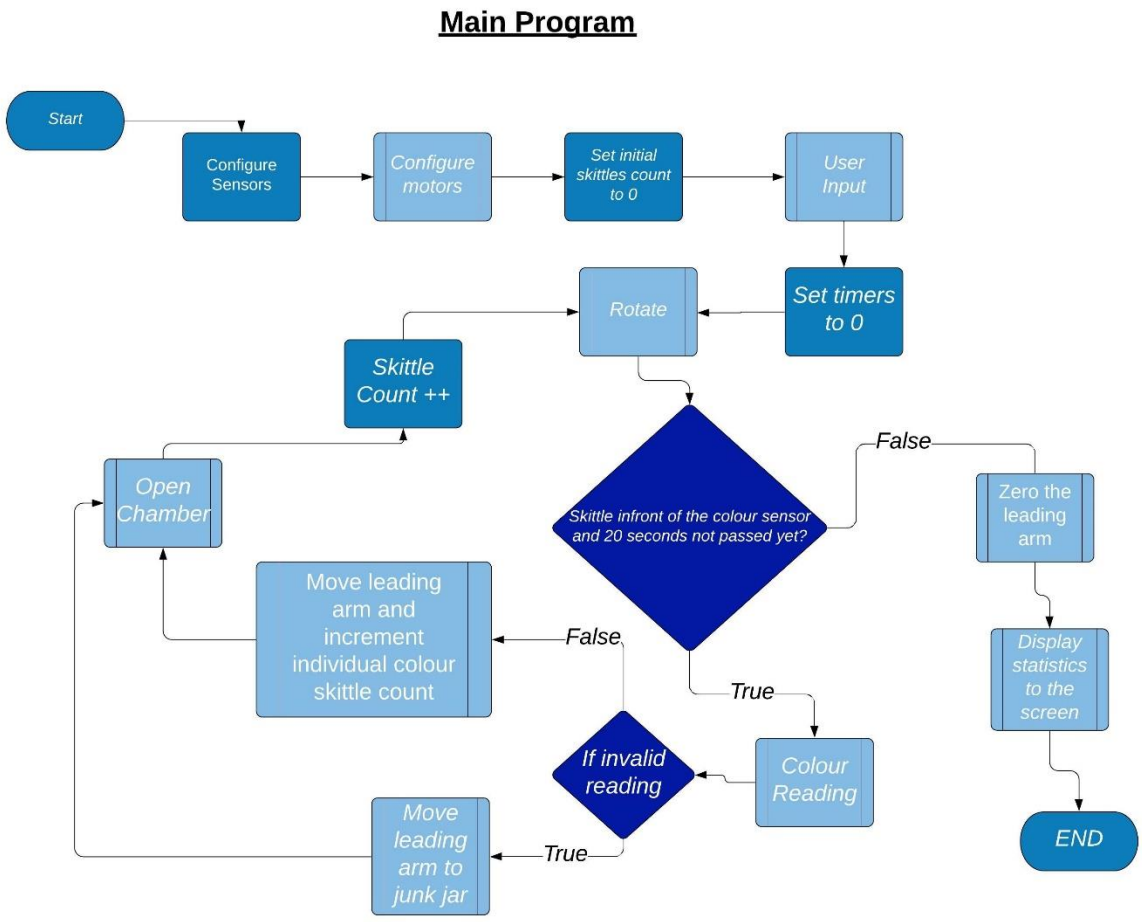


Figure 10: Raised shaker piece

The acrylic piece was cut so that the shaker piece is able to come into and out of the chamber while also being able to shake the skittle to read it accurately. This is shown in Figure 10. This decision was made so that the sensor could take readings of the skittle in a better position as it would not be able to if the shaker is too low. This change resulted in a vast improvement in the accuracy of colour readings and hence was kept.

Software design and Implementation

Overall Program Design



The overall code was broken into 4 different tasks, each with their own separate functions.

Task 1

The first task was configuring the motors and specifying the sorting process to the user's specific requests. We chose to create a function that would zero out the motors, as this would have to be run every time the robot started. We also created a function that would take input from the user, as we wanted the skittle sorting to be a modular process with different colour combinations allowed.

Task 2

The second task involved rotating the dispensing disk until the colour sensor read that a skittle had been placed in front of it. However, we wanted the robot to stop on its own if the colour sensor did not sense anything for 20 seconds. We accomplished this by creating a function that would rotate the dispensing disk and reset a timer that was constantly running every time a skittle was sensed.

Task 3

The third task involved using the colour sensor to compile data about the skittle in front of it. We chose to compartmentalize this task specifically, since it was very distinct from the other tasks. This was because the colour sensor would give unreliable readings that caused many problems for our group. We ended up fixing many of the consistency issues by taking an average of 15 readings before deciding whether a colour was confirmed. Then once the skittle has been confirmed, it is dropped into the chute using the leading arm.

Task 4

The final task involved exiting the loop once 20 seconds had passed without input. We decided to display the data gathered throughout the sorting process including the number of skittles sorted, the time to sort each skittle, as well as a count of each colour of skittle. Since these actions were only done once the sorter made sure that there were no skittles left in the chamber, we decided to put all of the data output in one function to be displayed at once.

No significant changes were made from previous task lists.

Functions

Table 2 - Function Descriptions

Function	Parameters	Return Type	How it Works (See Appendix B for flowcharts for each distinct function)	Who Wrote it
colourReading	none	int	Takes 15 colour sensor readings between being shaken. The mode of the readings is determined and if greater than 7 returns the colour's corresponding integer value. Otherwise returns 0	Jessie
userInput	int *jars	void	Prompts the user to press the buttons on the EV3 3 times (once for each colour). Depending on the button pushed, an array is filled with the corresponding motor encoder value that defines the location of the jar.	Brandon

rotate	none	bool	The motor which rotates the collection disks is turned on and left on until a skittle is detected by the colour sensor or 20s seconds has passed without detecting a skittle indicating that all skittles have been sorted	Omar
openChamber	tMotor motorA	void	Opens the chamber to allow the skittle to fall through to the leading arm. The piece that blocks the chamber is then moved back into its original position to block the exit of the chamber once again	Jessie
turn_arm	tMotor motorM, int index, int speed	void	Moves the leading arm the difference in motor encoder values of the current position of the arm and the next jar. The motor speed is positive or negative depending on the difference in these motor encoder values.	Brandon
display	int time, int skittleCount, int*coloursCount	void	Displays to the EV3 screen how many total skittles were sorted, how many of each colour were sorted, and the average time it took to sort one skittle.	Ali
zeroArm	none	void	The leading arm is instructed to rotate counterclockwise until it hits the touch sensor.	Omar
CalibrateMotors	none	void	Zeros the leading arm and moves the shaking arm so that it is blocking the opening of the chamber where the skittle is read by the colour sensor.	Ali
SkittleCounts	int currentColour, int* colours, int* coloursCount, int* jarsIndex	void	The function moves the leading arm to the appropriate jar depending	Jessie

			on the colour reading the colour sensor picked up. The count of the appropriate skittle is incremented.	

