Piano Robot Project Report

UNIVERSITY OF
Waterloo

Department of Mechanical and Mechatronics Engineering

A Report Prepared For:
The University of Waterloo

Prepared By:
Marco Xu (21065187), Seyon Menakan (21083297), Omer Farooq (21090356), Nitai Michalski (21090305)
Group 23
ME 101
July 30, 2024

Table of Contents

List of Figures

Summary

This report covers the development of an autonomous robot designed to play the piano using an EV3 robot controller and a color sensor to read sheet music. The primary goal of this project was to create a robot that could read sheet music and play two octaves of a piano while staying under a $100 budget (excluding the EV3 robot cost). This document details the project's background, problem definition, technical progress, mechanical and software design, verification, and project management. The motivation behind the robot was to showcase the EV3 robot's capabilities while creating a challenging project. The robot was designed to read sheet music using color sensors, save played songs, and accurately press piano keys. The engineering specifications included cost constraints, safety, note-playing delay, sheet music reading accuracy, and ease of setup. The final design featured a compact mechanism with color sensors, a conveyor belt for feeding sheet music, and robotic fingers for pressing the keys. During the conceptual design phase, the team brainstormed various solutions for the subfunctions of reading sheet music, saving songs, and pressing piano keys. The final design combined the most feasible solutions, such as a fixed color sensor with rotating sheet music and robotic fingers controlled by motors. Detailed sketches and CAD models ensured accurate and efficient mechanical design. Verification tests confirmed that the robot met all primary functional requirements, including playing notes with minimal delay, accurately reading sheet music, and effectively handling user input. The software design included functions for configuring sensors, storing and reading note values, and controlling the robotic fingers. The project management section highlighted task delegation and adherence to the project plan. Recommendations for future improvements included ensuring identical gears for the right finger, making the design more symmetrical and visually appealing, using a larger keyboard, upgrading motors for better performance, and enhancing the user interface for a more interactive experience. These enhancements would further improve the robot's functionality, reliability, and user experience. In conclusion, the project successfully achieved its objectives, and the report outlines the next steps for continued development.

1.0 Needs Analysis

1.1 Background Description

The project aimed to create an autonomous piano playing system for the EV3 robot, capable of playing notes on a keyboard based on sheet music. The robot would be able to save and play songs, appealing to anyone who understands music, and presenting a challenging programming and design challenge.

1.2 Needs Statement

A need exists for a robot to play all white keys of 2 octaves of a keyboard/piano while costing under $100 without including the cost of EV3 robot and any other given components. The primary functional requirement is to play 2 full octaves of the keyboard from the low “C” note to the high “C” note. The primary constraint requirement is to build the piano robot with the given components and any extra components giving a total cost of under $100. The budgeting and cost tracking will be highlighted in a later section.

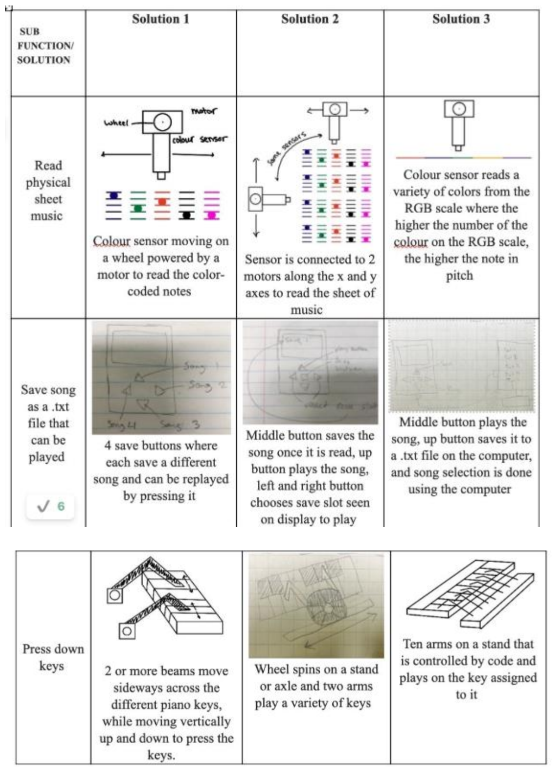
1.3 Engineering Specification

Each specification is carefully designed to ensure the project meets its functional, safety, and performance requirements [1]. The Engineering Design Specification for the project encompasses ten critical characteristics. Firstly, the cost is constrained to be less than $100.00 CAD, which will be verified through analysis by totaling the receipts of any additional expenses. Secondly, safety is a key constraint and is paramount; it is verified through demonstrations to ensure there are no pointy edges, pinch points, or potential fire hazards. Thirdly, a functional requirement is that the system must play notes with minimal delay, specifically less than one second. This characteristic will be tested using a stopwatch to measure the delay between key presses, calculating the difference between the highest and lowest delays. Furthermore, the ability to play modified sheet music will be examined by having an experienced pianist cross-check if the notes are played correctly. Another functional requirement is for the sheet music reader to read the color sensor combinations with an 80% accuracy at least which can be verified by outputting the notes onto a .txt file or onto the console and having someone cross-check with the predetermined color sensor combinations. A functional requirement that must be considered is the range of the notes being played, which is from Low C to High C. This is exactly two octaves and only the white keys will be played. This can be verified from making the robot play the C major scale on two octaves, which will therefore hit every key from Low C to

High C. A non-functional requirement for the system is that it should be easy to set up. The robot should be assembled every time without damage to the components, and without any of the components falling out of place. A functional requirement to consider is the maximum amount of notes the robot can play, which is limited to 70 notes. This is because as the robot plays notes using over a longer period, the robot becomes more and more inaccurate with the angles needed to hit each individual key. This can be verified by making the robot play a .txt file with over 70 notes and see if the robot completely stops playing at 70 notes. Moreover, user input is also a functional requirement to increase the robot's enjoyability. The design can accomplish this by using button controls that manipulate the robot's functions. This is verified through demonstration of how pressing the buttons controls the actions of the robot. The final requirement is the constraint requirement of the mass of the robot weighing less than 3 kg. This can be verified by using a scale to measure the entirety of the robot.

2.0 Conceptual Design

2.1 Morphological Matrix


wheel
motor
colour sensor
Song 1
Song 2
Song 3
Song 4

SUB FUNCTION/ SOLUTION	Solution 1	Solution 2	Solution 3
Read physical sheet music	Colour sensor moving on a wheel powered by a motor to read the color-coded notes	Sensor is connected to 2 motors along the x and y axes to read the sheet of music	Colour sensor reads a variety of colors from the RGB scale where the higher the number of the colour on the RGB scale, the higher the note in pitch
Save song as a .txt file that can be played	4 save buttons where each save a different song and can be replayed by pressing it	Middle button saves the song once it is read, up button plays the song, left and right button chooses save slot seen on display to play	Middle button plays the song, up button saves it to a .txt file on the computer, and song selection is done using the computer
Press down keys	2 or more beams move sideways across the different piano keys, while moving vertically up and down to press the keys.	Wheel spins on a stand or axle and two arms play a variety of keys	Ten arms on a stand that is controlled by code and plays on the key assigned to it

Figure 1: Morphological Matrix

The piano robot's subfunctions include reading sheet music, saving the song as a.txt file, and pressing piano keys with robotic fingers. Three solutions were considered: vertical color sensors, horizontal and vertical sensors, and fixed sensors. The chosen solutions balanced efficiency, cost, and complexity, ensuring the robot could effectively read sheet music.

2.2 Conceptual Designs Considered

These three designs were the conceptual ones formulated during the initial stages of the design process. Each design contributed various components that were incorporated into the final product. However, it primarily utilizes the most effective concepts from conceptual designs 2 and 3, which eventually resulted in the final product.

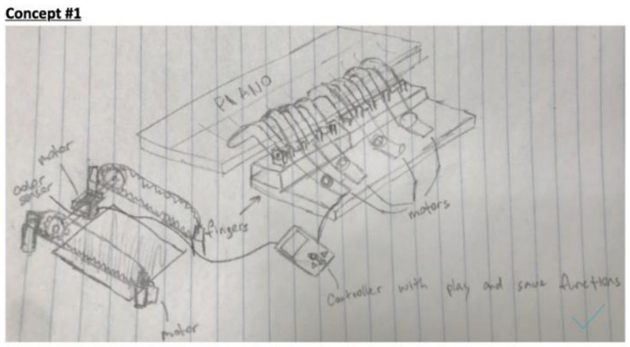


Figure 2: Conceptual Design 1

The first conceptual design was the inspiration for the sheet music reader. More specifically, the drawing shows a design utilizing suction rollers to roll the sheet music beneath the color sensor. The color sensor reads the colors on the sheet music and translates them into notes to be played by the robotic hand. Furthermore, the fingers of this design are not accurate to the final design, as it consists of multiple fingers, while the final design only has two fingers. This is one main difference between the conceptual design and the final product. Finally, the controller in this conceptual design was used for the same purpose in the final design, which was to operate and save songs that were already played.

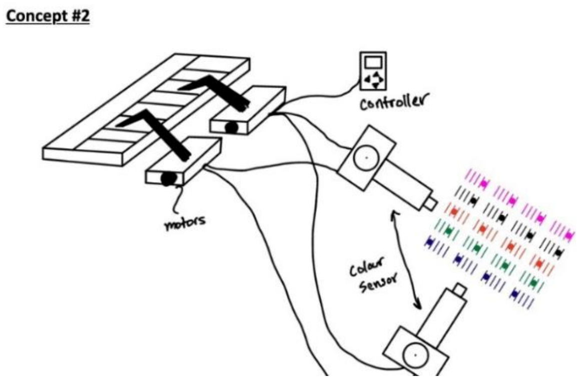


Figure 3: Conceptual Design 2

The second conceptual design served as the inspiration for the robot's fingers. The final product mirrored this design, featuring two separate robotic fingers, each supported by two motors and controlled through the controller. This configuration allowed for precise and independent movements of each finger, enhancing the robot's ability to perform complex tasks. Furthermore, the color reader, which was part of this design, proved to be somewhat accurate in its function. However, it didn't include specifications for the suction roller and conveyor belt design used for the sheet music, which were later incorporated into the final design of the robot. These additional components were essential for the accurate handling and processing of sheet music, ensuring smooth operation. Overall, this conceptual design was primarily used for robotic fingers, significantly influencing the development of the final product and contributing to its overall functionality and effectiveness.

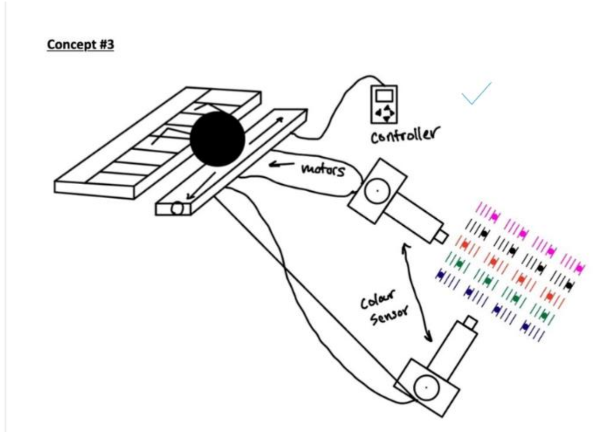
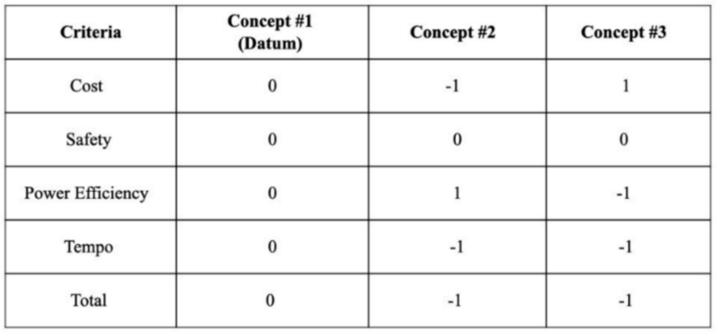


Figure 4: Conceptual design 3

The final conceptual design proved impractical due to the difficulty of creating a rolling mechanism that kept the robotic fingers in place for accurate key strikes. Balancing the mobility of the base with the stability and precision of the fingers was complex. Maintaining smooth and consistent motion without the fingers shifting was essential for accuracy, making the design challenging to implement and impractical for real-world application without significant modifications.

2.3 Decision Making Matrix



Criteria	Concept #1 (Datum)	Concept #2	Concept #3
Cost	0	-1	1
Safety	0	0	0
Power Efficiency	0	1	-1
Tempo	0	-1	-1
Total	0	-1	-1

Figure 5: Decision Making Matrix

The decision-making matrix essentially evaluates the feasibility of each conceptual design by ranking each important requirement (e.g., cost, safety, etc.). It ranks each requirement based on the following criteria: +1 being better, 0 being mediocre, and -1 being not as good. Moreover, the total score is summed up at the end, and the conceptual design with the highest score is expected to be the most feasible.

From the above, conceptual design 1 is the most feasible as it has the highest score of zero, while the two other conceptual designs both got a score of negative one. However, the final decision was made to proceed with the second conceptual design for the note-playing section of the robot because the first conceptual design was too simple. As for the robot's music reader section, a new design was needed as the current designs may be too difficult to do and limit the sheet music's size. This new design will be detailed in the next section.

2.4 Updated Mechanism Designs

For the sheet music reader part of the robot, a new design was created with a compact Tetrix frame, where a motor, mounted to the Tetrix frame, turned a suck roller. This suck roller would pull in the sheet music, assisted with two free-rolling rollers which would keep the sheet music properly tensioned. A color sensor bracket would hold two color sensors in place, which would read the sheet music as it is being pulled by the suck roller. This design can be seen below, in figure 6.

