ShotBot™ – MTE100/GENE121 Final Project

Final Design Report

Group 445

Balen Seeton – 20832255

Neil Fernandes – 20854840

Ansh Sahny - 20829104

Abhimanyu Dev Singh - 20833032

December 3rd, 2019

Table of Contents

Table of Figures

Table of Tables

Introduction

Problem

Basketball players at any age and any skill level struggle with shooting. Even NBA players work every offseason to make the smallest improvements in their overall shooting percentages, with emphasis placed on free throw shooting and three-point shooting. ShotBot™ is a robotic shooting partner to make shooting practice fun again, and in turn, improve the shooting percentage of the user.

Shooting practices are repetitive and boring, and as a result, the shooter experiences a large decrease in concentration over time. Athletes are widely known to be competitive, as this drive to be better than the competition is vital to success in their field. Our robot is programmed to closely imitate the shooters shooting percentage to provide a fast-paced and engaging competition. The saying goes that "Practice makes Perfect", and our robotic shooting partner will make shooting practice fun again. While practicing against the ShotBot™, the athlete will compete harder than if he/she was shooting alone, thus helping the player improve his or her shooting percentage.

The National Basketball Association rakes in the 3rd most revenue among major sports across the world, mainly due to its high scoring games and incredibly fast pace. Raising the overall shooting percentage across any basketball league would increase the score, thus attracting more people to watch, thus increasing the revenue of the league [1].

Stakeholders

The main stakeholders for this product include the basketball teams and the players. Others affected are the basketball leagues, team staff, and any shareholders of a basketball league, including the fans. Anyone who enjoys playing basketball, whether in pick-up games or simply shooting around at a park, could also be a stakeholder [1].

Scope

Main Functionality of the ShotBot™

The ShotBot™ successfully moves in the X and Y directions with the help of the LEGO EV3 motors and the Steel Rods and moves from one designated point in a region to another. Moreover, the shooting mechanism on the ShotBot™ works flawlessly with the help of the Rack and Pinion method. Unfortunately, the robot failed to move at a certain horizontal angle to face the basketball hoop in the right way. The vertical angle too never worked in the way it was designed to move in.

Sensors

The robot measured the overall distance for the X and Y directional movement with the help of the Ultrasonic Sensors attached at the Back and right side of the robot (see Figure 1).