Data Storage

Four different parallel arrays are used to organize and store data in the program. The first stores the integer colour values for each skittle. The next stores the motor encoder value that marks the location of the jars corresponding to each colour of skittle. Another one stores the string of the colour name for each skittle. The final one stores the count of how many of each skittle are sorted throughout the program.

Trade-offs of Software Design Decisions

During the creation of the software, there were many decisions that were made to compensate for the inaccuracy of the colour sensor. We opted to create a looping code that would agitate the skittle and take multiple readings to confirm that the correct colour was being sensed. A trade off had to be made regarding the number of readings, as taking too many readings would result in slower sorting times. The optimal amount of readings turned out to be 15. It allowed for an accurate system that also met the 3 second per skittle sorting speed requirement. The testing process will be elaborated on further in the next section.

Testing Process:

Skittle Colour Selection Testing

As we had many consistency issues with the colour sensor, we had to determine which colours could be detected consistently by the sensor. Initially, we tested the 5 colours of original skittles, but many proved to be unusable.

We tested each skittle by creating a “driver” main program that would drop the skittles into the sorting chamber where the readings were supposed to take place. Then, the main program would call upon a function that would take 15 readings from the colour sensor. Each reading would be interrupted by a shake of a motorized arm to give the colour sensor a new orientation of the skittle to read. This was done to allow a more accurate reading in case the skittle landed in an orientation that caused the sensor to read the wrong colour. The readings were summed together, and the resulting number was divided by 15 to get an average reading. This reading was then displayed onto the Lego Brick. We did this 10 times for each of the original skittle colours to extrapolate the most consistent colour of the bunch.

Table 3 - Average of 15 colour readings for different skittle colours

	Red	**Orange**	**Yellow**	**Green**	**Brown**
Average of 15 Scans	3.4	2.5	2.4	3	2.6
	3.3	2.6	3.4	3.5	3.4
	4.3	3	3.4	3	5.4
	3	3.6	2.8	2	3.3
	3.6	2.7	3.4	3.5	5.4
	3.2	2.4	2.8	4.5	3.4

	3.5	2.9	2.6	4.5	2.6
	4.1	3.4	3.2	2.5	3.3
	3	2.9	3.2	4	3.6
	4.1	2.6	2.4	4	3.6
Total	3.55	2.86	2.96	3.45	3.68

As seen from Table 3 - Average of 15 colour readings for different skittle colours, red and orange skittles gave very similar readings to each other. The green skittles were very random, and the readings were often complete nonsense. Brown and yellow remained consistent and it was because of this that we decided to use these colours in our final version.

In order to implement a third colour, we decided to extend past the scope of the original skittle colours. To our surprise, the blue colour of tropical skittles gave a consistent reading of 2 every time we tested it. Below shows a table of each colour we tested and the readings that were possible.

Table 4 - Possible Colour Readings

	Green	**Red**	**Orange**	**Brown**	**Yellow**	**Blue**
Readings the Colour Sensor gave	Blue, Black, Yellow	Yellow Red	Yellow Red	Black Brown	Yellow	Blue

The combination that worked the most consistently across our testing were the brown, yellow, and blue skittles, and thus they were chosen to be the skittles that the robot sorted during demo day.

User Input Testing

Our robot was required to have the functionality of taking in a user input that decided where to place individual skittle colours. We decided to let the user pick whether they wanted multiple colours in one jar or each colour in their own distinct jar. To test this code, we tried each possible colour combination with each jar configuration. This was done to make sure that there were no errors in the implementation of the motor encoder that powered the leading arm.

After testing the user input thoroughly, we noticed it was behaving as expected, but in the reverse order. Skittles coded to be placed in the left jar would end up in the right jar, and vice versa. This was less of an error with the software, and more of an issue with how the jars were positioned in relation to the Lego Brick that allowed for user input. To fix this, we simply reversed the locations in which the left and right jars were coded. We also made it abundantly clear which skittles would be placed in which jars by numbering each jar from 1 to 3 as well as the buttons on the Lego Brick. That way, both the software and the physical system would give the user an intuitive experience.

Optimal Number of Readings Testing

The final robot assembly was tested using integration testing. After all the components were individually unit tested to check for any abnormalities, we ran the sorting code together in a main source file that became our final robot code. When It ended up successful, we decided to optimize the robot further by finding out the optimal number of colour readings to take for each skittle. This was tested by running

the robot with a set of 30 skittles and counting the number of skittles that were misplaced by the robot arm.

Table 5 - Time to sort skittles and accuracy for different numbers of colour readings taken

Trial	Number of readings per skittle	Time to sort 1 skittle	Number of incorrectly sorted skittles
1	7	0.9	5
2	7	0.7	1
3	10	1.0	0
4	10	1.0	2
5	15	1.1	0
6	15	1.3	0

From Table 1, we were able to conclude that the optimal number of readings per skittle was 15. This was because the accuracy was highest.

Resolution of Problems:

A significant problem that was encountered was the accuracy of the colour sensor. Colours such as red, orange, purple and green could not even be picked up by the colour sensor, instead giving wide ranges of readings that were not even correct. As a result, blue, yellow and brown skittles were chosen to be sorted because they were the only colours for which reliable and consistent readings could be picked up.

Another obstacle was the integration of the RobotC code with certain components of the mechanical system. The movement of the arm responsible for shaking the skittle in front of the colour sensor was challenging to control. If the starting position of the piece was not lined up perfectly below the colour sensor, all the readings could be thrown off for the remainder of the sorting process. This problem was amended by using the following line of code to configure the motor position.

```
while(SensorValue[S1] != (int)colourWhite){}
```

It was discovered that when the arm was sitting directly below the colour sensor, it would read white, whereas when not, blue would be detected. This allows us to align the arm properly at the beginning of the program by instructing the motor to rotate until the colour sensor detects white. This vastly improved the robot's ability to sort skittles without interruption from “phantom” readings.

Verification

Before the robot was built and programmed, there were various requirements and constraints which were listed in the preliminary design report which were targeted to be met by the time of the robot demo. This was done so that during the design stages, and the build phase of the robot, our group knew exactly what kind of constraints we were dealing with, and what standards we required on the robot’s ability to complete the tasks. Now that the robot has been completed and tested, it is important to look back and “verify” how the overall demo results compared with the requirements which were emplaced prior.