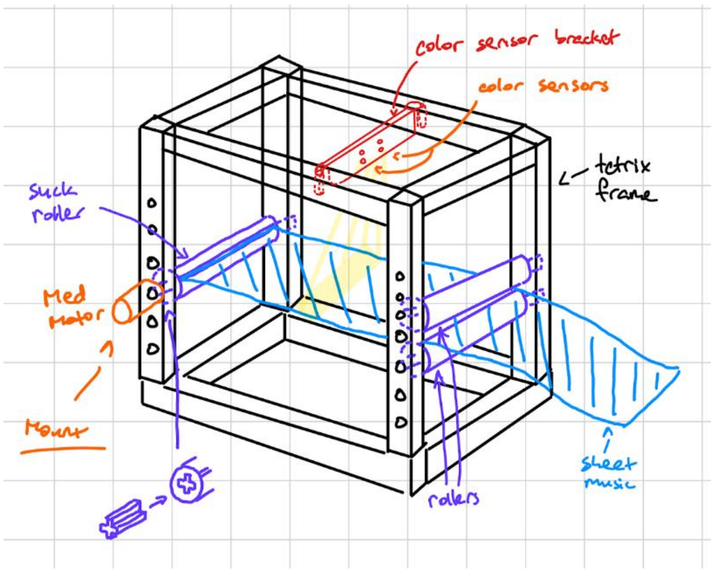


Figure 6: New sheet music reader design

For the note-playing part of the robot, a more detailed design of the arm movement mechanisms was created, as seen in figure 7. There would be one motor to control the horizontal rotation of the arm (to move to different keys) and another motor to control the vertical rotation of the arm (to press down on the key). The horizontal rotation motor had a gear attached to it, which meshed with another gear, which had an arm mount/base attached to it. This arm mount/base held the arm (with a gear attached to it), along with the vertical rotation motor (with a gear attached to it). This meant the vertical rotation motor would rotate horizontally with the arm while it is operating.

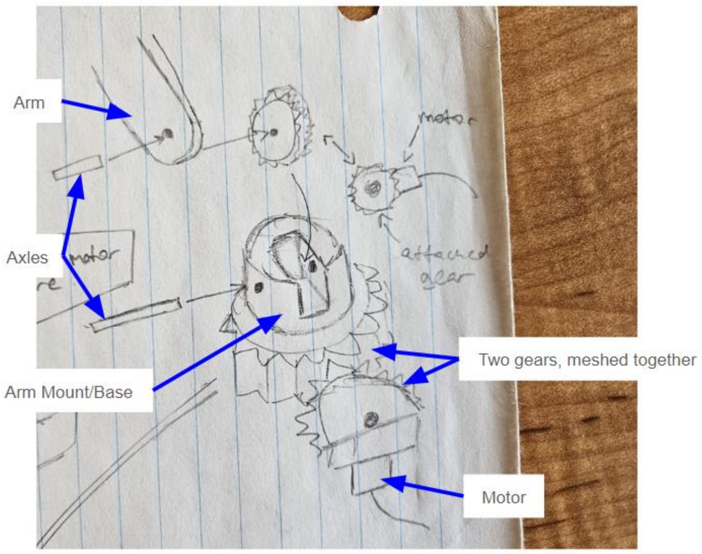


Figure 7: Design of the arm movement mechanisms

Additionally, a recalibration system was to be implemented. Two ideas were considered for the recalibration method: an ultrasonic sensor for each arm to detect the distance of the arm as it rotates, or a touch sensor for each arm to detect when the arm reaches a certain point. Ultimately, it was decided that the touch sensor method would be more reliable as the ultrasonic sensor could pick up unreliable readings due to the way the arm rotates. The recalibration function would involve the arm rotating until it pressed the touch sensor, at which point, the robot would know the exact position of the arm and would re-position the arm in its starting position.

2.5 Individual Part Designs

Sketches were made for parts that needed to be 3D-printed or laser-cut (appendix E), which included the suck roller, tensioning rollers, motor mount, color sensor bracket, and fingers. These sketches provided a rough starting idea for the CAD design and only had some rough dimensions.

For the tensioning rollers, the pegs at the ends of the roller were meant to fit in the Tetrix holes and be able to freely rotate. The size of the rollers was chosen so that two rollers next to each other in the Tetrix frame would result in a very small gap.

The suck roller had the same dimensions as the tensioning rollers, except for two changes: a slot for sheet music and a cross-shaped hole on the end of one of the pegs for a Lego axle to be inserted in. The cross-shaped hole would allow the motor to connect to the suck roller and rotate it.

The motor mount was designed to serve two purposes: hold the motor in place and mount to the Tetrix frame securely. Only a rough concept sketch of the motor mount was created; the actual design of the motor mount was to be adjusted during the CAD modelling stage.

For the color sensor bracket, it needed to be able to fit on the Tetrix frame without moving. Two pieces at the ends of the bracket would be inserted into the Tetrix frame holes. In the middle of the bracket, there would be four holes for the color sensors to mount to.

For the finger, a realistic finger shape was desired. Some cross-shaped holes were included so that it could be attached to the arm mount/base. The finger should also be thin so that it can play between the black keys.

3.0 Mechanical Design

3.1 Layout

The primary robot, designed using Lego and laser-cut parts, holds a piano and presses its keys. It has two individual Lego beams that pivot sideways while also moving up and down vertically to hit the piano keys. These arms are powered by two sets of motors and gear systems for each robotic finger. As seen below in figure 8, the larger motors and gears on the base control the rotation of the robotic arms along the horizontal axis. These motors are connected to a rectangular base and two rectangular prisms as shown in figure 9. Supporting beams secure the motors to the axle, and the right arm has one side of the robot also includes an additional stopper to limit the finger’s vertical motion of the finger upwards. On the other design, the finger’s upward motion is limited beyond a certain point by the bottom edge of the finger, as it presses against the base of the axle holder on the rotating structure. This is visible in figure 10:

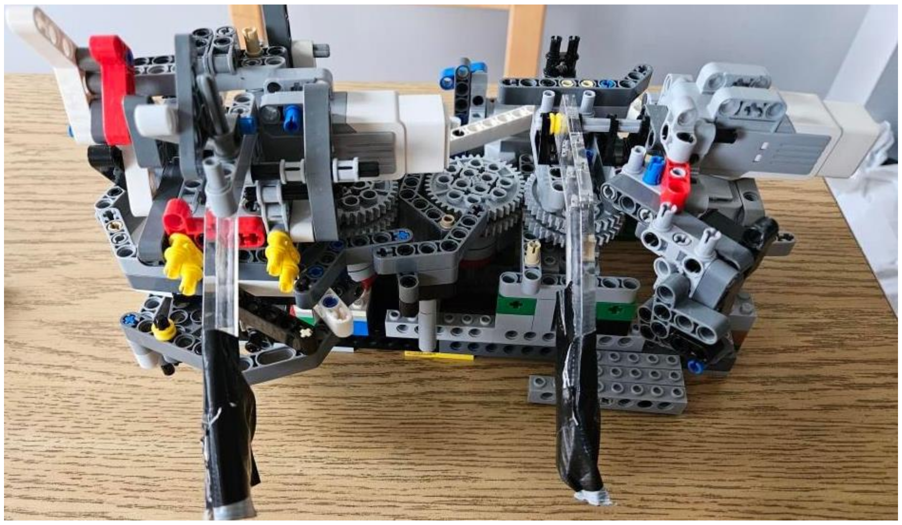


Figure 8: Top view of piano-player

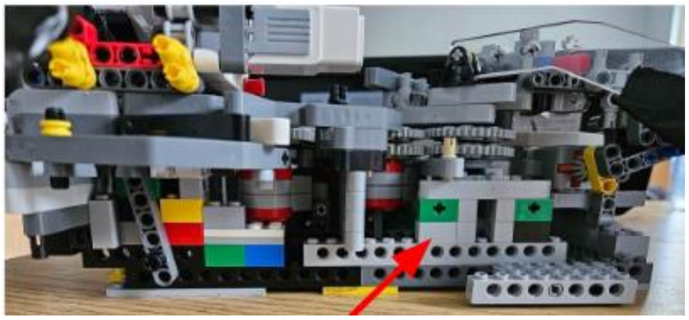


Figure 9: Front view of piano-player showing the rectangular base

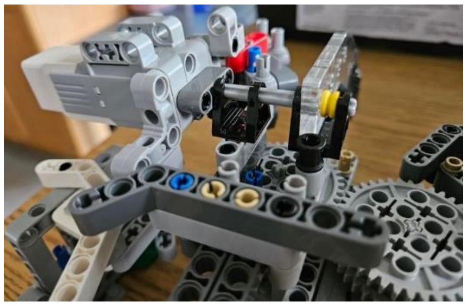


Figure 10: Side view of finger axle

The gear attached to the motor is held tightly in place with the gear attached to the moving structure through Lego walls that push them together from each side. The rotating base that houses the finger is attached to one of the gears, which allows the finger to move in an arc between each key. Each moveable base also includes a smaller motor attached directly to the axle which moves the fingers up and down to press each key. This design allows the motors to be rigidly attached and provide enough downward force. As shown in figure 11, the robot fingers themselves were laser cut from acrylic, which provided high strength while also being relatively light. This allowed the fingers to withstand the higher forces required to press down some of the further keys at wider angles. The fingers were also cut to angles and lengths to press down each key with maximum force, while also limiting its range of motion past the necessary requirements to hit the furthest keys. Piece of cardboard were also duct taped to the edges of the fingers to remove the unpleasant noise that resulted from the acrylic striking the keyboards plastic keys. These fingers included cross-shaped cuts in the exact shape of the axle to fit tightly, with small tolerances. This allowed the fingers to be powered by the motor attached to the axle from the other side, while remaining rigid when higher power was used on the motor. In front of the robot is a rectangular base that holds the piano in place. As shown in figure 12, the user can easily place the piano in the correct spot during each use by placing the piano in a way that holes on each piano fit into cylindrical Lego prices from each corner. This ensures the piano is placed in a position that allows the fingers to reach each key.

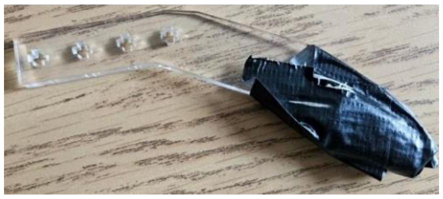


Figure 11: Robotic finger. The AutoCAD model can be found in appendix B, figure 40.

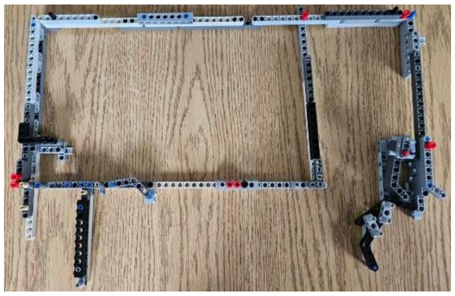


Figure 12: Robot piano base

One of the issues in the original design was the rigidity of the moveable base structure, and its ability to withstand reaction forces that resulted from pressing down each key. When the robot attempted to press down certain keys, especially further ones at larger angles, the base holding the finger and its base would move upwards, instead of pressing the keys downwards. This was solved using two separate methods. Firstly, the gear attached to the base structure that held the finger and its motor was attached to an axle, which was then fed through 2 different holes through pieces attached to the main base that held the gear structure. The base gear and axle structure are shown in figure 13. Also, the axle was held in place from moving up or down using a small Lego axle holder and a small, but thick, unused Lego gear that fit tightly on the axle. This ensured that the gear was held in place, and only rotated, without any up or down movements off its base. After the moveable base itself was secured tightly, the finger and axle attached to the base also needed to be secured, since they moved upwards when the keys were pressed down. This was accomplished by building a guide rail around each of the rotating bases, attached to the bottom base of the robot. Extruding Lego pieces were then added to the moveable bases which moved on top of the guide rail. This ensured that the upwards reaction force from pressing down the keys were counteracted by the guide rail that held in place the base and the finger attached to them. These components are highlighted in figures 14 and 15. This improved rigidity allowed the robot to completely press down the keys, while also using less motor power, since less energy was lost through movements within the robot itself.

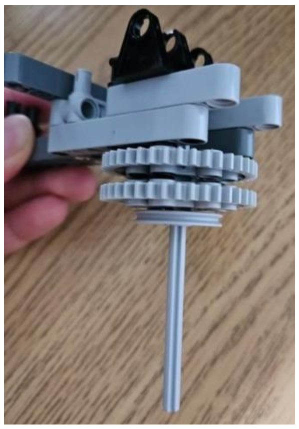


Figure 13: Gear and axle for rotating base

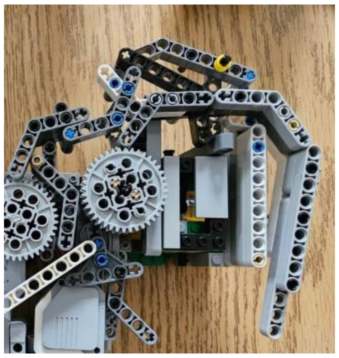


Figure 14: Guide rail on the outer side

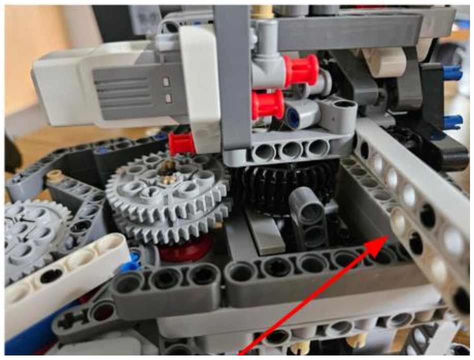


Figure 15: Extrusion that moves along guide rail

For the music sheet reader, a cube-like design was created with four vertical beams on each side, as shown in figure 16, and described in the previous section. Unlike robotic arms, this design was built using Tetrix parts, as well as laser cut, and 3D printed components. In this design, a motor was used to turn a 3D printed conveyor belt which took the fed paper sheet music in a thin slot. The motor was attached to the frame using the holder show in figure 17. This suck roller was alongside two free-rolling rollers that helped keep the sheet music properly tensioned and aligned in place for the color sensors. The suck roller and compressive rollers are shown in figures 18 and 19. These two color sensors were attached to the frame using a laser cut bracket that fit into the Tetrix frame, shown in figure 20. This allowed the two color sensors to be positioned to read the left and right sides of the sheet music, while also being properly tensioned to keep the colored sheet music straight, while also feeding at the appropriate speed. Figure 21 shows the sheet music being fed into the sheet music reader.