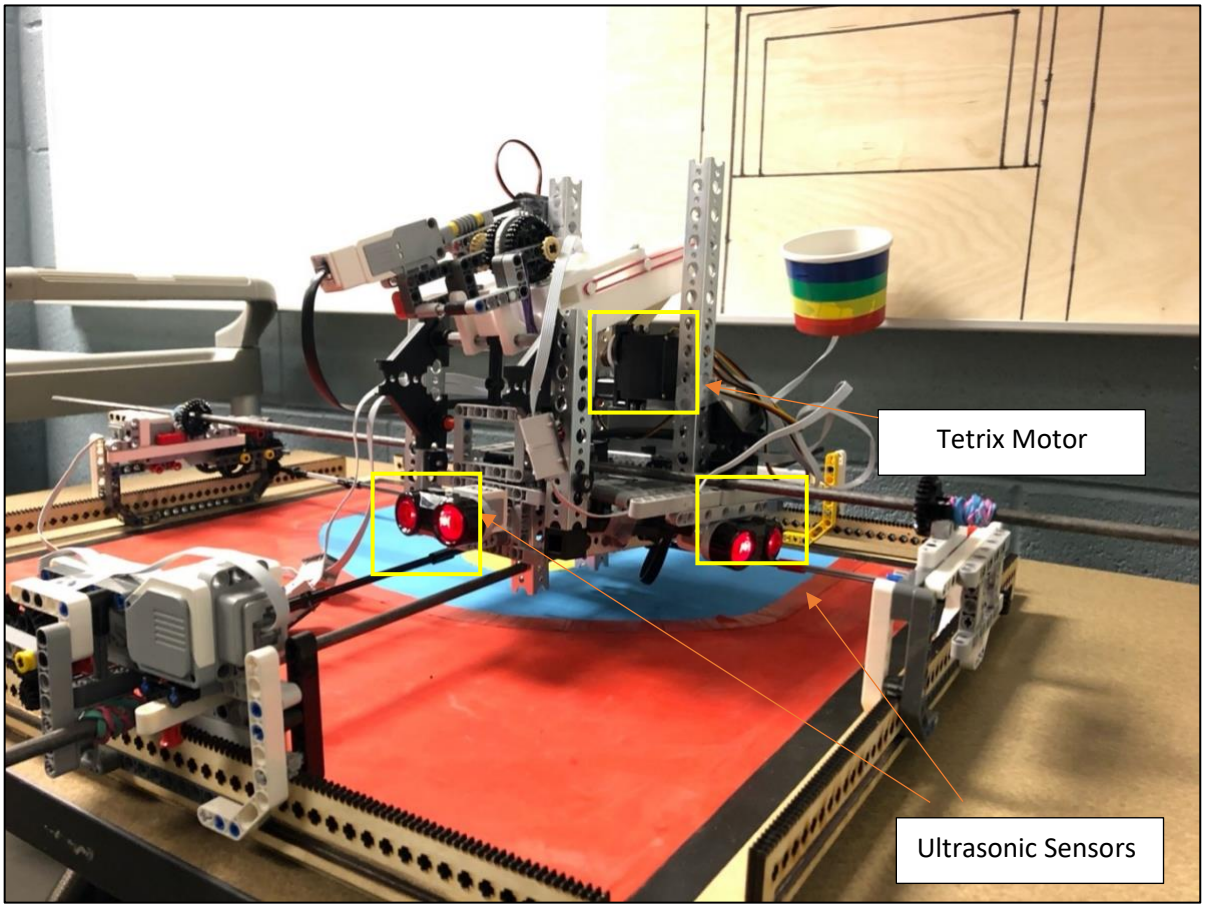

Figure 1: ShotBot™ with the sensors

The robot also used the touch sensor to help the robot keep record of the points the robot had just scored. In addition to these sensors, the robot also used the motor encoders present in the Tetrix motors to help adjust the shooting angle and to adjust the robot's horizontal angle to make the robot face

the basketball hoop. The buttons on the LEGO EV3 Brick also act as sensors as they take the users' actions as input.

Interaction to Environment

The ShotBot™ interacts with the environment by using the help of sensors such as the Ultrasonic sensor and touch sensor. The Ultrasonic sensor helps the robot to keep track of its position and where it has to move using the moveToPos() function. The touch sensor helps as it plays a key role in checking whether the robot has successfully scored a point from a specific region (see Figure 2), also with the help of the decisionMaking() function.

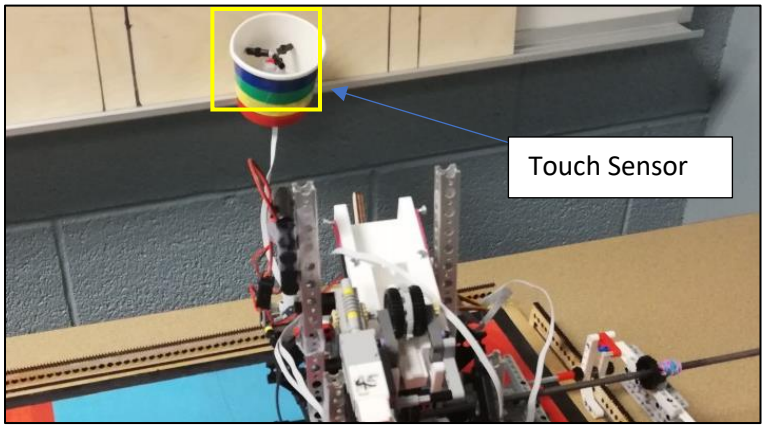

Figure 2: Touch sensor under hoop

Other than the sensors listed, the robot also interacts with the user's actions via the control buttons on the LEGO EV3 Brick. Using the brick, the user can input the desired time limit (time inputted has to be less than 30 minutes) at which the robot will run for. The user can also stop the program by pressing and releasing the center button, but only after the robot has reached the time limit. The Tetrix Prime motors are used for changing the vertical and horizontal angles of the robot by using the calcHorAngle and calcVertAngle functions respectively, to help shoot the ball at a greater range of distance, and in the right direction.

Unfortunately, the calcHorAngle function was functioning as intended and kept pointing the robot in the wrong direction. The robot also does not have a "safety switch" which the user can use in case of an unexpected event occurs.

Task Completion

The robot has recognized that the tasks are all done when the time limit that the user had input during the robot start-up procedure, has passed. After the robot's senses this, the ShotBot™ displays statistics of how well the robot had shot the balls and how many points it had scored over the specific period. The overall shutdown happens when the user presses the center button on the LEGO EV3 module.

Changes in Scope

There have been a lot of changes and revisions since the completion of the initial preliminary robot report. They are:

Mechanical Changes

i. Removal of Color Sensor:

The color sensor which was going to be used for sensing which zone the robot would be in was removed to accommodate for the Tetrix Motors by freeing an extra sensor port.

ii. Removal of Caterpillar Tracks

The caterpillar track design had posed a great problem as it was found that due to the weight of the robot, the tracks had sagged, tilting the robot to a great extent which would have caused the shot accuracy of the robot to decrease at a staggering rate.

iii. Usage of Cold Steel rods

The team used steel rods instead of wooden beams to help the robot move in the X and Y directions (see Figure 3). The idea behind the usage of the beams was because these steel rods were sturdier than the wooden beams and were proven to help the robot slide in the X and Y directions.