Criteria:

Requirement #1: Candy is Not Damaged

This requirement was set initially since we believed it would be of the utmost importance that if the robot had the primary task of sorting skittles, that it be able to do so without causing any damage to them. One of the things we did, was ensure that all parts of the robot through which the skittles would pass through, were sufficiently large enough so that they could pass with ease. Secondly, we ensured that there was never more than the maximum capacity of skittles in the collection chamber (which through testing was found to be 45), so that the programming of the robot could effectively prevent more than a single skittle passing into view of the colour sensor. Following through with all these measures resulted in a successful demo in which all 33 skittles were sorted without damage.

Requirement #2: Robot Must Stop by Itself

The second primary requirement we had enforced on our skittles sorter, was that it be able to autonomously detect when the entire skittles batch has been processed and know to stop running the program. We decided that this was an important requirement to incorporate into the final design since the whole idea behind having a robot to sort skittles was to try and minimize any user interference from the process. If the robot could sort the skittles into each jar and stop by itself, it would leave for the user the simple task of having to retrieve the sorted skittles from the jars. To do this, we incorporated into the software of the robot a loop which would keep the program running for as long as the colour sensor was not reading white for more than 20 seconds. As soon as the skittles were all sorted, the timer would wait 20 seconds to allow time for any potentially jammed skittle in the top chamber to be freed, before ending the program. As such, the requirement of stopping by itself was met.

Constraints:

Constraints #1: Must be fewer than 10% wrong Skittles in Total

We decided before going into the demo that the skittles sorter would be deemed “effective” if it could sort all the given skittles with a minimum 90% accuracy. This marginal error was incorporated to account for the colour sensor’s tendency for inaccuracy. On demo day, the robot was filled with 33 skittles, all but 1 of which were sorted into the correct container; for an overall accuracy of 97%. To add, the one skittle that ended in the wrong container only got there after bouncing out of the correct one. The way that we achieved this high degree of accuracy whilst using a rather inaccurate colour sensor, was by testing all the skittles colours on the sensor and determining which were best read, and which the colour sensor had difficulty reading and differentiating. After many test cases and trials, we found that the colours it sensed best were blue, yellow, and brown. So, for the demo, we made sure to only use those skittles.

Constraints #2: Must not sort fewer than one Skittle per 3 seconds

This constraint was emplaced so that we would keep speed and efficiency as a high priority during the designing of the robot. We decided upon 3 seconds spent per skittle as a reasonable amount of time for the sorter to filter out the skittle from the main chamber, get it to the colour sensor to be read, and have it delivered to the correct container. During the test phases of the robot, we tried various combinations of motor speeds and colour sensor reading methods to try and find a way which

maximized the accuracy, and efficiency of the robot. The combination we found that worked best, and the one that we ended up using on demo day was having the motor rotate the layers at a speed of 30, while having the colour sensor taking 15 readings per skittle. On demo day our robot was able to sort 33 skittles in about 37 seconds, meaning 1.1 seconds spent per skittle, which actually exceeded expectations.

Project Plan

TEAM ABU 468

Period Highlight: 1 — Plan Duration — Actual Start — % Complete — Actual (beyond plan) — % Complete (beyond plan)

ACTIVITY	PERSONS	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE
Finish GANTT Chart	Brandon	1	1	1	2	100%
3D Model for Report	Omar	1	1	1	0	100%
Scope for Report	Jessica Ali Jess	1	2	1	0	100%
Introduction for Report	Brandon	1	2	2	0	100%
Conclusion for Report	Ali Omar	2	2	2	0	100%
Finish 3D Model	Omar	2	3	0	0	100%
Mechanical Overview Slides	Omar Brandon	3	2	0	0	100%
Procedure Overview Slides	Ali	3	2	0	0	100%
Software Overview Slides	Jessica	3	2	0	0	100%
Intro Conclusion Slides	Brandon	3	2	0	0	100%
Mechanical Overview Slides	Omar	3	2	0	0	100%
Mechanical Prototype Build	Omar Jess Brandon Ali Omar Brandon	5	2	0	0	100%
Final Design Meeting	Jess Ali	6	1	0	0	100%
CNC Parts	Omar Brandon	6	3	0	0	100%
Skittle Rotation Software Code	Brandon Jess	6	3	0	0	100%
GUI Software Code	Ali Omar	6	3	0	0	100%
Skittle Rotation Software Code	Brandon Jess	6	3	0	0	100%
Assembly of Printed Parts	Omar Brandon	7	3	0	0	100%
Motor Attachment	Ali Jess	9	3	0	0	100%
Final Assembly	Omar Ali Omar Jess	10	4	0	0	100%
Bug Testing Software	Brandon Ali	10	4	0	0	100%
Test Trials For Sorting Skittles	Omar Brandon	10	10	0	0	100%
Test Trials For Dispensing Skittles	Jess Ali Omar Jess	10	10	0	0	100%
Final Troubleshooting for Demo	Brandon Ali	20	1	0	0	100%
Software Component of Report	Jess Ali	21	8	0	0	100%
Physical Component of Report	Brandon Omar	21	8	0	0	100%

PERIODS 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36

This project plan was taken directly from the Preliminary Design Document [2].

Task Separation

As seen in the original project plan, the mechanical, software, and presentation tasks were all separated nicely between the four group members. Tasks that could be completed in tandem were divided amongst the group members to maximize efficiency. This included the slideshow presentation, where each group member was responsible for their own set of slides. Similarly, when it came to actually creating the robot, each group member was tasked with something they could complete either on their own or with another partner. Smaller tasks like CNC cutting the parts or the skittle rotation

software code were split into groups of two, whereas more general tasks like final assembly required all four members of the group working together.

For the tasks that required two people, a group member with more skill in that particular area was often assigned to the task so that there would be no time wasted in going back and fixing faulty hardware or code. By optimizing to the strengths of the group members, the tasks would theoretically be completed quicker and with greater quality.

Revisions

There were no revisions to the project plan made after the fact.

Plan Compared to Actual Schedule

The actual schedule was much more hectic as compared with the GANTT chart. Group members would often fall behind schedule and others would have to assist them to finish on time. This was especially true with tasks that had hard deadlines like the Preliminary Design Document and Slideshow Presentation.

In addition, the Mechanical Prototype Build task ended up getting scrapped, as our prototype build became our final project since we had no complaints with how the CNC parts turned out.

A task that ended up taking much longer than expected was the final assembly of the robot. This was because of the problems we were having with the colour sensor detecting incorrect values. The final assembly of the CNC acrylic had to be taken apart to remove the clear sheet that was reflecting colours and causing the inaccurate readings. This extended the final assembly portion by 2 days, as the colour reading code had to be redone to account for new values coming from the colour sensor.
In addition, a new shaking arm had to be put in place in order to fit the shape of the modified acrylic piece as demonstrated in Figure 10.