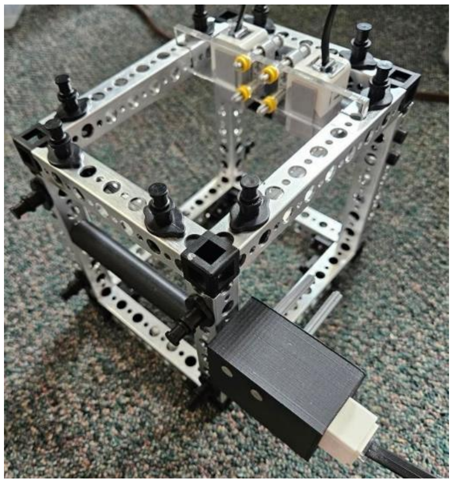


Figure 16: Music sheet reader

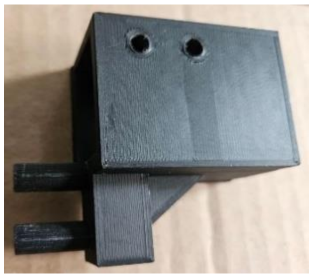


Figure 17: Sheet music motor holder. The SolidWorks model can be found in appendix B, figure 37.

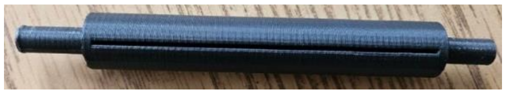


Figure 18: Sheet music suck roller. SolidWorks model can be found in appendix B, figure 36.

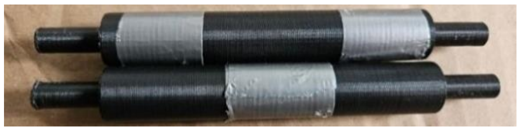


Figure 19: Sheet music compressive rollers*. The SolidWorks model can be found in appendix B, figure 36.*

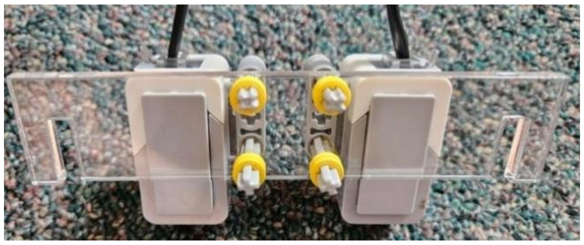


Figure 20*: Bracket holding the two-color sensors. The AutoCAD model can be found in appendix B, figure 39.*

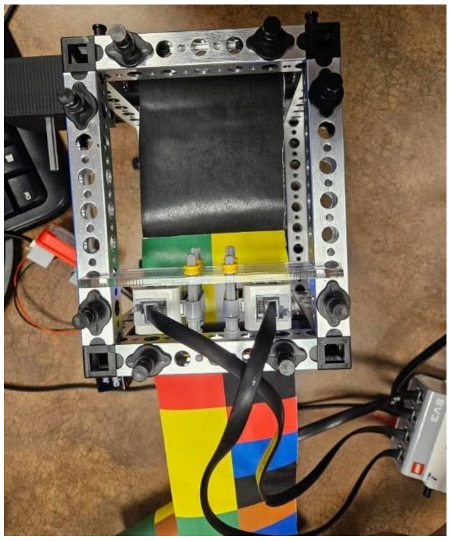


Figure 21: Sheet music reader pulling in sheet music.

3.2 Functions

There are a multitude of functions for the piano robot that all need to be met for it to successfully operate. Furthermore, the functions of this robot were generalized into 6 main functional requirements that had to be met for a successful final product. These included:

i. The robot should be able to successfully play the notes with minimal time delay
ii. The robot should be able to read and play the colored sheet music
iii. The robot reads in the color sensor combinations successfully
iv. The robot should have a range of notes that spans roughly two octaves
v. The maximum amount of notes a robot can play in a song should be 70
vi. The robot should be able to handle user input successfully (via the EV3 controller)

The robotic fingers' ability to move between notes quickly (Function I) was achieved through a lightweight design and precise angle measurements. The conveyer belt design for the sheet music (Function II) allowed the color sensor to register each note, with software ensuring accurate note correspondence. The optimal speed for the sheet music to roll beneath the color sensor (Function III) was found using suction rollers to impose tension. Stability of the robotic arms (Function IV) was ensured by the pivot of the gears, arm length, and a stable Lego base. The note-reading limit (Function V) was enforced through software, tested with a 70-note song. Finally, a straightforward coding solution (Function VI) generated a user interface on the EV3 controller for saving, playing, or exiting the program.

4.0 Verification of Design

4.1 Line by Line Verification

The following table consists of the engineering design specification and the method of testing for each specification.

Table

Table 1: Line by Line Verification of Engineering Design

No.	Characteristic	Relation	Value	Units	Verification Method	Comments
1	Cost	<	100	$, CAD	Analysis	Total the receipts of additional costs
2	Safety	N/A	N/A	N/A	Analysis	Ensure there are no pinch points, pointy edges, or potential fire hazards
3	Plays notes with minimal delay	<	1	s	Test	Use a stopwatch to measure minimal delay between each key press, then take the difference between the highest and lowest delays
4	Plays modified sheet music	N/A	N/A	N/A	Examination	Have an experienced pianist listen to the played notes and verify that the robot is playing the correct notes
5	Reads the color sensor combinations accurately	≥	80	%	Test	Output the notes read from the sheet music reader onto a .txt file or onto the robot's console cross-check with the predetermined color sensor
6	Range of notes	=	2	octaves	Test	Make the robot play the C major scale on two octaves, which will therefore hit every key from Low C to High C

7	Easy setup	N/A	N/A	N/A	Demonstration	While assembling the robot, show that the system is assembled every time without damage to the components, and any components falling out of place
8	Maximum amount of notes the robot can play in a song	=	70	notes	Test	Make the robot play a .txt file with over 70 notes and see if the robot completely stops playing at 70 notes
9	Robot includes user input	N/A	N/A	N/A	Demonstration	Have volunteers press buttons on the robot to control its actions which will therefore display user input
10	Mass of the entirety of the robot	<	3	kg	Test	Use a scale to measure the entirety of the robot

The verification results are as follows:

1) After totaling all receipts, the final cost is $44.00 which is well below the maximum cost planned for this project
2) After running through the robot's physical mechanism, there are no pinch points, pointy edges or potential fire hazards that came across the PianoBot
3) After taking measurements using the stopwatch, the highest delay was 0.94 seconds and lowest delay was 0.52 seconds, giving a delay well under 1 second
4) After running modified sheet music through the sheet music reader, the PianoBot was able to play those corresponding notes
5) After making the robot read the sheet music, the group calculated the number of notes that were correctly read and divided it by the total notes that were read. This gave a value of 87%, which was over the target of 80%
6) After testing the robot with the C major scale, it was shown to play 2 full octaves from Low C to High C
7) After countless times of setting the robot up, it was time and time again demonstrated that none of the components slipped off or broke

8) After making the robot read a .txt file with over 70 notes, the robot only played 70 and stopped after the 70th note, proving the test correct
9) After having volunteers test out the robot on demo day, the user input aspect is fulfilled using buttons either playing a different .txt file or saving read notes into a .txt file
10) After weighing the entire robot structure, the total mass came out to be 1.947 kg which is well under 3 kg.

4.2 Assessment of Verification

After running through all the verification tests for all functional requirements, non-functional requirements, and constraint requirements, it is safe to say that the design works as it passed all the tests and fulfilled all requirements. While it does not play the piano or read the notes with 100% accuracy, it fulfilled all predetermined goals and therefore is a success.

5.0 Software Design and Implementation

5.1 Overall Software Operation

The software operates by starting with the user pressing the enter button, recalibrating the arm, and then prompting the user to press a button. If the button is pressed for under 3 seconds, the sheet music reader plays the music, if it's held for 3 seconds or more, the note values are saved, and the program shuts down. The flow chart in the Appendix explains the process.

5.2 Robot Task List

The robot task list is as follows:

Startup:

- Configure all sensors.
- Run arm re-calibration function.
  - Arm spins until a touch sensor is hit, ensuring the position of the arm is calibrated.
- Display on controller screen: “Press enter button to play music from sheet music. Press up/down/left/right buttons to play saved music. To exit the program, hold the enter button.”

Operation:

- User presses the enter button.
- Turn on the motor that connects to the suck roller.
- Two colour sensors detect two colours from the sheet music, which are translated into a note value that is associated with an angle for the arm motors to rotate.
- An arm motor rotates so that the finger hovers over the note that corresponds to the colour combination detected by the colour sensors.
- When a certain colour combination is detected (eg. black and black), this means the song has ended
- Display on controller screen: "Hold up/down/left/right buttons to save the song. Press up/down/left/right buttons to play saved music. Press enter button to play music from sheet music. To exit the program, hold the enter button."
- If any of the up/down/left/right buttons are held:
    - Save the note values into a text file.
    - Display on the controller: "Song saved to up/down/left/right button slot."
    - Display on controller screen: "Hold up/down/left/right buttons to save the song. Press up/down/left/right buttons to play saved music. Press enter button to play music from sheet music. To exit the program, hold the enter button."
- If any of the up/down/left/right buttons are pressed:
    - Read the note values from the associated text file and play the song.
    - A finger motor rotates so that the finger presses down on the note, then rotates the opposite way so that the finger returns to its previous position.
    - The robot will run the arm re-calibration function
    - Display on controller screen: "Hold up/down/left/right buttons to save the song. Press up/down/left/right buttons to play saved music. Press enter button to play music from sheet music. To exit the program, hold the enter button."

Shutdown Procedure:

- If the enter button is held:
    - Display on controller screen: "Skee-yee" and "EEEEEE"
    - Exit the program.

5.3 How Overall Code was Separated

The code was divided into smaller parts based on different functions, such as recalibration, which was separated from the rest to avoid confusion. The group used at least 6 non-trivial functions to separate repetitive or self-performing parts.

5.4 Functions Breakdown

Table 2: How Each Function Works

Function	How it Works	Programmer
void configureAllSensors();	A modified version of the configureAllSensors function from Professor Carol Hulls. For the source code, see appendix A, figure 24.	Marco Xu
void storeValue (short * tempStoredValues, short * colour1, short * colour2, int noteIndex);	Uses the two color sensor readings to determine the corresponding note value, then stores it in the tempStoredValues array. For the source code, see appendix A, figure 25.	Nitai Michalski
void readAndStore(short * tempStoredValues, short * colour1, short * colour2);	Reads the sheet music and calls the storeValue function. For the source code, see appendix A, figure 26.	Omer Farooq
void rotateArmAndPlay(int angleNew, int angleOriginal, int noteValue, short * timeHold);	Based on the note value, it decides which arm to rotate (left or right). Then, depending on the arm's position, it rotates clockwise or counterclockwise to the key, then plays the key. For the source code, see appendix A, figure 27.	Marco Xu
void playFromFile(TFileHandle & fin, short * angle, short * timeHold);	Reads in note values from a file, then calls the rotateArmAndPlay function. For the source code, see appendix A, figure 28.	Seyon Menakan
void playFromReader(short * tempStoredValues, short * angle, short * timeHold);	Reads in note values from tempStoredValues array, then calls the rotateArmAndPlay function. For the source code, see appendix A, figure 29.	Seyon Menakan
void arrayToSaveFile(TFileHandle & fin, short * tempStoredValues);	Writes note values from the tempStoredValues array to a file. For the source code, see appendix A, figure 30.	Omer Farooq
void buttonOptions(TFileHandle & saveUp, TFileHandle & saveLeft, TFileHandle & saveDown, TFileHandle & saveRight, short * angle, short * timeHold, short * tempStoredValues);	Based on how long a button is held, it runs the playFromFile function or arrayToSaveFile function. The button being pressed determines the parameters in the function call. For the source code, see appendix A, figure 31.	Nitai Michalski
bool playOrSave(TFileHandle & saveUp, TFileHandle & saveLeft, TFileHandle &	Based on button input, it calls the readAndStore function and the playFromReader function, or the	Marco Xu

saveDown, TFileHandle & saveRight, short * angle, short * timeHold, short * tempStoredValues, short * colour1, short * colour2);	buttonOptions function. Holding the enter button causes the function to return true, so that the loop in the main program exits. Otherwise, the function will return false. For the source code, see appendix A, figure 32.	
void recalibrate();	Recalibrates the left arm by rotating the arm until it hits the touch sensor, then rotating the arm to its starting position. Initially, both arms were to be recalibrated with a touch sensor for each arm, but due to a port shortage, only the left arm would be recalibrated with a touch sensor, and the other arm would just be manually recalibrated. For the source code, see appendix A, figure 33.	Seyon Menakan
void displayMenu();	This function essentially displays the menu with the different options to the user. This includes saving the song, playing a saved song or quitting the program. For the source code, see appendix A, figure 34.	Omer Farooq

5.5 Data Storage

This program stores data in arrays or.txt files. The colour1 and colour2 arrays determine note values for sheet music reading. The tempStoredValues array stores note values from the sheet music reader or notes in.txt files. The angle and timeHold arrays determine the robot's finger movement and time holding piano keys. SaveUp, saveLeft, saveDown, and saveRight .txt files store note value data for buttons held for 3 seconds or more.

5.6 Software Design Decisions

One software design decision implemented was saving memory by turning the array values from integers to shorts. Due to having a large code with a lot of saved data, when having int arrays, the memory of the Lego EV3 robot becomes full. Therefore, by significantly reducing the memory sizes of the arrays, the group managed to clear wasted memory space. Another decision used was how to read the color sensor combinations. The group decided to store the values by using two color arrays and to run through a loop where both arrays would match and give an index value which becomes the note value.

5.7 Code Testing

The configureAllSensors function was tested by first turning on the suck roller motor, which was connected to the motor multiplexer, at a speed of 10 without calling the configureAllSensor function. Without the configureAllSensors function, the motor should only be able to turn at max speed. After making note of the motor speed, the program was modified to call the configureAllSensors function before turning on the suck roller motor at a speed of 10. The motor turned significantly slower in the second test than the first test, showing that the configureAllSensors function was properly configuring the suck roller motor. The other sensors were assumed to be properly configured as they were just a modified version of the configureAllSensors function written by Professor Hulls.