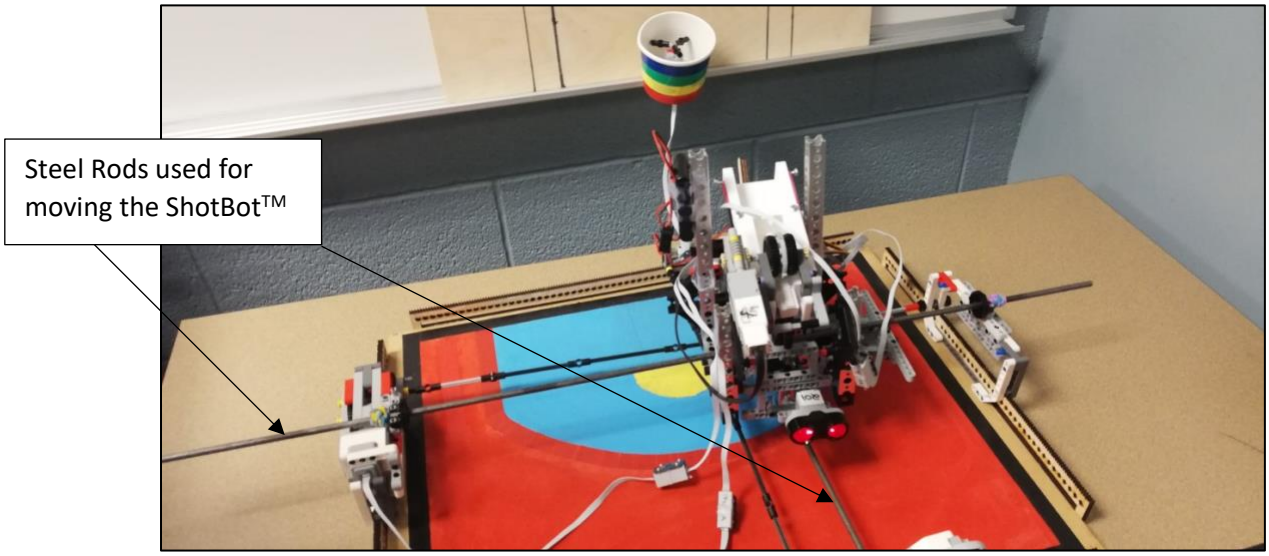

Figure 3: The Steel Rods on the ShotBot™

iv. Placement of Backboard

The backboard was not attached to the playfield to provide some distance so that the robot can shoot the ball inside the basket.

v. Removal of the Ball Collection System

The idea for a Ball collection system was scrapped since the system was resource-intensive (as it needed 2-3 motors) and the lack of time that the team had.

Software Changes

There were a few software changes, mainly additional functions added to the software. They are:

i. calcHorAngle()

The calcHorAngle was introduced so that the robot can calculate the horizontal angle so that robot can face the basket.

ii. calcVertAngle()

This function was introduced so that the robot can calculate the angle of reach so that the ball could go inside the basket

iii. roundingAlg()

This function was introduced so that the robot can round whole numbers to make the robot's movement easier.

iv. makeDecision()

This function was added to act as the brains of the ShotBot™ where all the movement and shooting functions created by the team were used.

v. moveToPos()

This function was introduced to the software to help the ShotBot™ move across the playfield.

Mechanical Design and Implementation

Design Debrief

The fundamental design of the ShotBot was based on the workings of a 2D plotter. This allowed the system to complete all the desired tasks. The rack and pinion were used to control the motion on the x and y axes which gave the system flexibility to move to any coordinate across the playing field. The shooter is located at the intersection of both axes and is designed to align itself in the direction of the basket, adjust itself to the desired launch angle, and shoot the ball. The shooter consisted of a metal beam mount containing all the mechanical and hardware components alongside the shooting mechanism. The integration of all these smaller design ideas used to complete the overall system within the time constraint was the biggest challenge as it involved assembling the designs which was manufactured using a variety of tools and parts like 3D printer, laser cutter, Tetrix parts, and LEGO pieces.

Play Field

Final Design

The playing field was created to provide a base for the physical parts of the system. It also helped to depict a scaled-down model of an official NBA basketball court. It was made from Medium Density Fiber (MDF) board with the length to width ratio staying true to the dimensions of a regulation basketball court. The dimensions were specifically 60cm x 48cm. The board was divided into three zones, designated by three different colors, each assigned with a different number of points scored. These zones were divided using colored electrical tape and acrylic paint to help the robot understand which zone it has to shoot from and how many points they have scored.