Other tasks took shorter than expected to complete. An example of this was the motor attachment, which was scheduled to take 3 days to complete. Since we ended up mounting all the motors within the same time frame without any complications, it only took 1 day. This allowed us to spend more time bug testing our code before demo day to make sure that everything was working properly.
Overall, the actual schedule was quite different from the GANTT chart, but the plan allowed us to keep track of deadlines and pick up the pace when we were behind schedule.

Conclusion

In conclusion, the goal of our robot was to efficiently sort skittles based on colour, into jars of a user's choice to potentially prevent someone with a specific food colour dye allergy from accidentally inducing an allergic reaction. This was done by using various sensors included in the Lego EV3 kit, and the construction parts from the Tetrix kit. Our robot used 3 motors for the final design, one to rotate the top layers, one to shake the skittle in the colour sensor position, and one to rotate the leading arm. It also used 4 sensor readings including a colour sensor, a touch sensor, the motor encoder, and the EV3 buttons.

The robot was programmed by breaking down the overall main task into various smaller functions. The main function could be broken down into the following tasks: First, obtain user input data on choice of

jar designations based on colour; rotate the top layers for as long as there is skittles; take colour sensor readings of the skittle once it is in view; move the arm into position depending on the colour determined earlier; finally, open the chamber to release the skittle into the leading arm and into the desired jar. This is a basic overview of the way that the whole mechatronic system works together to complete the main desired task of sorting a batch of skittles based on colour, which our robot was able to successfully complete during the demo, while also following all design requirements and constraints.

Recommendations

Although our skittles sorting robot was able to successfully sort the skittles into the desired containers, there are various things which could have been improved upon which would have made the robot overall more effective and useable in a real-life test scenario. These design "recommendations" can be categorized into either of two categories, software improvements, and mechanical design improvements.

One of the problems which our robot encountered was skittles jumping out of the correct container into the wrong one after leaving the leading arm. What could have been done to prevent this in the future, would be to design the containers such that once the skittles enter, they get funneled into a more organized assortment, this would both prevent the problem, while also making it easier for the user to retrieve the skittles once they have been assorted, as opposed to having to cup all the skittles out with their hands.

Another mechanical design improvement would be to mount the colour sensor in a way that would allow for more accurate readings, possibly even allowing for more colours to be accepted for sorting. Currently, the colour sensor was programmed to only sort blue, yellow, and brown skittles. However, if we were able to potentially mount the reading chamber in a way that would allow each individual skittle to be perfectly aligned with the colour sensor, without the need to constantly shake it, the sensor could have given more accurate readings.

On the other hand, there is also various software improvements which could be done to improve the effectiveness of the robot. For one, we could have used a different colour sensor mode which may have been more effective at accurately determining the exact colour of each skittle, this would have also allowed us to sort an even greater variety of skittle colours. Currently, the colour sensor was being used in the "colour" mode, where it detected 7 different colour readings. What this meant, was that there was often some degree of ambiguity whenever there was a colour that could not be categorized exactly into one of these 7 colours. To circumvent this, one thing we could have considered is the EV3 RGB colour sensor mode. This would have allowed us to obtain the exact RGB light intensity values each ranging from 0 to 255, theoretically allowing the colour sensor to detect colours of a much greater spectrum.

References

[1] M. Mittler, "Red and Yellow Dyes May Be Causing Your Stomach Aches," 4 November 2019. [Online]. Available: https://www.verywellhealth.com/red-and-yellow-may-be-the-cause-3956894. [Accessed 4 November 2019].

[2] J. Ramseyer, T. Brandon, A. A. Omar and A. Ali, "Preliminary Design Document," Waterloo, 2019.

[3] snackhistory.com, "Skittles," 2019. [Online]. Available: https://www.snackhistory.com/skittles. [Accessed 4 November 2019].