Next, the displayMenu function was tested by calling the function and checking the robot controller display. It was verified that the function was working properly when the menu text was properly displayed on the controller screen and it was erased after a short delay.

After, the arrayToSaveFile function was tested by having the function read a pre-defined tempStoredValues array and write the numbers from the array into a file. The program was run, the file on the robot was checked, but the numbers were not on the file. The issue turned out to be because the file was not closed in between converting the file from read-only to write-only, and vice versa. After making these changes, the numbers were on the file, as expected.

Similarly, the buttonOptions function was tested by having a pre-defined tempStoredValues array and calling the arrayToSaveFile function. After running the program, the file on the robot was checked, which showed the numbers from the array on the file.

The storeValue function was tested by adding test code to display the note value after it was found. The color sensors were held over a set of colours, then the program was run, and the note value displayed was checked with the spreadsheet.

The playOrSave function and readAndStore function were tested together. First, holding the enter button for more than three seconds ended the program, which was the expected result. Next, the reading and storing functionality was tested by holding the enter button for less than three seconds. The suck roller motor was turned on until it reached the end of the sheet music. There were some issues with the motor stopping before the end of the sheet music or continuing after the end of the sheet music, but these were resolved by increasing the tensioning on the sheet music and changing the position of the color sensors in relation to the sheet music. The note values were read and stored in the tempStoredValues array. Then, the up button was held (which called the buttonOptions function), transferring the values from the tempStoredValues array to the saveUp file. The file was checked, and the numbers on the file were compared to the spreadsheet containing the correct numbers. After resolving tensioning issues and reading issues, the numbers were mostly correct. The incorrect numbers were likely caused by suboptimal setup of the color sensors in relation with the sheet music.

The recalibrate function was tested by ensuring that the robot hit the touch sensor, then returned to its starting position. A couple angle measurements were needed to ensure the robot consistently rotated to its starting position.

The rotateArmAndPlay function was tested together with the playFromFile and playFromReader functions. The enter button was pressed to activate the sheet music reader. Next, the playFromReader function was called, which used the rotateArmAndPlay function to play the notes. After ensuring that the arm was moving to the keys as expected, the up/down/left/right buttons were pressed to call the playFromFile function, which also used the rotateArmAndPlay function to play the notes. It was verified that the arm was moving to the keys as expected.

Finally, the entire program was tested. After configuring the sensors, recalibrating, and displaying the menu, the enter button was pressed, which activated the sheet music reader. After the sheet music reader finished reading, the robot played the notes that were read by the sheet music reader. When the robot finished the song, it recalibrated and displayed the menu. The up button was held to save the song into the saveUp slot. The up button was pressed to replay the song. Next, the left, right, and down buttons were pressed to play the song saved in those slots. Finally, the enter button was held to exit the program. These functions were performed as expected, concluding the test of the robot.

5.8 Problems Encountered and Solutions

The testing process faced issues with color sensor combinations, which led to inaccurate results. To address this, a timer was used to stop the motor in the middle of color boxes, increasing accuracy to over 85%. The angles array was also problematic, with sometimes inaccurate angles for each note. Troubleshooting included using a console display message to fix logic errors and ensure smooth code execution.

6.0 Project Management

6.1 Task Delegation

The following table displays the tasks each person of the group did:

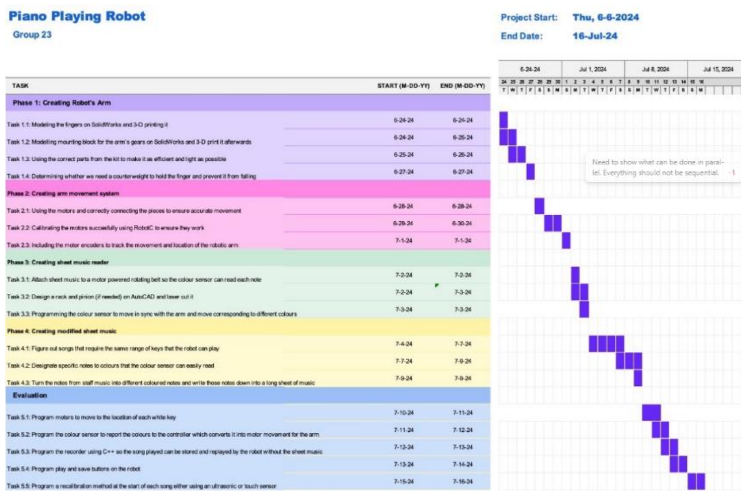
Table 3: Task Delegation

Group Member	Tasks
Marco Xu	• Created all SolidWorks drawings • 3-D printed all components • Created sheet music color combinations

	• Created modified configure sensors function • Created rotate arm and play function • Created play or save function • Contributed to task main code • Contributed to report writing
Nitai Michalski	• Worked on piano playing mechanism of PianoBot • Helped 3-D print all components • Created sheet music color combinations • Created store values function • Created button options function • Contributed to task main code • Contributed to report writing
Omer Farooq	• Worked on sheet music reader mechanism of PianoBot • Helped laser cut all components • Printed out the sheet music • Created read and store function • Created array to save file function • Created display menu function • Contributed to task main code • Contributed to report writing
Seyon Menakan	• Created all AutoCAD drawings • Laser cut all components • Created sheet music • Created play from file function • Created play from reader function • Created recalibrate function • Contributed to task main code • Contributed to report writing

6.2 All Schedule Revisions

When first creating a schedule, the group created a Gantt's chart to plan times to do specific parts of the project. The image under is the Gantt's chart first determined:



Piano Playing Robot
Group 23

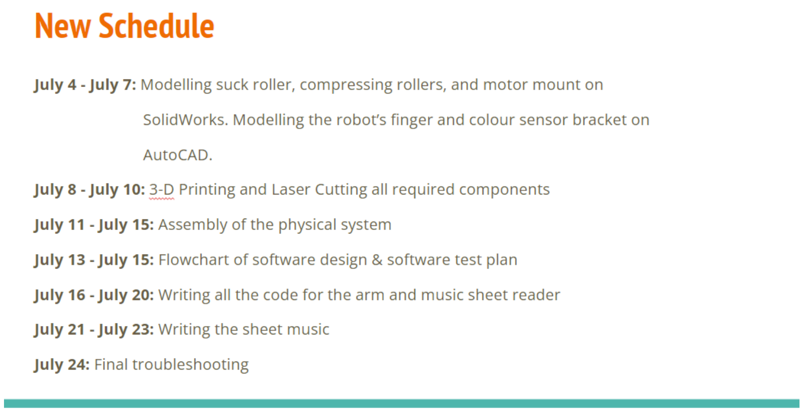
Project Start: Thu, 6-6-2024
End Date: 16-Jul-24

TASK	START (M-DD-YY)	END (M-DD-YY)
Phase 1: Creating Robot's Arm		
Task 1.1: Modeling the fingers on SolidWorks and 3-D printing it	6-24-24	6-24-24
Task 1.2: Modelling mounting block for the arm's gears on SolidWorks and 3-D print it afterwards	6-24-24	6-25-24
Task 1.3: Using the correct parts from the kit to make it as efficient and light as possible	6-25-24	6-26-24
Task 1.4: Determining whether we need a counterweight to hold the finger and prevent it from falling	6-27-24	6-27-24
Phase 2: Creating arm movement system		
Task 2.1: Using the motors and correctly connecting the pieces to ensure accurate movement	6-28-24	6-28-24
Task 2.2: Calibrating the motors succesfully using RobotC to ensure they work	6-29-24	6-30-24
Task 2.3: Including the motor encoders to track the movement and location of the robotic arm	7-1-24	7-1-24
Phase 3: Creating sheet music reader		
Task 3.1: Attach sheet music to a motor powered rotating belt so the colour sensor can read each note	7-2-24	7-2-24
Task 3.2: Design a rack and pinion (if needed) on AutoCAD and laser cut it	7-2-24	7-3-24
Task 3.3: Programming the colour sensor to move in sync with the arm and move corresponding to different colours	7-3-24	7-3-24
Phase 4: Creating modified sheet music		
Task 4.1: Figure out songs that require the same range of keys that the robot can play	7-4-24	7-7-24
Task 4.2: Designate specific notes to colours that the colour sensor can easily read	7-7-24	7-9-24
Task 4.3: Turn the notes from staff music into different coloured notes and write those notes down into a long sheet of music	7-9-24	7-9-24
Evaluation		
Task 5.1: Program motors to move to the location of each white key	7-10-24	7-11-24
Task 5.2: Program the colour sensor to report the colours to the controller which converts it into motor movement for the arm	7-11-24	7-12-24
Task 5.3: Program the recorder using C++ so the song played can be stored and replayed by the robot without the sheet music	7-12-24	7-13-24
Task 5.4: Program play and save buttons on the robot	7-13-24	7-14-24
Task 5.5: Program a recalibration method at the start of each song either using an ultrasonic or touch sensor	7-15-24	7-16-24

Need to show what can be done in parallel. Everything should not be sequential. -1

Figure 22: Gantt's chart

The group faced a significant issue with the Gantt chart due to changes in design components over time. They decided to create a more accurate schedule to align with the current project plan.



New Schedule

July 4 - July 7: Modelling suck roller, compressing rollers, and motor mount on SolidWorks. Modelling the robot's finger and colour sensor bracket on AutoCAD.

July 8 - July 10: 3-D Printing and Laser Cutting all required components

July 11 - July 15: Assembly of the physical system

July 13 - July 15: Flowchart of software design & software test plan

July 16 - July 20: Writing all the code for the arm and music sheet reader

July 21 - July 23: Writing the sheet music

July 24: Final troubleshooting

Figure 23: Updated schedule

This schedule was used for the remainder of the project working period and it was because of this schedule that the group was able finish the project successfully.

6.3 Project Plan vs. Actual Plan

The first project plan differed from the most updated one as with the newest one, the components were fully determined and the design outline was finalized whereas for the initial plan illustrated in the Gantt's chart, some components were still undecided upon. For example, one idea was to use a rack and pinion which was scheduled to be created in the Gantt’s chart. However, after spending more time gathering ideas for the design, the group decided that the rack and pinion will not be used for the final design and therefore does not need to be created. Moreover, working on this project for a bit every single day was not ideal but was laid out in the Gantt’s chart. With the improved schedule, there would be a range of days when the task would have to be completed, which was ideal for the group’s busy schedules.

7.0 Conclusion

Throughout this project, a robot was created that can play two octaves of a piano while costing less than $100. The PianoBot can also read modified sheet music and can store songs into .txt files that can be played on command. The songs can consist of any notes in a 2 octave range from Low C to High C but must be 70 notes in length. The design also consists of over 80% accuracy of reading the notes. The design does not fall apart while setting up and weighs much less than 3 kg. Therefore, it can be concluded that the final design met the requirements insisted upon.

8.0 Recommendations

As you take over the work on the piano playing robot, it's essential to identify areas for improvement and outline the next steps to enhance the project's success. Here are several key areas where advancements can be made:

1. One of the critical improvements needed is ensuring that the gears for the right finger are identical. This will help them align properly, resulting in smoother and more precise movements. Utilizing 3D printing technology can facilitate the creation of these gears with high accuracy and consistency.
2. To enhance the overall functionality and aesthetics of the robot, it is important to align the fingers correctly and clean up the design. Both sides of the design should be made more like maintain symmetry. This not only improves the robot's performance but also its visual appeal and structural integrity.
3. Expanding the piano keyboard can significantly reduce issues related to hitting the keys accurately. A larger keyboard provides more space, allowing the robot to maneuver

its fingers more effectively without the risk of missing or striking multiple keys simultaneously.

4. Upgrading to stronger motors can address the issue of momentum when hitting the keys. More powerful motors will ensure that each key is pressed with the necessary force, improving the accuracy and consistency of the robot's performance. This enhancement is crucial for playing more complex and dynamic pieces.

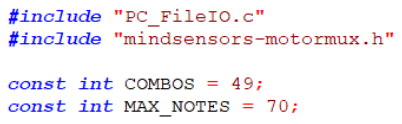
5. The current user interface can be made more interactive and feature rich. Enhancements such as the ability to remix songs, alongside an overall cleaner and more intuitive design, will provide users with a better experience. Improving the UI not only makes the robot more user-friendly but also opens new possibilities for creative expression and customization.

By focusing on these key improvements, you will be able to advance the functionality, reliability, and user experience of the piano playing robot. Aligning the gears and fingers, working with a larger keyboard, upgrading the motors, and enhancing the user interface are crucial steps that will contribute to the project's success. Your efforts in these areas will pave the way for future innovations and ensure that the robot reaches its full potential.

References

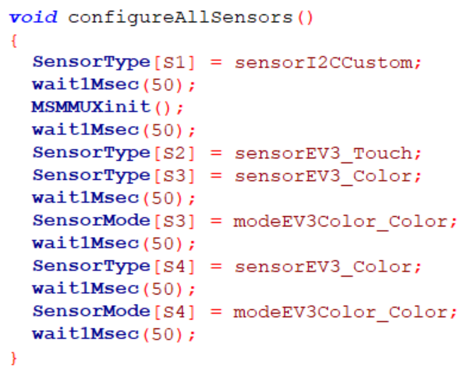
[1] “ME 101 project S24 - final report and demonstration details” https://learn.uwaterloo.ca/d2l/le/content/1021051/viewContent/5537825/View. (accessed 30 July 2024.)