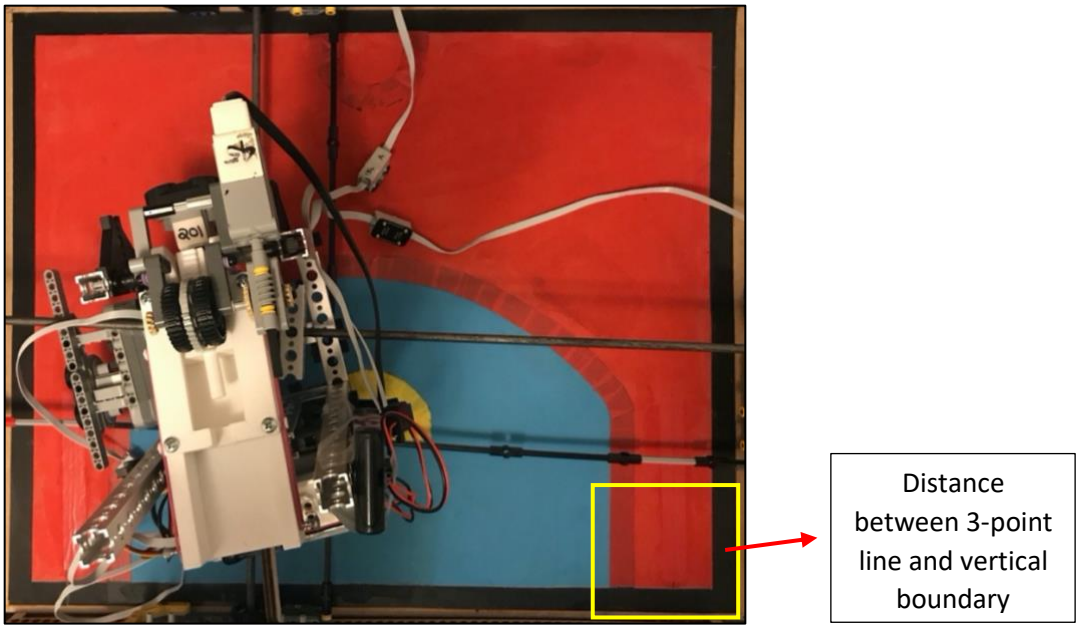

Figure 4: Play field top view

Design Decisions

The original design, as stated in the preliminary design report, declared that we will be making each zone of the playfield around the same proportions resembling an official NBA court. However, while making the actual setup, the proportions of the zones were changed to make it possible for our shooter to fit and make a shot from every possible position. Thereby, the distance of the three-point line that forms the vertical boundary was kept as 6cm instead of 3cm, as per the original plan. (see **Error! Reference source not found.**)

Driving Rails

Final Design

The driving rails of the ShotBot are an essential part as these were supposed to take the load of the robot and accurately position it on the x and y coordinate system. The rack and pinion mechanism is located on all four sides of the board. It was first designed on AutoCAD, as per the dimensions corresponding to the LEGO technic 40 tooth gear. The design of the rack had a slit in between, as seen in Figure 5, to provide space for the pinion assembly to move smoothly over the rack. The pinion assembly consisted of a LEGO motor and was constructed completely using LEGO pieces. The motor was used to drive the pinion gear allowing the assembly to move over the rack gear smoothly. Finally, to connect both sides of the same axis, a steel rod of 1/2in x 36in was used. Alongside the rod, the pinion on the driving end was connected to a pinion on the other side using an axle to transfer the torque and connect the assembly.