Appendix A:

```
//Omar Abuabah, Jessie Ramseyer, Brandon Tu, Ali Arshad
//Group 468

int colourReading()
{
    int readingsCount[8] = {0,0,0,0,0,0,0,0};
    int readings[15] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

    for (int reading =0; reading <15; reading++)
    {
        motor[motorA] = 30;
        wait1Msec(20);
        motor[motorA] = 0;
        wait1Msec(20);
        motor[motorA] = -30;
        wait1Msec(10);
        motor[motorA] = 0;
        wait1Msec(20);
        readings[reading] = SensorValue[S1];
    }
    for (int reading =0; reading < 15; reading++)
        readingsCount[readings[reading]] ++;

    for (int size =0; size <8; size++)
    {
        if (readingsCount[size] > 6)
        {
            return size;
        }
    }
    return 0;
}

void userInput(int *jars)
{
    string colourNames[3] = {"yellow", "blue", "brown"};//DO NOT
REMOVE
    for(int count =0; count <3; count++)
    {
        displayString(3,"What jar would you like %s skittles in?",
colourNames[count]);

        while(!getButtonPress(buttonAny))
        {}

        if(getButtonPress(buttonLeft))//if statement for each colour
        {
            while(getButtonPress(buttonLeft))
            {}
```

```
            jars[count] = -90; //populate array with motor encoder
values of each jar
        }
        else if (getButtonPress(buttonUp))
        {
            while(getButtonPress(buttonUp))
            {}
            jars[count] = -60;
        }

        else if(getButtonPress(buttonRight))
        {
            while(getButtonPress(buttonRight))
            {}
            jars[count] = -30;
        }
    }
    eraseDisplay();
}

bool rotate()
{
    time1[T1] =0;

    motor[motorB] = 30;

    while(SensorValue[S1] == 6  && time1[T1] < 20000)//may see
something when nothing is there and give a false positive
    {}
    motor[motorB] = 0;
    if(time1[T1] > 19999)
        return false;
    return true;
}

void openChamber(tMotor motorA)
{
    motor[motorA] = -50; //make sure that the correct motor is being
called and the power moves it in the right direction
    while(nMotorEncoder[motorA] >-30)
    {}
    motor[motorA] =0;
    wait1Msec(50); //test for time that works

    motor[motorA] = 50; //make sure that the correct motor is being
called and the power moves it in the right direction
    while(nMotorEncoder[motorA] <0)
    {}
    motor[motorA] =0;
}

void display(int time, int skittleCount, int*coloursCount)
```

```
{
    string colourNames[3] = {"yellow", "blue", "brown"};//DO NOT
REMOVE

    displayString(3, "Total # of skittles sorted: %d", skittleCount);
    int line = 4;

    for (int index = 0; index < 3; index++)
    {
        displayString(line+index, "%s skittle sorted: %d",
colourNames[index], coloursCount[index]);
    }

    displayString(7, "Skittles sorted / second: %f ",
(float)skittleCount/((time-20000)/1000));
    displayString(8, "Seconds to sort 1: %f ", 1/(
(float)skittleCount/((time-20000)/1000)) ) ;
    displayString(9, "Time: %f",time);
}

void turn_arm(tMotor motorM, int index, int speed) {
    int i_angle = nMotorEncoder[motorM];
    if(index - i_angle > 0)
    {
        motor[motorM] = speed;
        while(nMotorEncoder[motorM] < index)
        {}
        motor[motorM] = 0;
    }
    else
    {
        motor[motorM] = -speed;
        while(nMotorEncoder[motorM] >= index)
        {}
        motor[motorM] = 0;
    }
}

void zeroArm()
{
    motor[motorD] =    40;
    while(SensorValue[S2] == 0){}
    motor[motorD] = 0;
    nMotorEncoder[motorD] = 0;
}

void CalibrateMotors(){
    zeroArm();
    if(SensorValue[S1] != (int)colourWhite)
    {
        motor[motorA] = 30;
        wait1Msec(400);
```

```
        while(SensorValue[S1] != (int)colourWhite){}
        motor[motorA] = motor[motorD] = 0;
    }
    nMotorEncoder[motorA] = 0;
}

void skittleCounts(int currentColour, int* colours, int* coloursCount,
int* jarsIndex)
{
    if (currentColour == 0)
        {
            turn_arm(motorD, 0, 30);
        }
        else
        {
            for (int index = 0; index <3; index++)
            {
                if(currentColour == 1 && colours[index] == 7)
                {
                    turn_arm(motorD,jarsIndex[index], 40);
                    coloursCount[2]++;
                }

                else if (colours[index] == currentColour)
                {
                   turn_arm(motorD, jarsIndex[index], 40);//move
     leading arm
                    if(colours[index] == 2)
                    {
                        coloursCount[1]++;
                    }
                    else if(colours[index] == 4)
                    {
                        coloursCount[0]++;
                    }
                    else if(colours[index] == 7)
                    {
                        coloursCount[2]++;
                    }
                }
            }
        }
}

task main()
{
    SensorType[S1] = sensorEV3_Colour;
    wait1Msec(50);
    SensorMode[S1] = modeEV3Colour_Colour;
    wait1Msec(50);
    SensorType[S2] = sensorEV3_Touch;
    CalibrateMotors();
```

```
    int currentColour =0, skittleCount = 0;
    int colours[3] = {4,2,7};
    int jarsIndex[3] ={0,0,0};
    string colourNames[3] = {"yellow", "blue", "brown"};
    int coloursCount[3] = {0,0,0};

    userInput(jarsIndex);

    time1[T2] = 0; //timing the entire sorting process

    while(rotate())
    {
        currentColour = colourReading();
      skittleCounts(currentColour, colours, coloursCount, jarsIndex);
        wait1Msec(200);
        openChamber(motorA);//make sure its the right motor
        skittleCount ++;
    }

    zeroArm();

    display(time1[T2], skittleCount, coloursCount);
    wait1Msec(20000);
}
```

Appendix B:

Figure 11: Flowchart for colourReading function

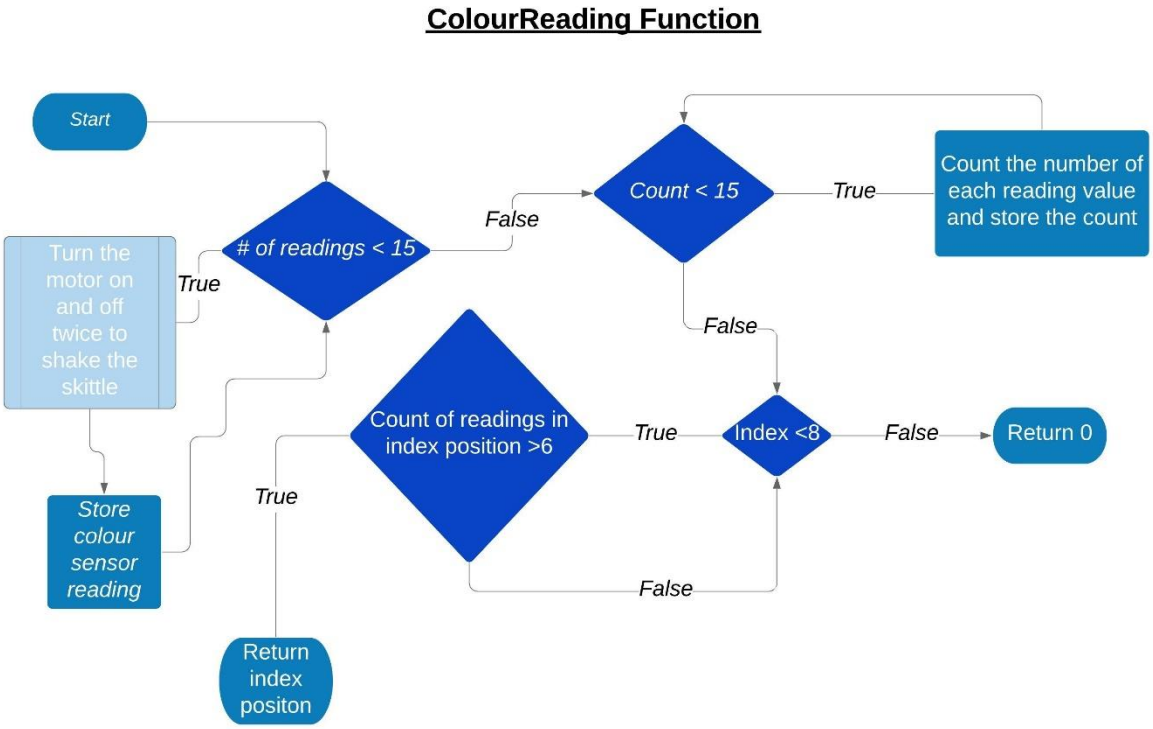


Figure 12: Flowchart for userInput function

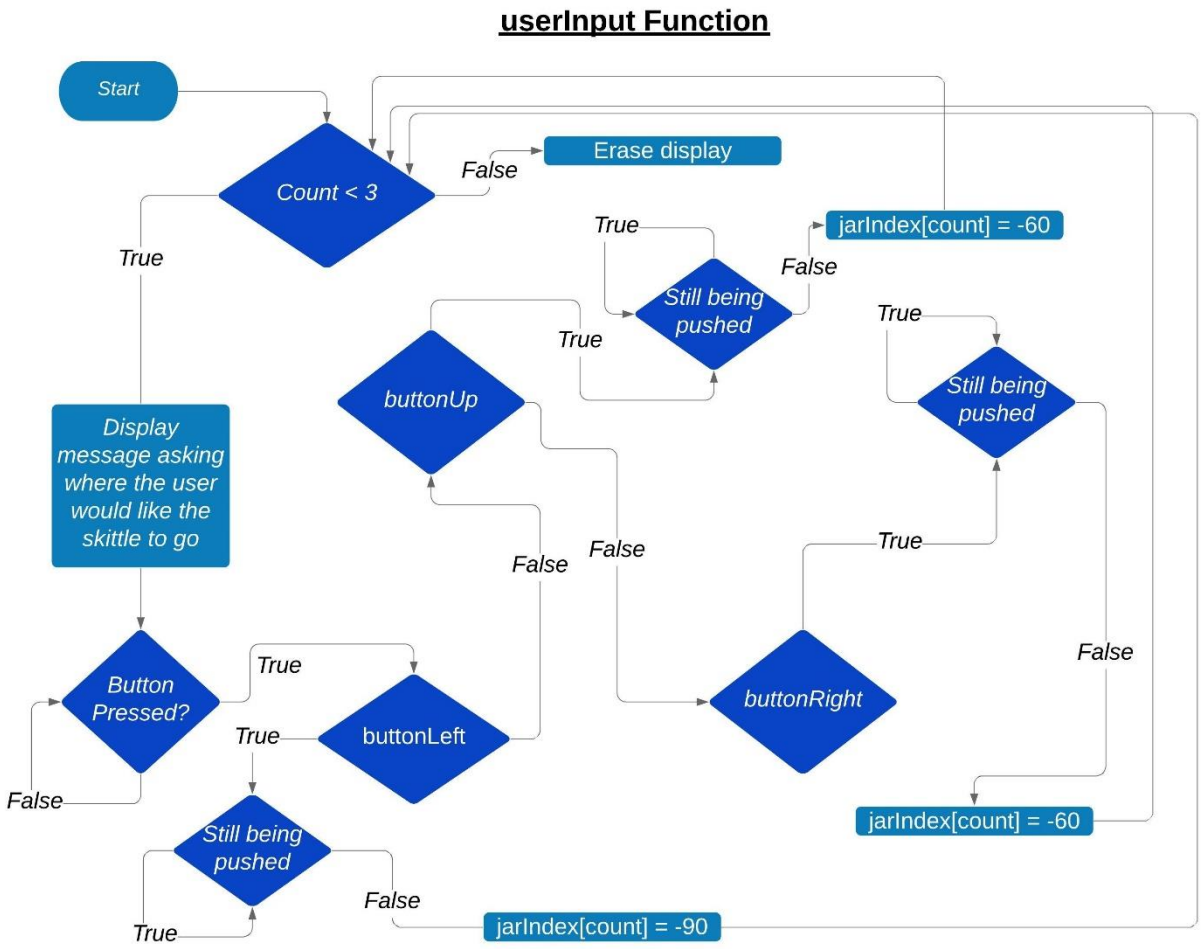


Figure 13: Flowchart for rotate function

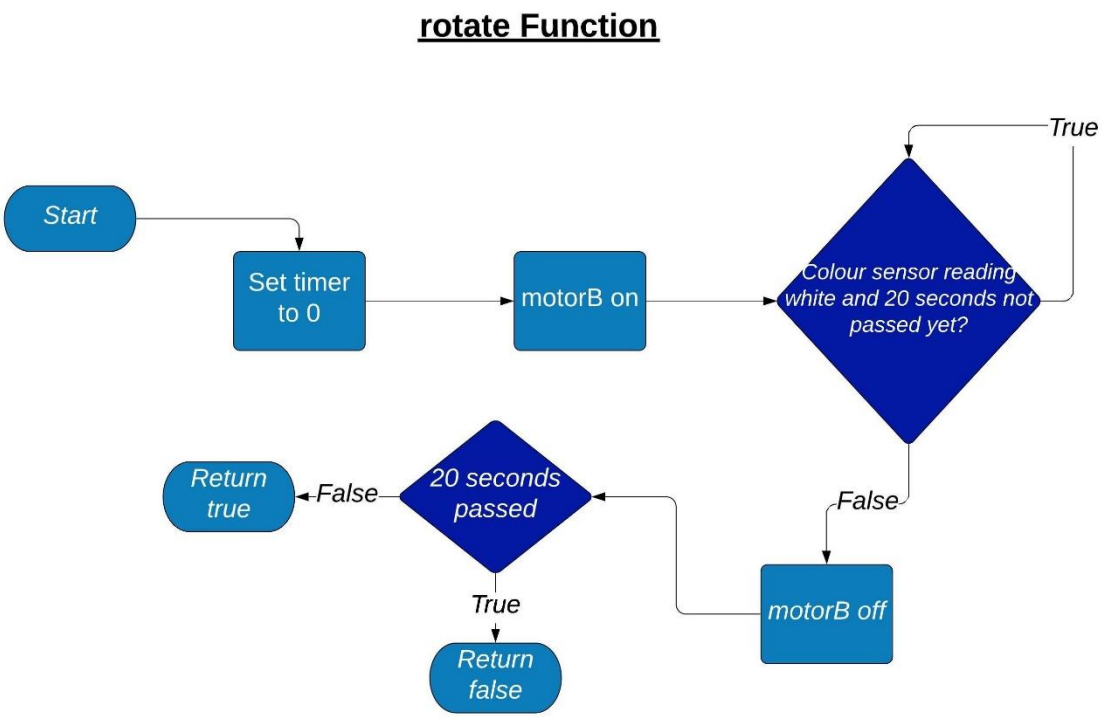