Appendix A: Source Code



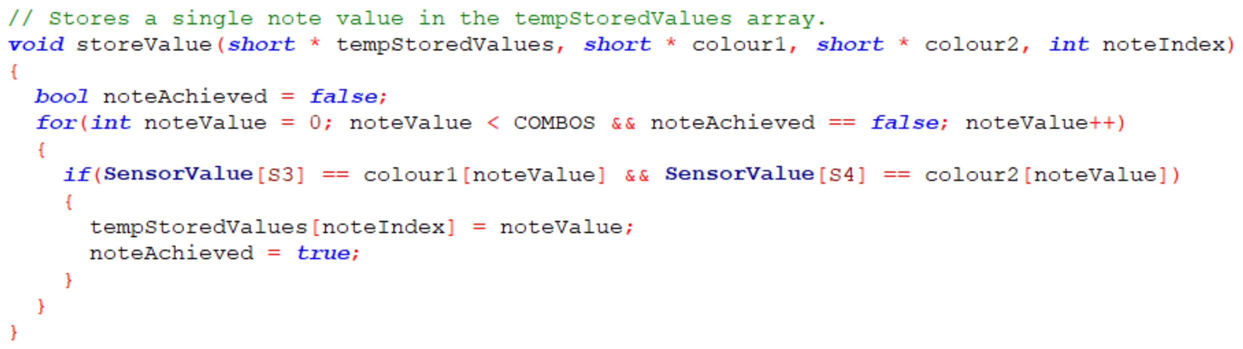
```
#include "PC_FileIO.c"
#include "mindsensors-motormux.h"

const int COMBOS = 49;
const int MAX_NOTES = 70;
```

Figure 24: Files included and global constants.



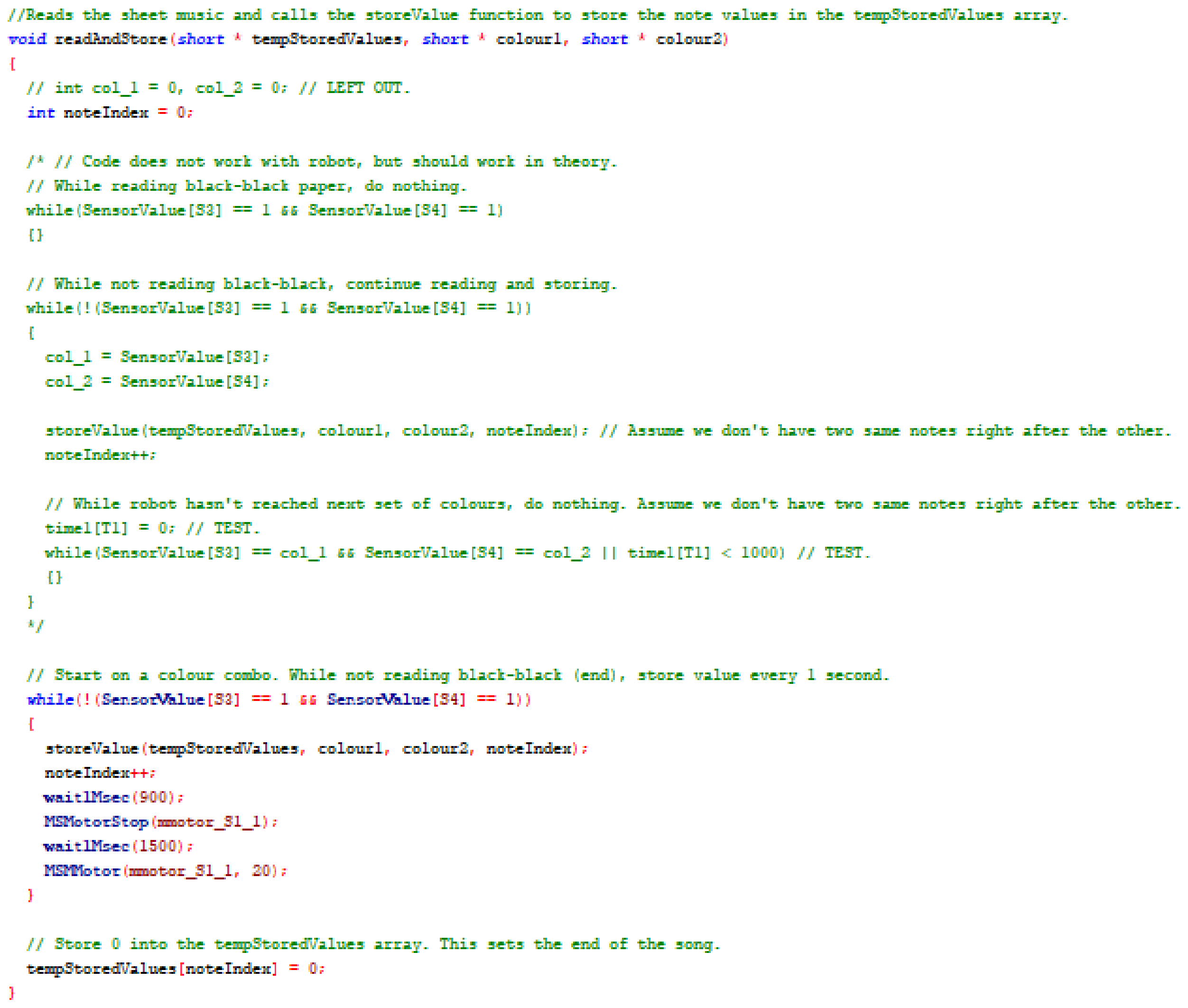
```
void configureAllSensors()
{
  SensorType[S1] = sensorI2CCustom;
  wait1Msec(50);
  MSMMUXinit();
  wait1Msec(50);
  SensorType[S2] = sensorEV3_Touch;
  SensorType[S3] = sensorEV3_Color;
  wait1Msec(50);
  SensorMode[S3] = modeEV3Color_Color;
  wait1Msec(50);
  SensorType[S4] = sensorEV3_Color;
  wait1Msec(50);
  SensorMode[S4] = modeEV3Color_Color;
  wait1Msec(50);
}
```

Figure 24: Configure all sensors function



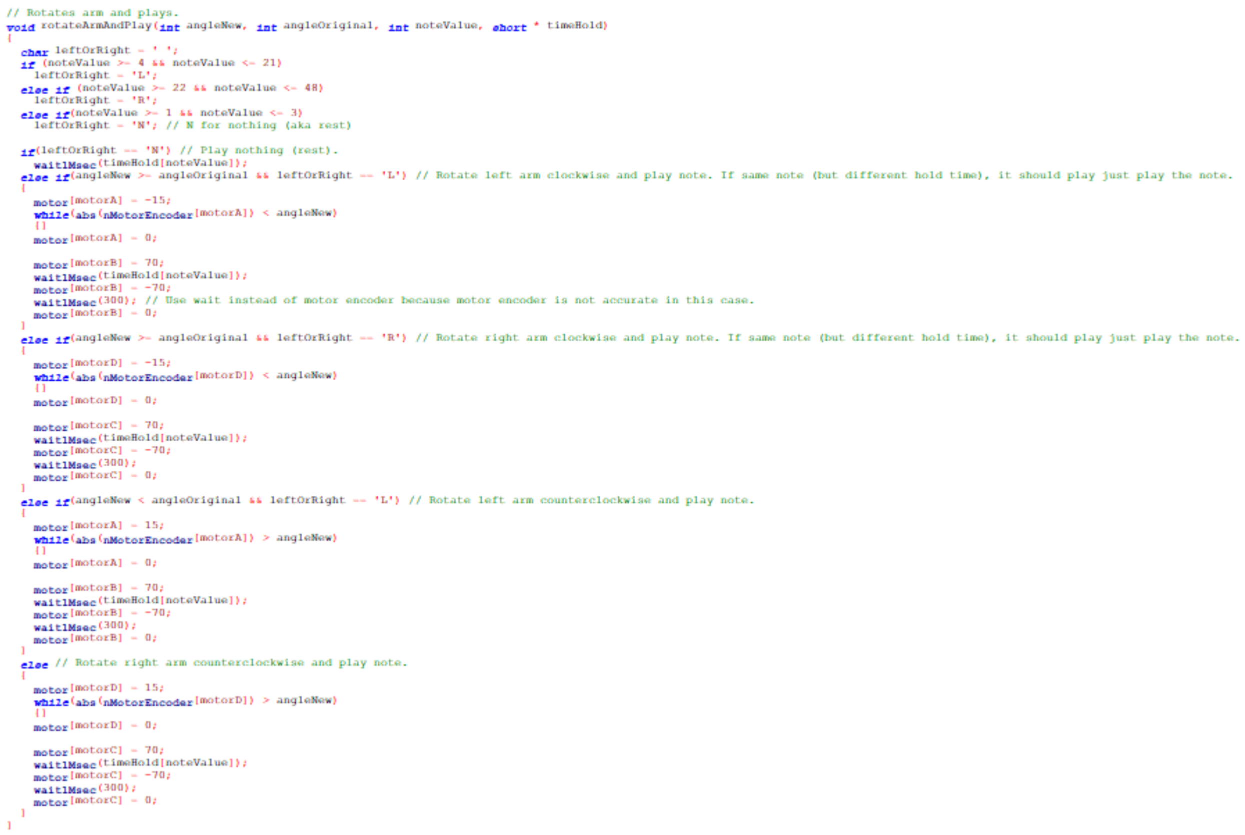
```
// Stores a single note value in the tempStoredValues array.
void storeValue(short * tempStoredValues, short * colour1, short * colour2, int noteIndex)
{
  bool noteAchieved = false;
  for(int noteValue = 0; noteValue < COMBOS && noteAchieved == false; noteValue++)
  {
    if(SensorValue[S3] == colour1[noteValue] && SensorValue[S4] == colour2[noteValue])
    {
      tempStoredValues[noteIndex] = noteValue;
      noteAchieved = true;
    }
  }
}
```

Figure 25: Store value function



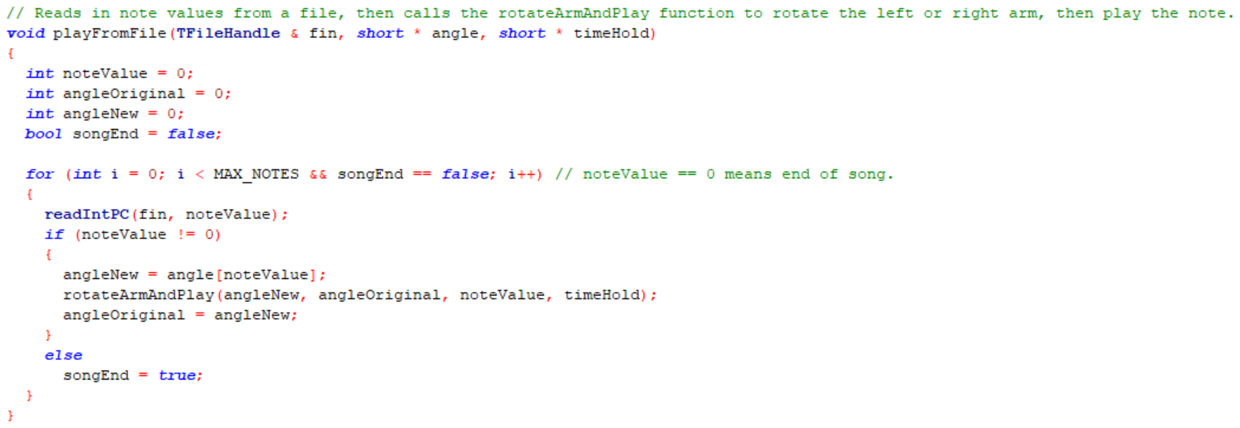
```
//Reads the sheet music and calls the storeValue function to store the note values in the tempStoredValues array.
void readAndStore(short * tempStoredValues, short * colour1, short * colour2)
{
  // int col_1 = 0, col_2 = 0; // LEFT OUT.
  int noteIndex = 0;

  /* // Code does not work with robot, but should work in theory.
  // While reading black-black paper, do nothing.
  while(SensorValue[S3] == 1 && SensorValue[S4] == 1)
  {}

  // While not reading black-black, continue reading and storing.
  while(!(SensorValue[S3] == 1 && SensorValue[S4] == 1))
  {
    col_1 = SensorValue[S3];
    col_2 = SensorValue[S4];

    storeValue(tempStoredValues, colour1, colour2, noteIndex); // Assume we don't have two same notes right after the other.
    noteIndex++;

    // While robot hasn't reached next set of colours, do nothing. Assume we don't have two same notes right after the other.
    time1[T1] = 0; // TEST.
    while(SensorValue[S3] == col_1 && SensorValue[S4] == col_2 || time1[T1] < 1000) // TEST.
    {}
  }
  */

  // Start on a colour combo. While not reading black-black (end), store value every 1 second.
  while(!(SensorValue[S3] == 1 && SensorValue[S4] == 1))
  {
    storeValue(tempStoredValues, colour1, colour2, noteIndex);
    noteIndex++;
    wait1Msec(900);
    MSMotorStop(mmotor_S1_1);
    wait1Msec(1500);
    MSMMotor(mmotor_S1_1, 20);
  }

  // Store 0 into the tempStoredValues array. This sets the end of the song.
  tempStoredValues[noteIndex] = 0;
}
```