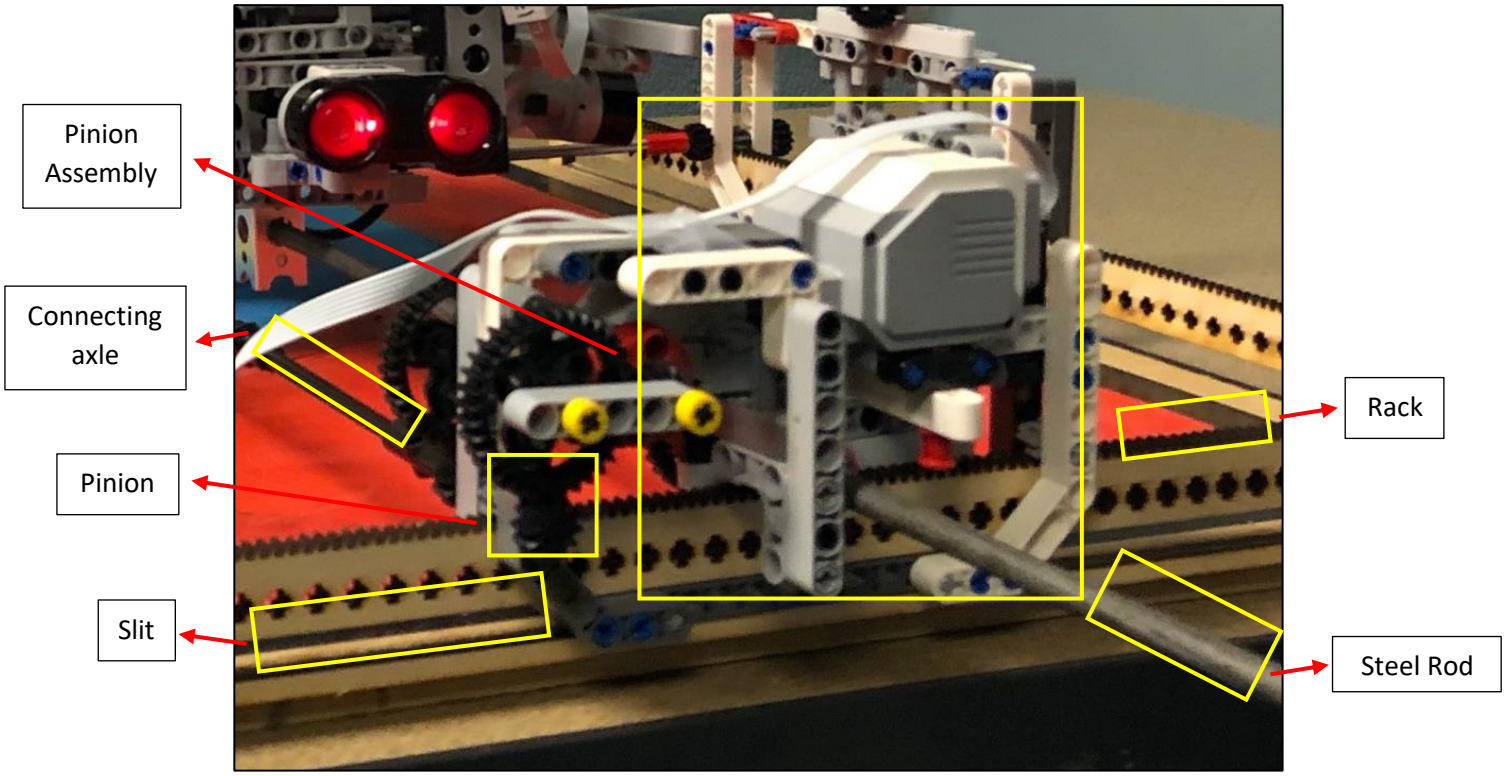

Figure 5: Rack and Pinion assembly

Design Decisions

To ensure that the task of the rails and the criteria of the system were successfully fulfilled, critical design decisions and changes were made involving the driving rails. To begin with, the rack and pinion were used instead of other moving mechanisms such as conveyer belt, lead screw and nut, and linear actuator. That is because the rack and pinion can provide the most adjustable motion and highest accuracy at the same time. Other mechanisms were either to slow and highly accurate, like the lead screw and nut, or were not accurate at all, such as a conveyer belt.

Moreover, the rack had the dimensions of LEGO for the convenience of attaching it to the LEGO assembly of the pinion gear. The slit between the rack gear was also dimensioned as per the LEGO beam to make the assembly simpler.

Finally, the powered and the non-powered assemblies, located on both the ends of the field, had to be connected to make the system move simultaneously on both ends. Steel rods were chosen as steel is more durable and the cylindrical shape provides low friction for the shooter to slide on when compared to other materials. However, connecting the two systems using only the rod was not enough as the torque could not be transferred efficiently between the powered and non-powered assemblies due to the horizontal lag created as the mounts of the rod had some loose space. To fix this issue, the power of the driving pinion was transferred to the other pinion directly, using an axle. This allowed both the pinions to rotate at the same velocity, hence moving the entire assembly simultaneously.

Shooting Mount

Final Design

The main purpose of the shooting mount was to mount the shooting mechanism, the EV3 brick, ultrasonic sensors, and servos to adjust the horizontal and vertical angle of the shooting mechanism. The frame of the mount was constructed from Tetrix beams and rivets were used to mount LEGO pieces on it as all the components mounted on it had the assembly holes of the dimensions of either Tetrix or LEGO. The mount and all the components can be seen in Figure 6.