Figure 14: Flowchart for openChamber function

openChamber Function

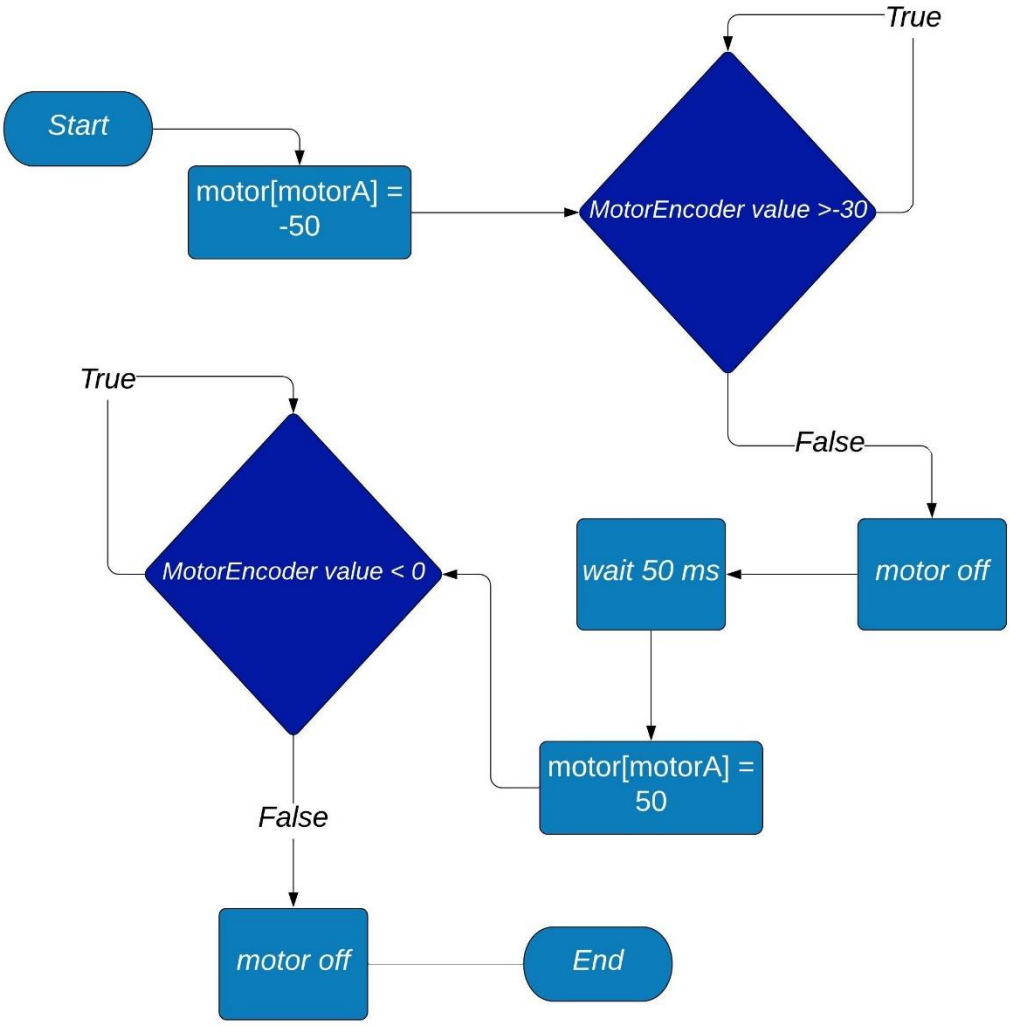


Figure 15: Flowchart for display function

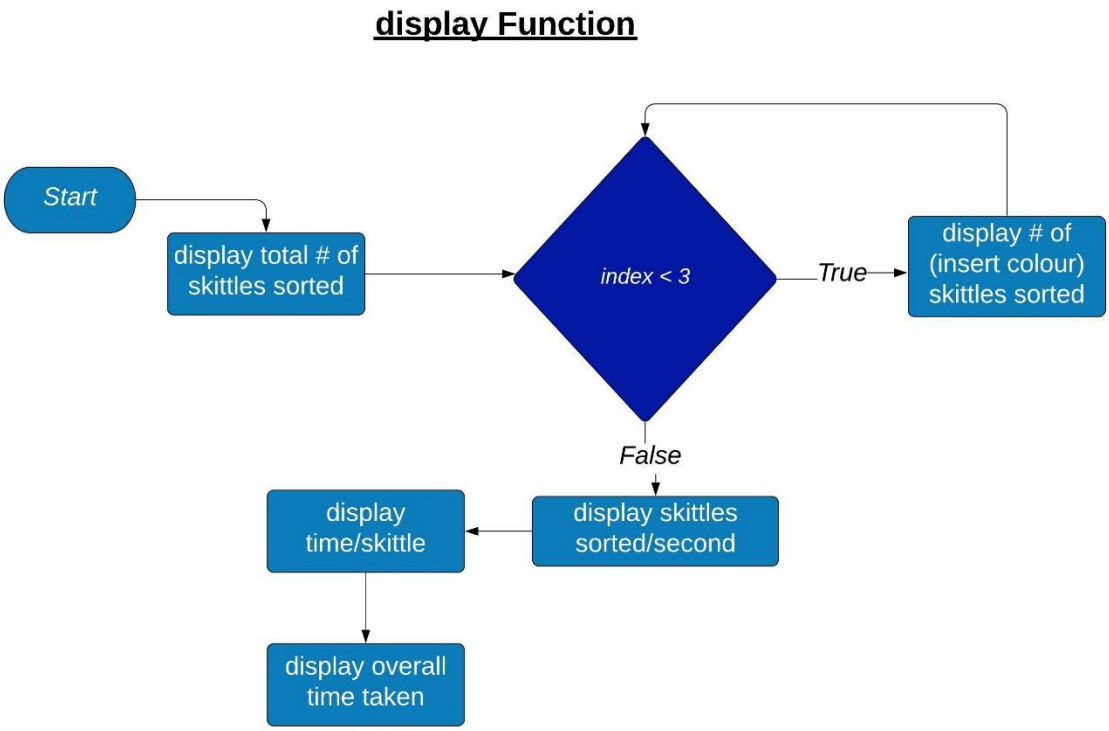


Figure 16: Flowchart for turn_arm function