Figure 26: Read and store function



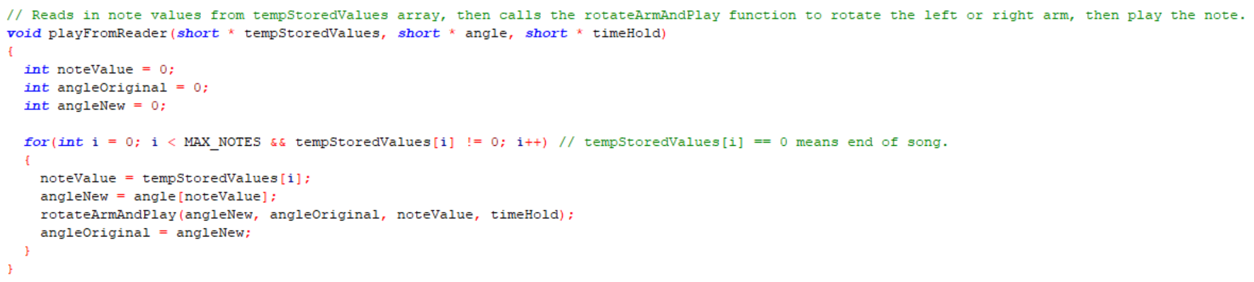
```
// Rotates arm and plays.
void rotateArmAndPlay(int angleNew, int angleOriginal, int noteValue, short * timeHold)
{
  char leftOrRight = ' ';
  if (noteValue >= 4 && noteValue <= 21)
    leftOrRight = 'L';
  else if (noteValue >= 22 && noteValue <= 48)
    leftOrRight = 'R';
  else if(noteValue >= 1 && noteValue <= 3)
    leftOrRight = 'N'; // N for nothing (aka rest)

  if(leftOrRight == 'N') // Play nothing (rest).
    wait1Msec(timeHold[noteValue]);
  else if(angleNew >= angleOriginal && leftOrRight == 'L') // Rotate left arm clockwise and play note. If same note (but different hold time), it should play just play the note.
  {
    motor[motorA] = -15;
    while(abs(nMotorEncoder[motorA]) < angleNew)
    {}
    motor[motorA] = 0;

    motor[motorB] = 70;
    wait1Msec(timeHold[noteValue]);
    motor[motorB] = -70;
    wait1Msec(300); // Use wait instead of motor encoder because motor encoder is not accurate in this case.
    motor[motorB] = 0;
  }
  else if(angleNew >= angleOriginal && leftOrRight == 'R') // Rotate right arm clockwise and play note. If same note (but different hold time), it should play just play the note.
  {
    motor[motorD] = -15;
    while(abs(nMotorEncoder[motorD]) < angleNew)
    {}
    motor[motorD] = 0;

    motor[motorC] = 70;
    wait1Msec(timeHold[noteValue]);
    motor[motorC] = -70;
    wait1Msec(300);
    motor[motorC] = 0;
  }
  else if(angleNew < angleOriginal && leftOrRight == 'L') // Rotate left arm counterclockwise and play note.
  {
    motor[motorA] = 15;
    while(abs(nMotorEncoder[motorA]) > angleNew)
    {}
    motor[motorA] = 0;

    motor[motorB] = 70;
    wait1Msec(timeHold[noteValue]);
    motor[motorB] = -70;
    wait1Msec(300);
    motor[motorB] = 0;
  }
  else // Rotate right arm counterclockwise and play note.
  {
    motor[motorD] = 15;
    while(abs(nMotorEncoder[motorD]) > angleNew)
    {}
    motor[motorD] = 0;

    motor[motorC] = 70;
    wait1Msec(timeHold[noteValue]);
    motor[motorC] = -70;
    wait1Msec(300);
    motor[motorC] = 0;
  }
}
```

Figure 27: Rotate arm and play function



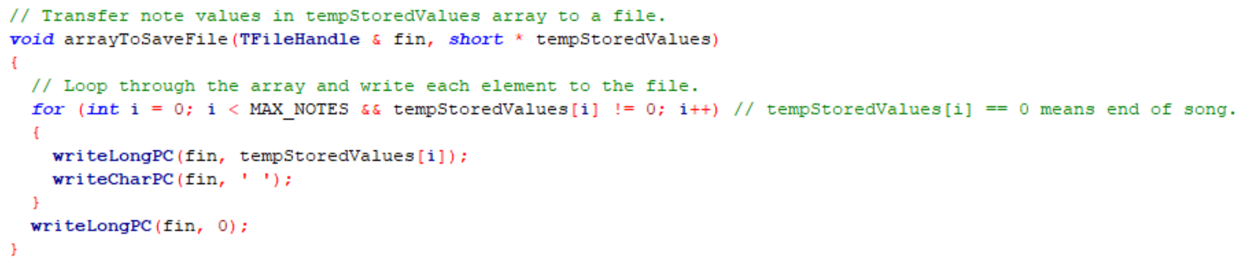
```
// Reads in note values from a file, then calls the rotateArmAndPlay function to rotate the left or right arm, then play the note.
void playFromFile(TFileHandle & fin, short * angle, short * timeHold)
{
  int noteValue = 0;
  int angleOriginal = 0;
  int angleNew = 0;
  bool songEnd = false;

  for (int i = 0; i < MAX_NOTES && songEnd == false; i++) // noteValue == 0 means end of song.
  {
    readIntPC(fin, noteValue);
    if (noteValue != 0)
    {
      angleNew = angle[noteValue];
      rotateArmAndPlay(angleNew, angleOriginal, noteValue, timeHold);
      angleOriginal = angleNew;
    }
    else
      songEnd = true;
  }
}
```

Figure 28: Play from file function



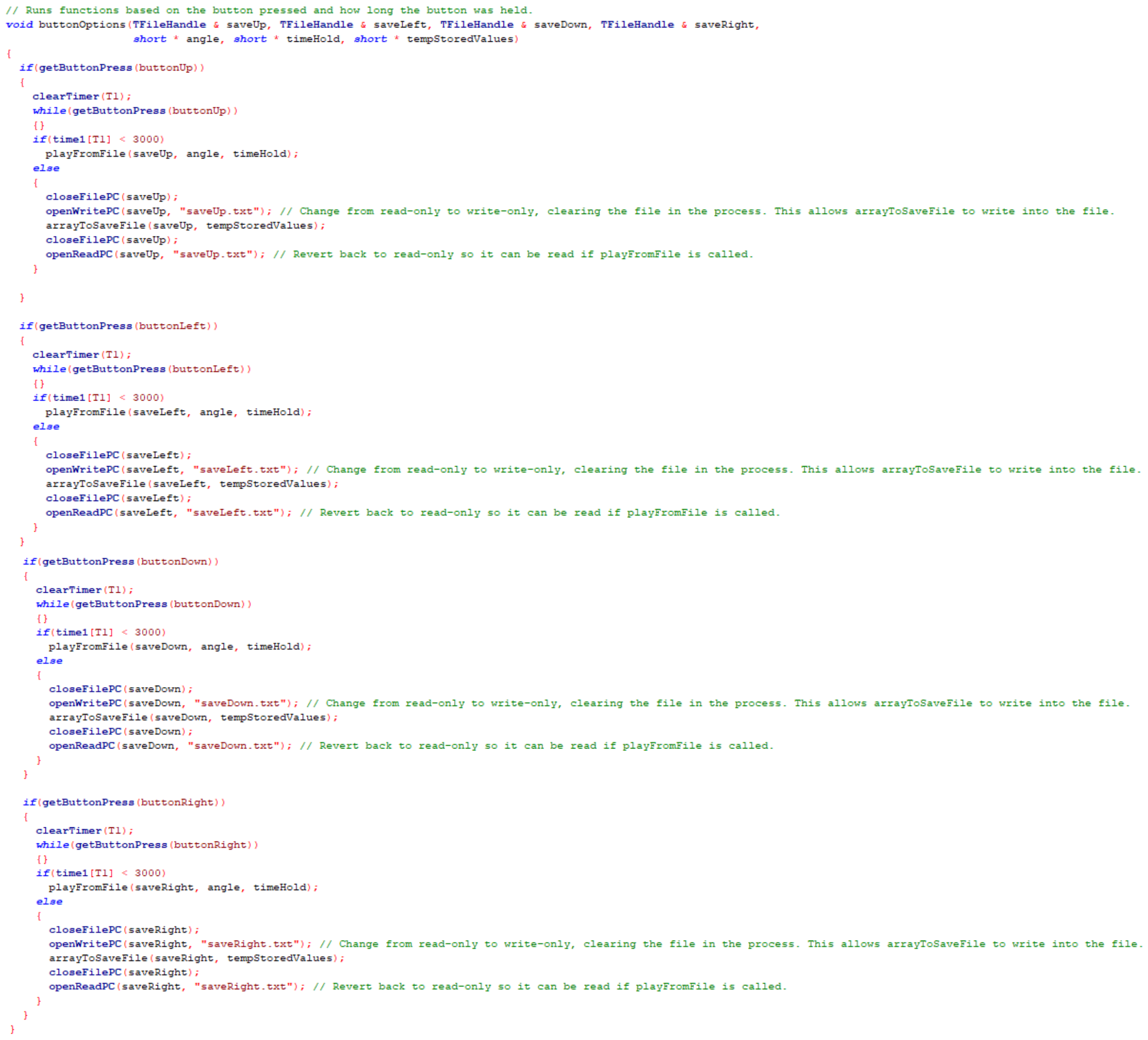
```
// Reads in note values from tempStoredValues array, then calls the rotateArmAndPlay function to rotate the left or right arm, then play the note.
void playFromReader(short * tempStoredValues, short * angle, short * timeHold)
{
  int noteValue = 0;
  int angleOriginal = 0;
  int angleNew = 0;

  for(int i = 0; i < MAX_NOTES && tempStoredValues[i] != 0; i++) // tempStoredValues[i] == 0 means end of song.
  {
    noteValue = tempStoredValues[i];
    angleNew = angle[noteValue];
    rotateArmAndPlay(angleNew, angleOriginal, noteValue, timeHold);
    angleOriginal = angleNew;
  }
}
```

Figure 29: Play from reader function



```
// Transfer note values in tempStoredValues array to a file.
void arrayToSaveFile(TFileHandle & fin, short * tempStoredValues)
{
  // Loop through the array and write each element to the file.
  for (int i = 0; i < MAX_NOTES && tempStoredValues[i] != 0; i++) // tempStoredValues[i] == 0 means end of song.
  {
    writeLongPC(fin, tempStoredValues[i]);
    writeCharPC(fin, ' ');
  }
  writeLongPC(fin, 0);
}
```

Figure 30: Array to save file function



```
// Runs functions based on the button pressed and how long the button was held.
void buttonOptions(TFileHandle & saveUp, TFileHandle & saveLeft, TFileHandle & saveDown, TFileHandle & saveRight,
                   short * angle, short * timeHold, short * tempStoredValues)
{
  if(getButtonPress(buttonUp))
  {
    clearTimer(T1);
    while(getButtonPress(buttonUp))
    {}
    if(time1[T1] < 3000)
      playFromFile(saveUp, angle, timeHold);
    else
    {
      closeFilePC(saveUp);
      openWritePC(saveUp, "saveUp.txt"); // Change from read-only to write-only, clearing the file in the process. This allows arrayToSaveFile to write into the file.
      arrayToSaveFile(saveUp, tempStoredValues);
      closeFilePC(saveUp);
      openReadPC(saveUp, "saveUp.txt"); // Revert back to read-only so it can be read if playFromFile is called.
    }

  }

  if(getButtonPress(buttonLeft))
  {
    clearTimer(T1);
    while(getButtonPress(buttonLeft))
    {}
    if(time1[T1] < 3000)
      playFromFile(saveLeft, angle, timeHold);
    else
    {
      closeFilePC(saveLeft);
      openWritePC(saveLeft, "saveLeft.txt"); // Change from read-only to write-only, clearing the file in the process. This allows arrayToSaveFile to write into the file.
      arrayToSaveFile(saveLeft, tempStoredValues);
      closeFilePC(saveLeft);
      openReadPC(saveLeft, "saveLeft.txt"); // Revert back to read-only so it can be read if playFromFile is called.
    }
  }
   if(getButtonPress(buttonDown))
   {
     clearTimer(T1);
     while(getButtonPress(buttonDown))
     {}
     if(time1[T1] < 3000)
       playFromFile(saveDown, angle, timeHold);
     else
     {
       closeFilePC(saveDown);
       openWritePC(saveDown, "saveDown.txt"); // Change from read-only to write-only, clearing the file in the process. This allows arrayToSaveFile to write into the file.
       arrayToSaveFile(saveDown, tempStoredValues);
       closeFilePC(saveDown);
       openReadPC(saveDown, "saveDown.txt"); // Revert back to read-only so it can be read if playFromFile is called.
     }
   }

   if(getButtonPress(buttonRight))
   {
     clearTimer(T1);
     while(getButtonPress(buttonRight))
     {}
     if(time1[T1] < 3000)
       playFromFile(saveRight, angle, timeHold);
     else
     {
       closeFilePC(saveRight);
       openWritePC(saveRight, "saveRight.txt"); // Change from read-only to write-only, clearing the file in the process. This allows arrayToSaveFile to write into the file.
       arrayToSaveFile(saveRight, tempStoredValues);
       closeFilePC(saveRight);
       openReadPC(saveRight, "saveRight.txt"); // Revert back to read-only so it can be read if playFromFile is called.
     }
   }
 }
```

Figure 31: Button options function

```
// Based on button input, calls readAndStore function, playFromReader function, or buttonOptions function. Return false --> continue program. Return true --> shutdown.
bool playOrSave(TFileHandle & saveUp, TFileHandle & saveLeft, TFileHandle & saveDown, TFileHandle & saveRight,
                short * angle, short * timeHold, short * tempStoredValues, short * colour1, short * colour2)
{
  while(!getButtonPress(buttonAny))
  {}
  if(getButtonPress(buttonEnter))
  {
    clearTimer(T1);
    while(getButtonPress(buttonEnter))
    {}
    if(time1[T1] < 3000)
    {
      // Activate suck roller. Read and store note values.
      MSMMotor(mmotor_S1_1, 20); // Turns on motor (multiplexer).
      readAndStore(tempStoredValues, colour1, colour2);
      MSMotorStop(mmotor_S1_1); // Turns off motor (multiplexer).

      // Play the stored notes.
      playFromReader(tempStoredValues, angle, timeHold);
    }
    else
    {
      displayBigTextLine(1, "If you see me");
      displayBigTextLine(3, "and you tryna");
      displayBigTextLine(5, "say what's up.");
      wait1Msec(1000);
      displayBigTextLine(7,"SKEE-YEE");
      wait1Msec(1000);
      displayBigTextLine(9,"EEEEEEEEEEEE");
      wait1Msec(5000);
      eraseDisplay();
      return true;
    }
  }
  else
    buttonOptions(saveUp, saveLeft, saveDown, saveRight, angle, timeHold, tempStoredValues);

  return false;
}
```