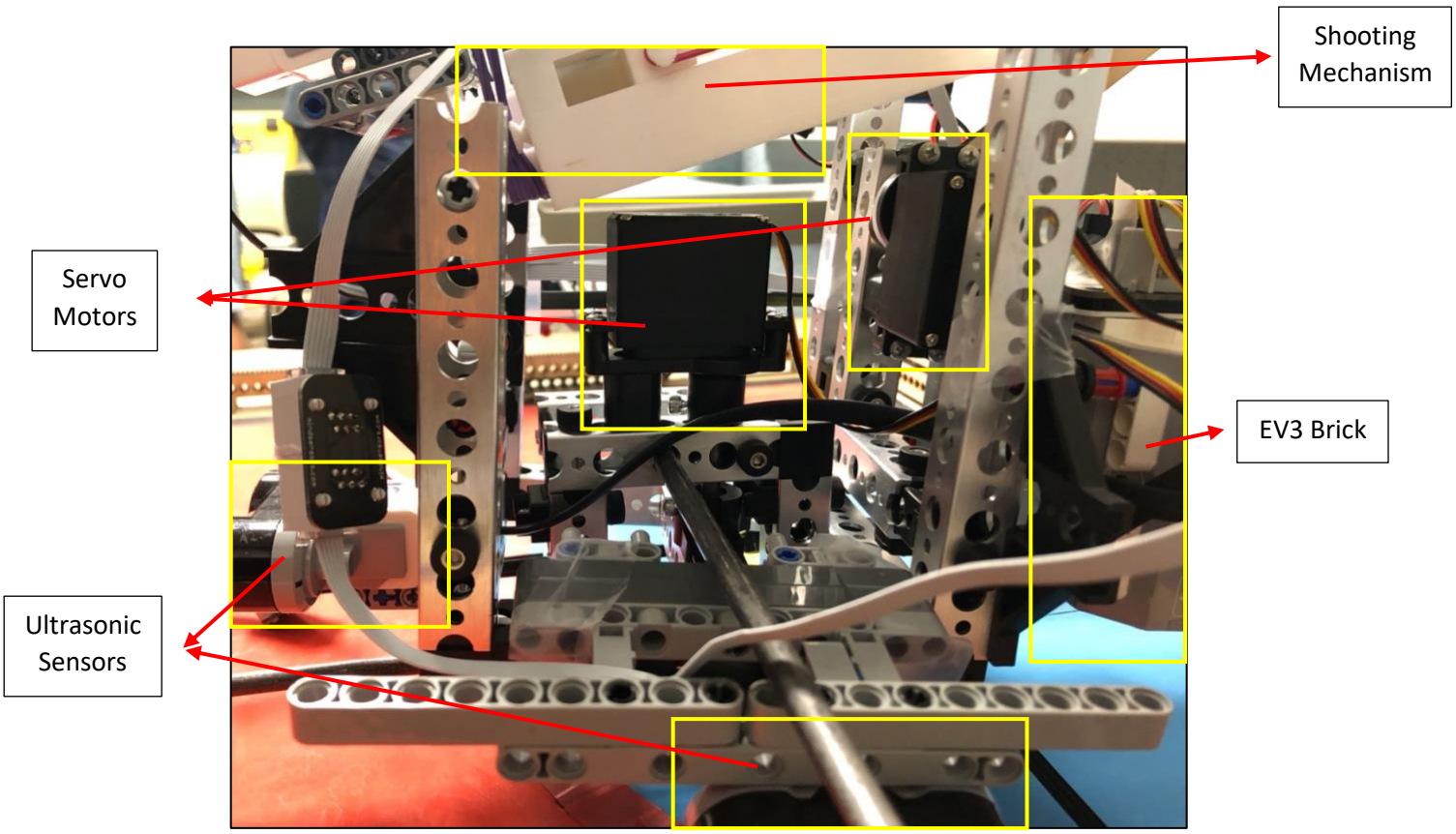

Figure 6: Shooting mount assembly

Design Decision

No changes were made from the preliminary design report to the design of the mount as all the components that were supposed to be mounted were present. The main reason for using the Tetrix beams as the base of the mount was to ensure that the base can hold the load. Another reason is that the holes on the Tetrix beams were big enough for the steel rods, due to which any extra parts were not needed to connect the rods to the mount. The EV3 brick was mounted on the base as most of the motors and sensors were mounted on the shooter mount. The motors, which were not on the mount, were equidistant from the shooter, making the wiring easier. Two standard servos were used because the horizontal and vertical angles were supposed to be adjusted between +90° and -90° which can accurately be done using those two standard servos.

Shooting Mechanism

Final Design

The shooting mechanism was the part used to complete the final function of the system, shooting the ball. It could store only one ball at a time, and it worked based on the slip gear mechanism developed beforehand. The slip gear was controlled through a medium motor that transferred power using a worm gear, as seen in Figure 7. The worm gear pulled the rack gear back which had an elastic force acting on it by the elastic on the side of the trigger, as seen in Figure 8. This trigger was then released as the gear came in contact with the missing teeth.