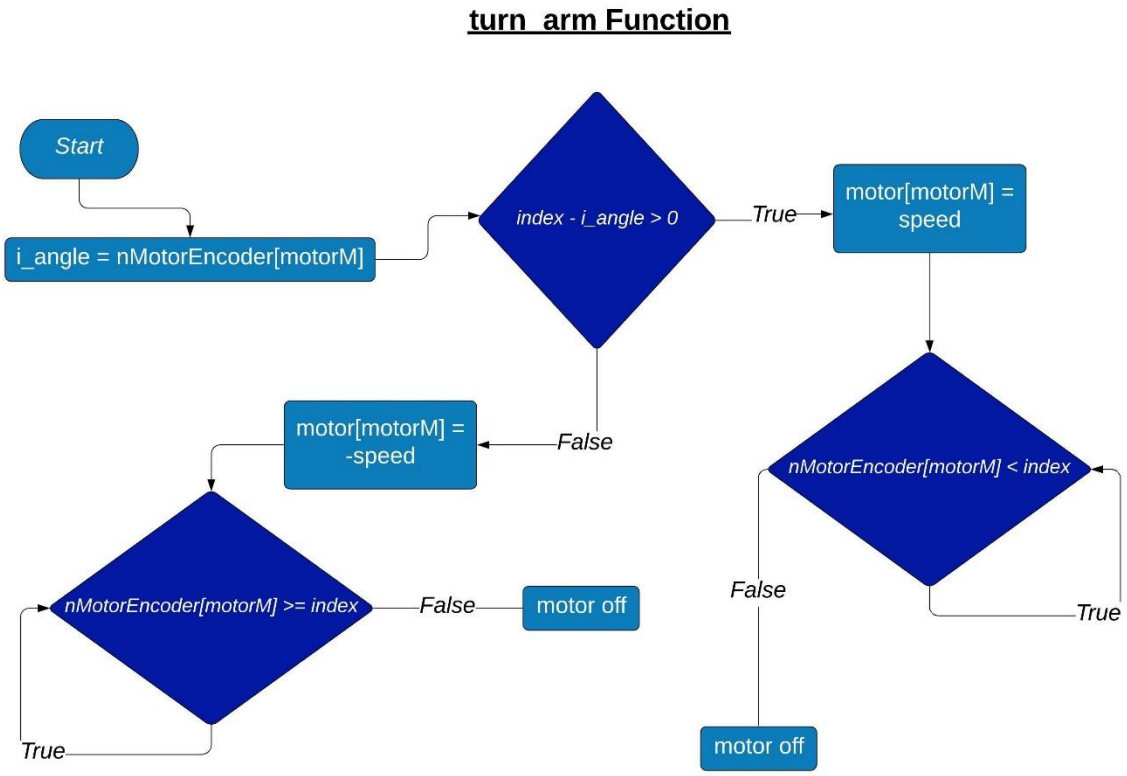


Figure 17: Flowchart for zeroArm function

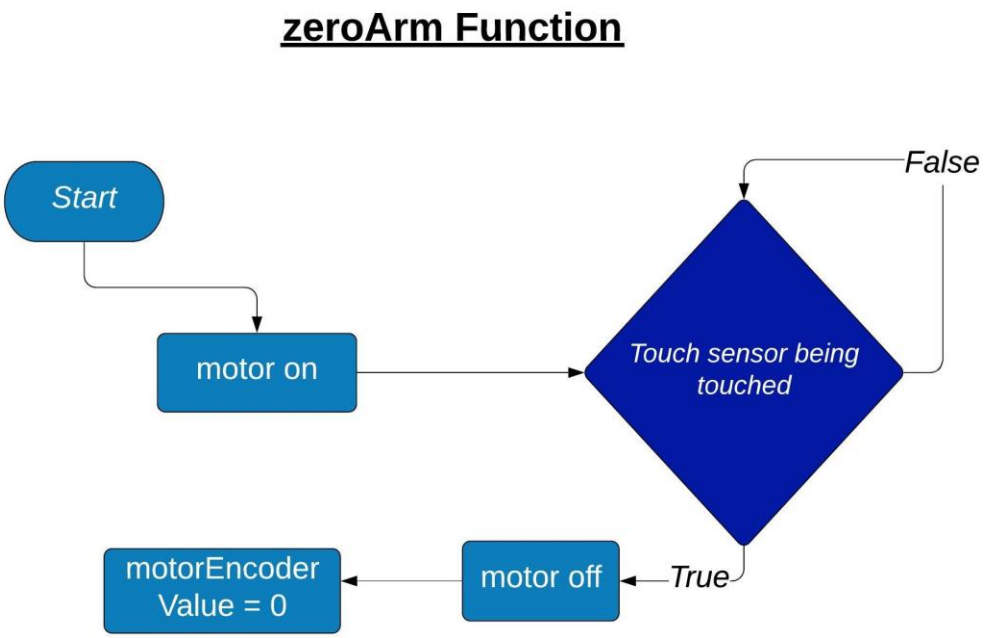


Figure 18: Flowchart for CalibrateMotors function

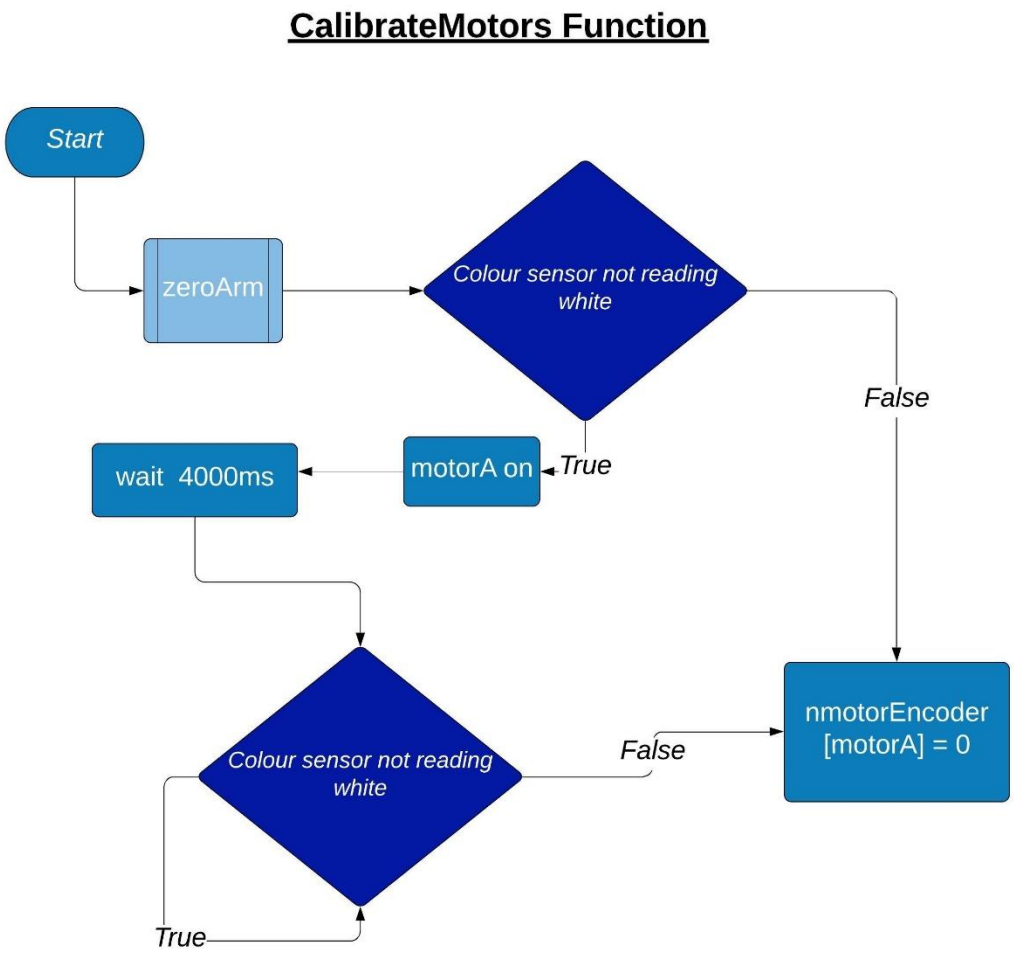


Figure 19: Flowchart for skittleCounts function