Figure 32: Play or save function

```
// Recalibrates left arm. Right arm is to be manually calibrated because there are not more ports for another touch sensor.
void recalibrate()
{
  // Lift up finger.
  motor[motorB] = -50;
  wait1Msec(1000);
  motor[motorB] = 0;

  // Recalibrate left arm.
  motor[motorA] = 10;
  while(SensorValue[S2] == 0)
  {}
  nMotorEncoder[motorA] = 0;
  motor[motorA] = -10;
  while(abs(nMotorEncoder[motorA]) < 36)
  {}
  motor[motorA] = 0;
  nMotorEncoder[motorA] = 0;
}
```

Figure 33: Recalibrate function

```
// Displays menu text on controller screen.
void displayMenu()
{
  displayString(1,"Hold up/down/left/right ");
  displayString(2, "buttons to save the song.");
  displayString(4, "Press up/down/left/right ");
  displayString(5, "buttons to play saved music.");
  displayString(7,"Press enter button to play");
  displayString(8, "music from sheet music.");
  displayString(10,"To exit, hold the enter button.");
  wait1Msec(8000);
  eraseDisplay();
}
```

Figure 34: Display menu function

```
task main()
{
  // Open files and check that they open correctly.
  TFileHandle saveUp;
  TFileHandle saveLeft;
  TFileHandle saveDown;
  TFileHandle saveRight;
  bool saveUpOkay = openReadPC(saveUp, "saveUp.txt");
  bool saveLeftOkay = openReadPC(saveLeft, "saveLeft.txt");
  bool saveDownOkay = openReadPC(saveDown, "saveDown.txt");
  bool saveRightOkay = openReadPC(saveRight, "saveRight.txt");
  if(!saveUpOkay || !saveLeftOkay || !saveDownOkay || !saveRightOkay)
  {
    displayString(4, "Error opening save files.");
    wait1Msec(5000);
  }

  // If files open properly, run the program
  else
  {
    short colour1[COMBOS] = {1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4,
                             5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 7, 7, 7, 7, 7, 7, 7};
    short colour2[COMBOS] = {1, 2, 3, 4, 5, 6, 7, 1, 2, 3, 4, 5, 6, 7, 1, 2, 3, 4, 5, 6, 7, 1, 2, 3, 4, 5, 6, 7,
                             1, 2, 3, 4, 5, 6, 7, 1, 2, 3, 4, 5, 6, 7, 1, 2, 3, 4, 5, 6, 7};
    short tempStoredValues[MAX_NOTES] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                         0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    short angle[COMBOS] = {0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 13, 13, 13, 22, 22, 22, 29, 29, 29, 37, 37, 37,
                           0, 0, 0, 7, 7, 7, 12, 12, 12, 15, 15, 15, 22, 22, 22, 29, 29, 29, 36, 36, 36, 43, 43, 43, 50, 50, 50};
    short timeHold[COMBOS] = {0, 500, 1000, 2000, 500, 1000, 2000, 500, 1000, 2000, 500, 1000, 2000, 500, 1000, 2000, 500, 1000, 2000,
                              500, 1000, 2000, 500, 1000, 2000, 500, 1000, 2000, 500, 1000, 2000, 500, 1000, 2000, 500, 1000, 2000,
                              500, 1000, 2000, 500, 1000, 2000, 500, 1000, 2000, 500, 1000, 2000}; // Left arm handles first 6 notes, right arm handles last 9 notes.

    // Wait for buttonEnter to be pressed.
    while(!getButtonPress(buttonEnter))
    {}
    while(getButtonPress(buttonEnter))
    {}

    configureAllSensors();
    recalibrate();
    displayMenu();

    while(playOrSave(saveUp, saveLeft, saveDown, saveRight, angle, timeHold, tempStoredValues, colour1, colour2) == false)
    {
      recalibrate();
      displayMenu();
    }
  }
  closeFilePC(saveUp);
  closeFilePC(saveLeft);
  closeFilePC(saveDown);
  closeFilePC(saveRight);
}
```

Figure 35: Task main function

Appendix B: CAD Models

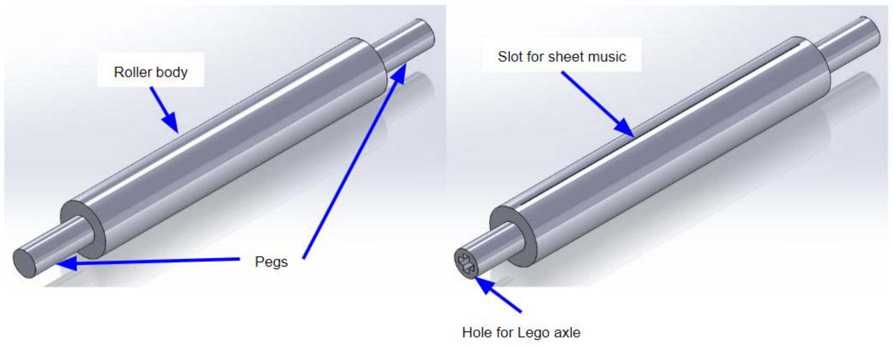


Figure 36: 3D CAD model of tensioning roller (left) and suck roller (right)

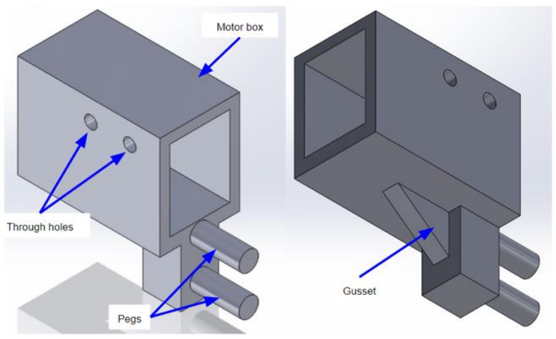


Figure 37: 3D CAD model of motor mount

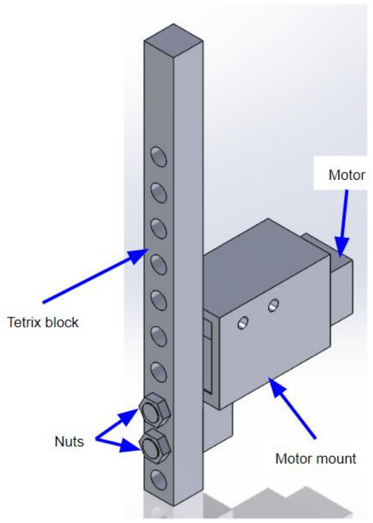


Figure 38: SolidWorks assembly with motor mount, motor, Tetrix block, and nuts

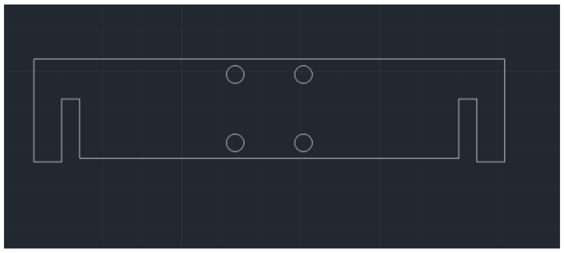


Figure 39: AutoCAD drawing of color sensor bracket

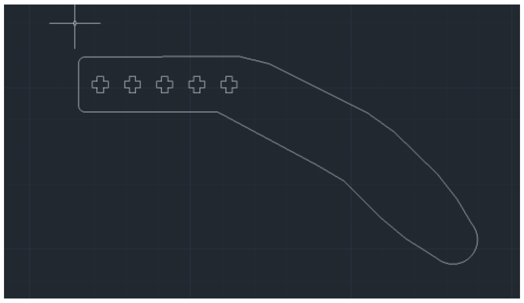


Figure 40: AutoCAD drawing of finger

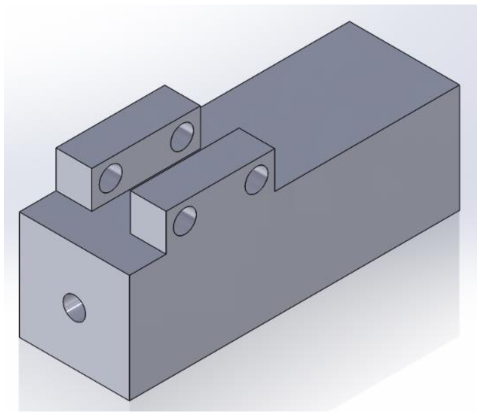


Figure 41: SolidWorks model of a medium Lego motor

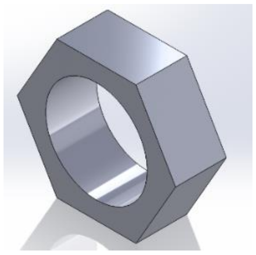


Figure 42: SolidWorks model of a nut

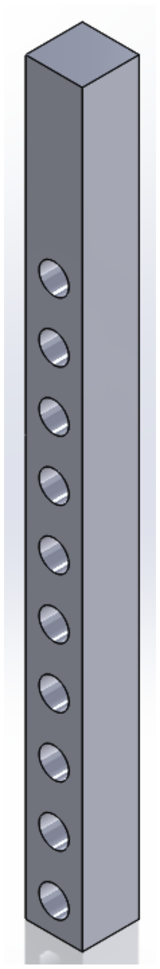
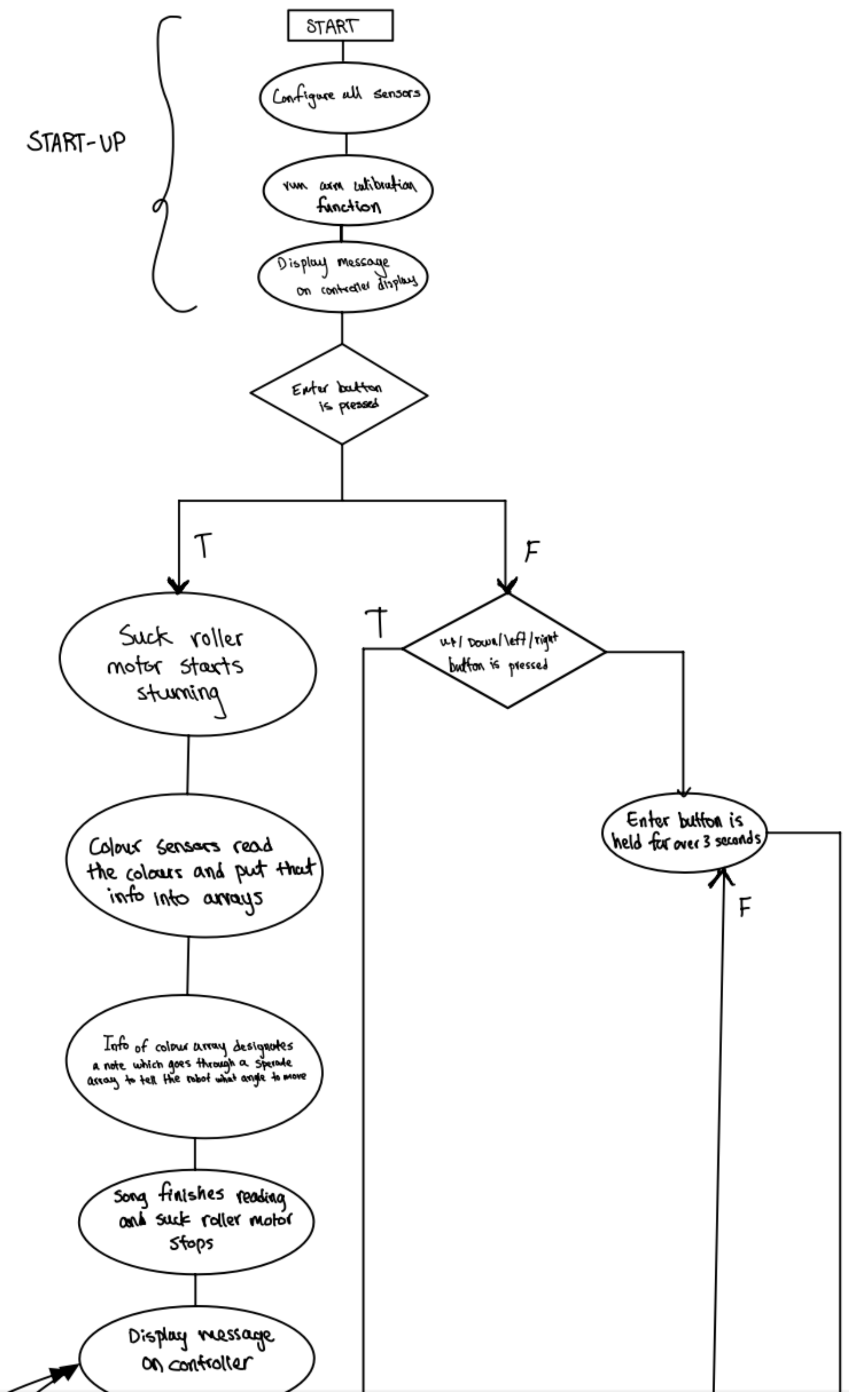


Figure 43: SolidWorks model of a Tetrix block

Appendix C: Software Flow Chart


START
START-UP
Configure all sensors
run arm calibration function
Display message on controller display
Enter button is pressed
T
F
Suck roller motor starts sturning
T
up/Down/left/right button is pressed
Colour sensors read the colours and put that info into arrays
Enter button is held for over 3 seconds
F
Info of colour array designates a note which goes through a Sperade array to tell the robot what angle to move
Song finishes reading and suck roller motor stops
Display message on controller