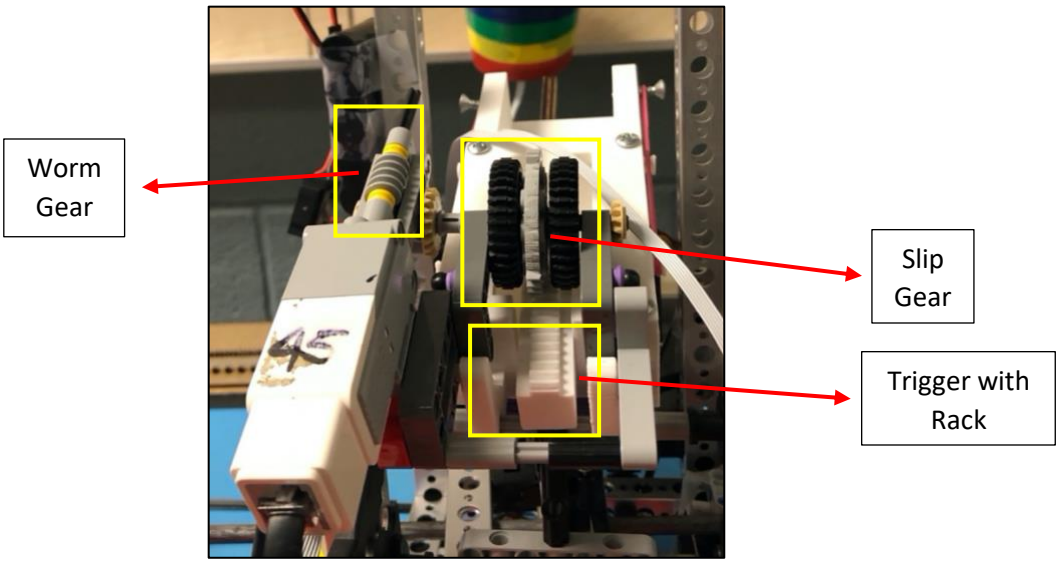

Figure 7: Shooting Mechanism top view

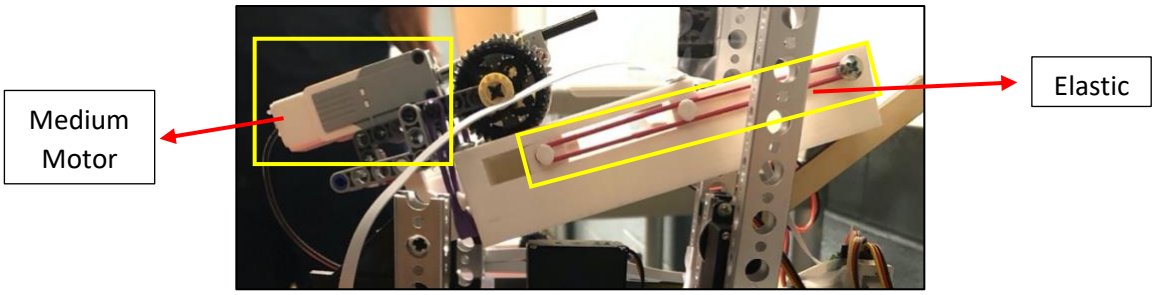

Figure 8: Shooting Mechanism side view

Design Decision

The decision of using a slip gear mechanism was because the slip gear mechanism is more consistent when compared to any other type of shooter. It can release the ball with the same power almost every time it is supposed to shoot. The major reason for using a worm gear is because the worm gear can counter the elastic force without slipping. This is because of the high gear ratio that allows us to pull the trigger each time without fail. As there was a high torque present, provided by the worm gear, we could use the medium motor without worrying about the load on it and also fit a smaller size motor using fewer parts. The 3D printed slip gear and the shooting mechanism helped to customize the power and design as per the needs of the model making it more efficient. One of the biggest changes that were made from the CAD model presented in the preliminary report to the final system was that the ball storage basket was missing from the 3D printed shooter. This reduced the functionality, but the system still completed the tasks it was supposed to do.

Trade-Offs

The biggest trade-off, due to the design decisions made throughout, was that the ball collection system could not be finished. Other minor changes made in the system did not affect the functionality which is the reason why we could meet most of the constraints and criteria.

Software Design and Implementation

High-Level Software Design

The software component of this robot was coded solely in ROBOTC. The program was broken down into 8 functions and a main program, and each group member programmed at least 1 of these functions. The functions were grouped together based on their purpose for the robot. The functions were divided into groups, including the math/physics related functions, the decision-making, and movement functions, and the file and user input functions.

Task List

The task list used for the demo is shown in Figure 9 below.

Group 445 Checklist - Final Project

Start-Up Tasks

- [ ] User inputs the time for which the design will run for
- [ ] Read in the coordinates and the statistics from the file
- [ ] Position the design at the starting point

Regular Operations

- [ ] Design moves in the x- and y-directions to it's position
- [ ] Design rotates to face the basket
- [ ] Shooting arm changes angle
- [ ] Shooting arm shoots the ball
- [ ] Gather data to output the statistcs

Unexpected Events

- [ ] Valid time needs to be entered (between 0 and 30 minutes)
- [ ] Software will not run if file input is invalid/does not work
- [ ] The ball needs to push the touch sensor fully down in order for the basket to count
- [ ] Pinion gears are aligned with the rack in the event of gears slipping

Shutdown Procedure

- [ ] Design will complete the demo after the time has passed
- [ ] Robot will return to the starting position
- [ ] Robot will display the statistics on the screen
- [ ] Press and release the center button to end the program

Criteria

- [ ] Points are in the range of +/- 20% of the score
- [ ] Robot shoots from a distance of 53.24 cm or less

Constraints

- [ ] System weighs less than 3 kilograms
- [ ] Robot runs for 30 minutes or less

Figure 9: A picture of the final task list used to determine the success of the design on demo day

Function Description

Table 1: Description of Functions

Function Name	Parameters & Return	How it Works	Author(s)
setTimer()	Parameters: None.	Pressing the buttons on the EV3 will increment the time until the	

	Returns: Time inputted by the user. (FLOAT)	user has inputted his/her desired length of time. The user will press the enter button to exit this process.	Abhimanyu Dev Singh
calcDistance()	Parameters: The distance to the x and y-direction ultrasonic sensors. (2x FLOATS) Returns: The distance from the shooting release point to the basket. (FLOAT)	Using, trigonometry, the function will calculate the distance to the basket.	Neil Fernandes Ansh Sahny
calcHorAngle()	Parameters: The distance to the x and y-direction ultrasonic sensors. (2x FLOATS) Returns: The horizontal shooting angle in degrees. (INT)	Using trigonometry, the function will calculate the necessary angle to point the shooter towards the basket.	Neil Fernandes Ansh Sahny
calcVertAngle()	Parameters: The distance to the x and y-direction ultrasonic sensors. (2x FLOATS) Returns: The vertical shooting angle in degrees. (FLOAT)	Using kinematics, the function will calculate the necessary angle to raise the shooting arm to shoot the ball the correct distance.	Neil Fernandes Ansh Sahny
shotMechanics()	Parameters: The distance to the x and y-direction ultrasonic sensors. (2x FLOATS) Returns: None.	Calls previous functions to set the angle of the shooting mechanism and releases the ball via a rack and pinion system.	Neil Fernandes Ansh Sahny Balen Seeton
roundingAlg()	Parameters: A positive number. (FLOAT) Returns:		Balen Seeton