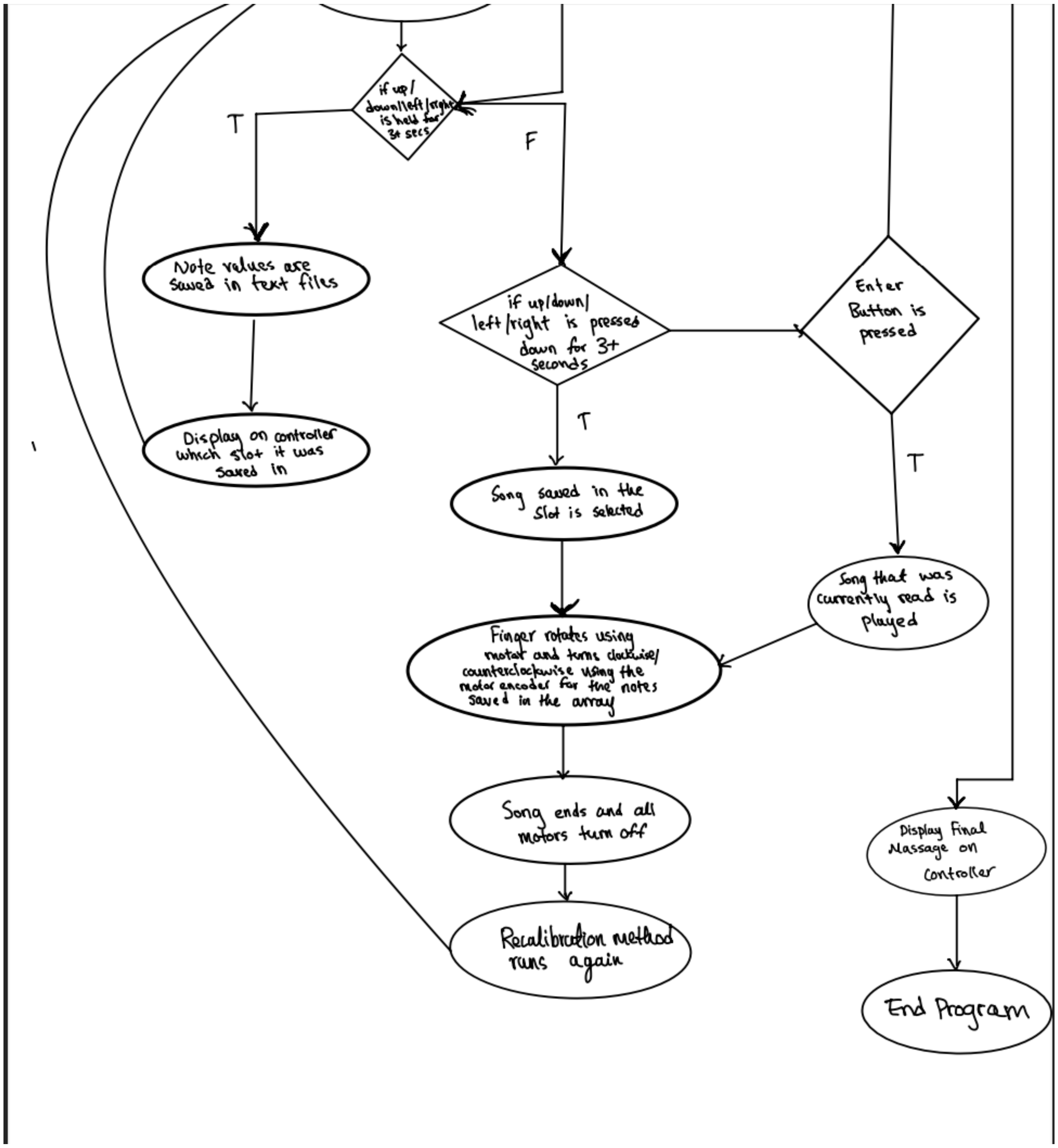

if up/down/left/right is held for 3+ secs
T
F
Note values are saved in text files
Display on controller which slot it was saved in
if up/down/left/right is pressed down for 3+ seconds
T
Enter Button is pressed
T
Song saved in the slot is selected
Song that was currently read is played
Finger rotates using motor and turns clockwise/counterclockwise using the motor encoder for the notes saved in the array
Song ends and all motors turn off
Recalibration method runs again
Display Final Massage on Controller
End Program

Figure 44: Software flow chart

Appendix D: Robot Pictures

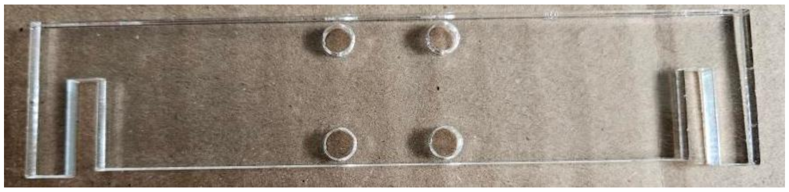


Figure 45: Color sensor bracket

Figure 46: Electronic keyboard

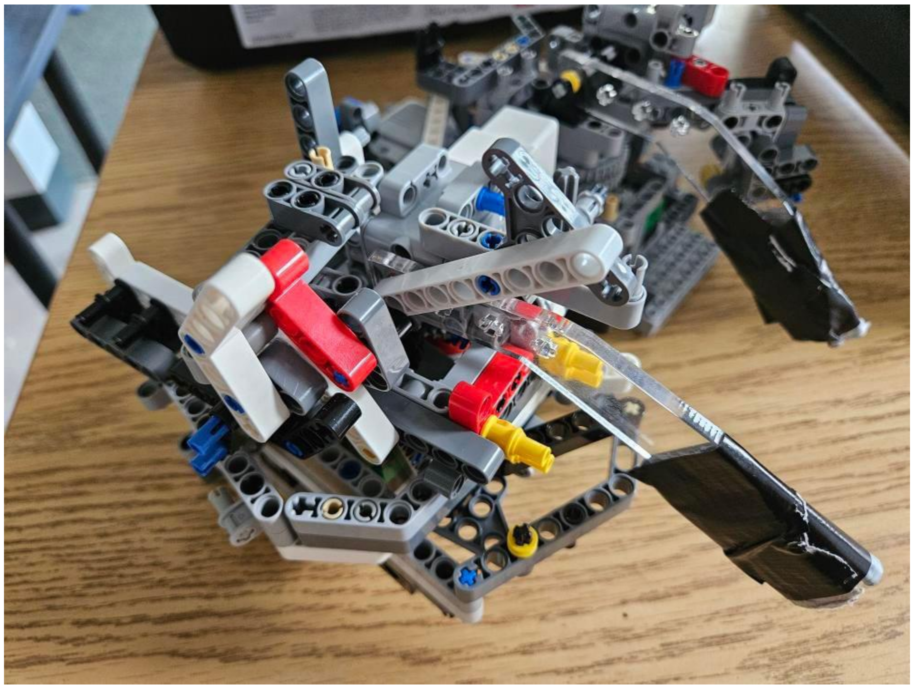


Figure 47: Note-playing section of robot

Appendix E: Extra Sketches

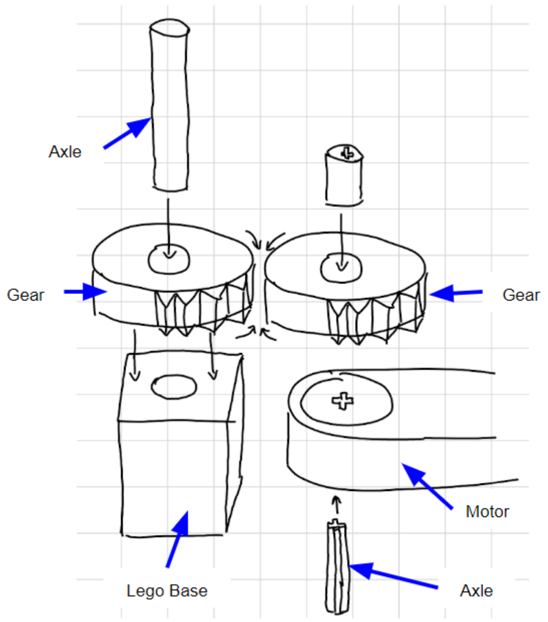
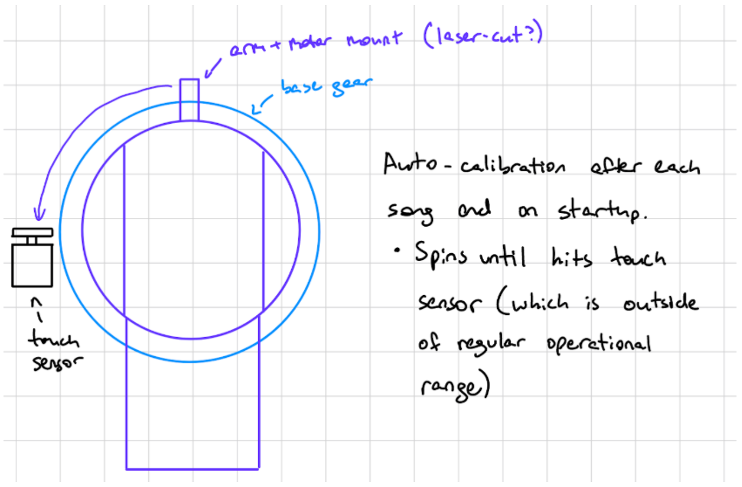


Figure 48: A more detailed design of the horizontal rotation mechanism



Auto-calibration after each song and on startup.

- Spins until hits touch sensor (which is outside of regular operational range)

Figure 49: A conceptual sketch of the recalibration system

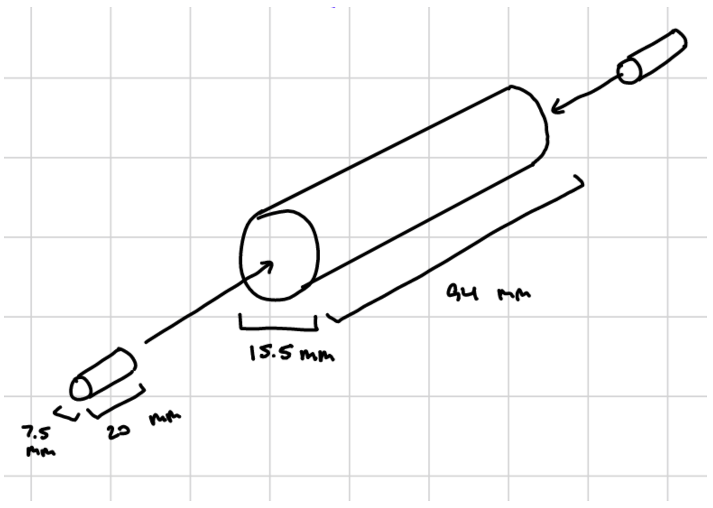


Figure 50: Dimensions for the rollers

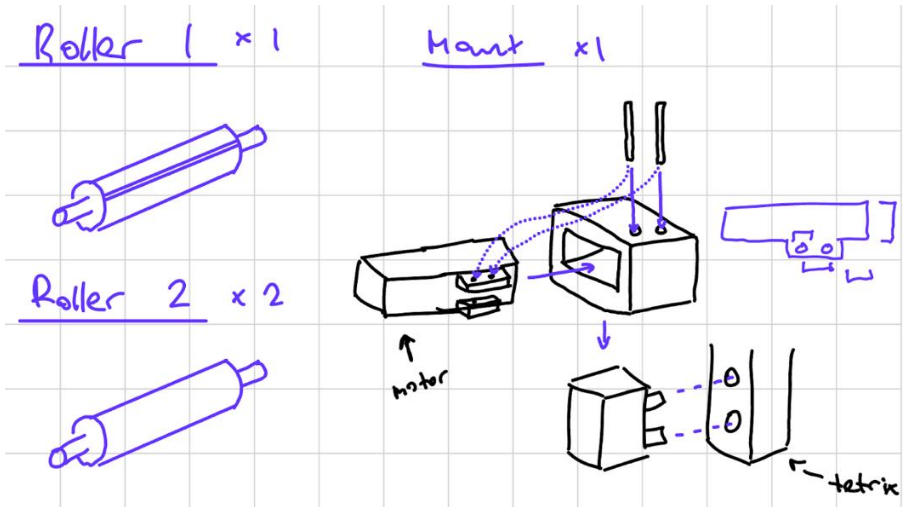


Figure 51: Sketches of the suck roller (roller 1), tensioning roller (roller 2), and motor mount

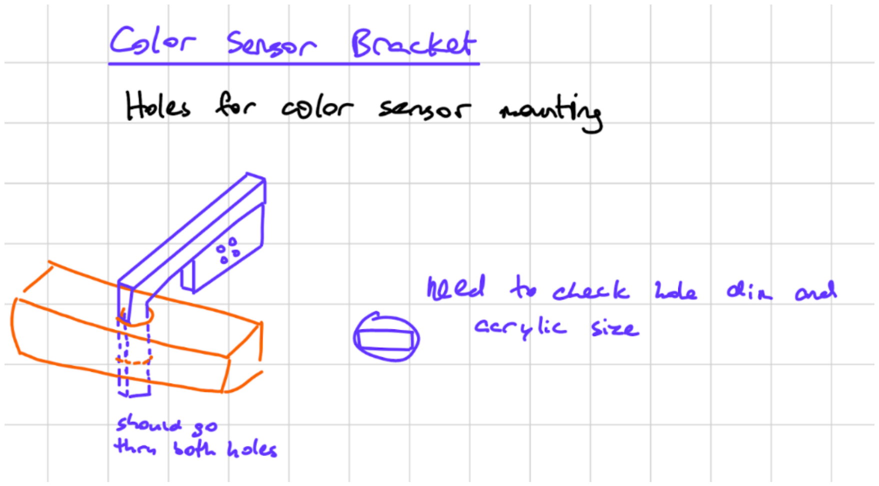

Color Sensor Bracket
Holes for color sensor mounting
need to check hole dia and acrylic size
should go thru both holes

Figure 52: Color sensor bracket sketch

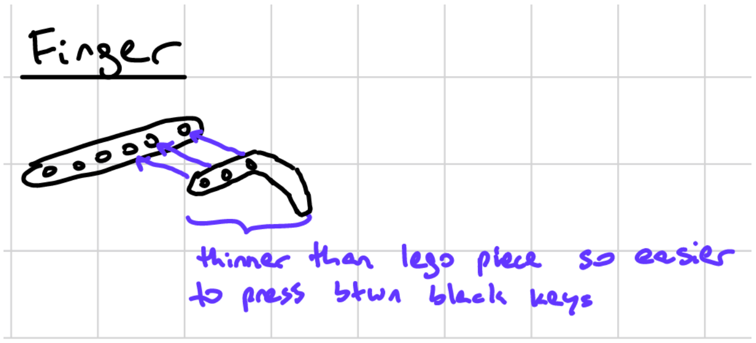


Figure 53: Finger sketch