	The integer closest to that number from 1 to 3. (INT)	Will round the decimal to the nearest whole number to simplify the robot's movement.	
makeDecision()	Parameters: The number of shots taken by the robot (INT), the robot's current points/shot and the desired points/shot (points/shot of the human) (2x FLOATS) Returns: A number (1, 2, or 3) representing the colour the robot should move to. (INT)	Using a basic averaging algorithm, this function serves as the robot's "intelligence". It will use its points/shot and the deviation from the desired points/shot to make the best decision to reach the end goal of matching desired points/shot.	Balen Seeton
moveToPos()	Parameters: A number (1, 2, or 3) representing the colour the robot should move to (INT), The distance to the x and y-direction ultrasonic sensors. (2x FLOATS) 2 other variables for the position it will move to are passed by reference in the parameter. (2x FLOATS) Returns: None.	A random operator will choose a point from a dataset in the system, and then move to that exact point on the board, constantly taking data from the ultrasonic sensors.	Balen Seeton
decisionMaking()	Parameters: None. Returns: None.	This function serves as the "main" of the program, so it serves many functions. 1. Open file and file Check 2. Initializing and populating the array of points from the file 3. Allows user to set the desired amount of time	Abhimanyu Dev Singh Balen Seeton

		4. Make Decision and Move to Desired Position (looped) 5. Display Statistics and wait for Button Press	

The flowcharts for each of these functions are shown below from figures 9-17. The flowchart for the main function is not there as our main function only comprises of the desicionMaking() function.

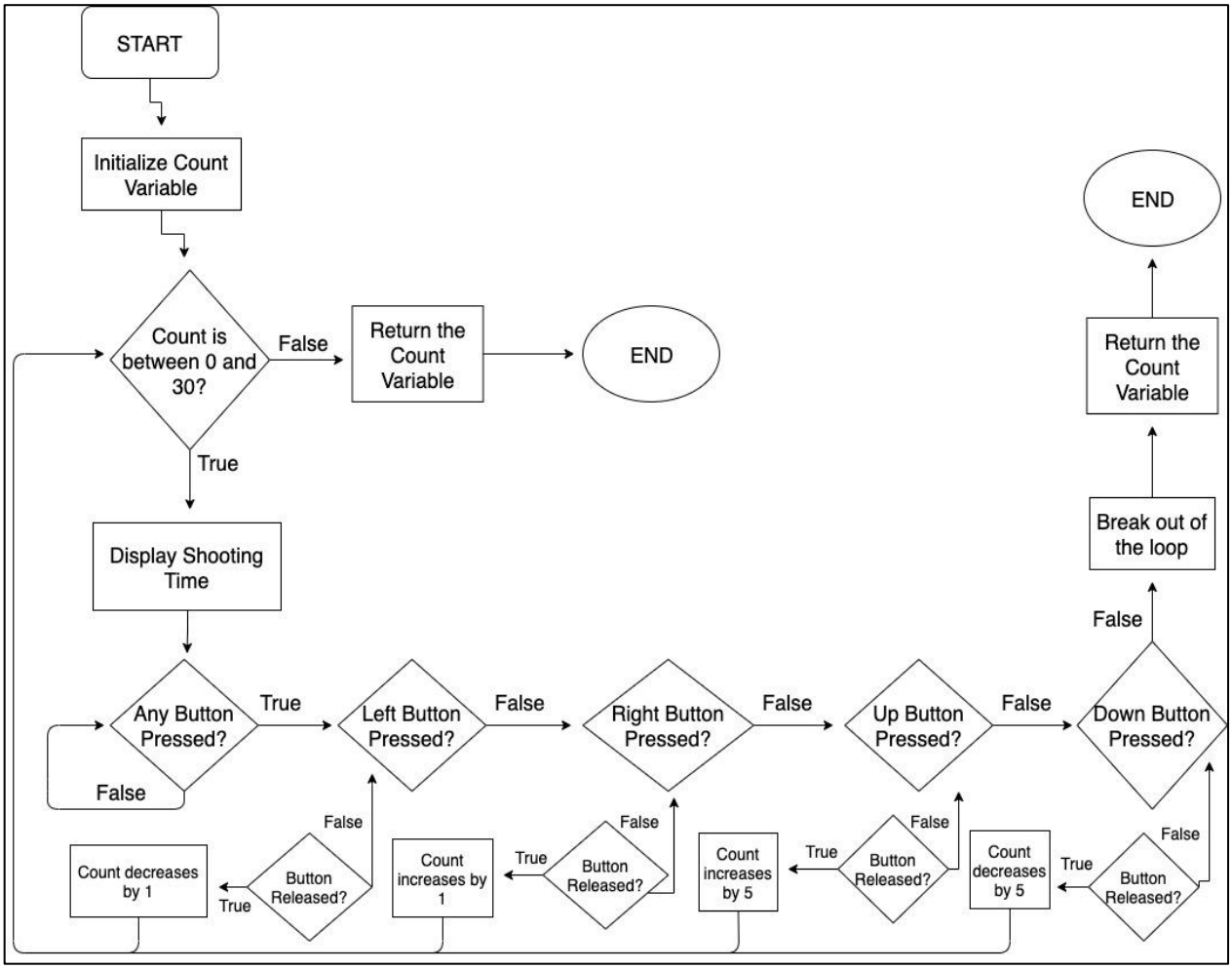

Figure 10: The flowchart for how the setTimer() function works

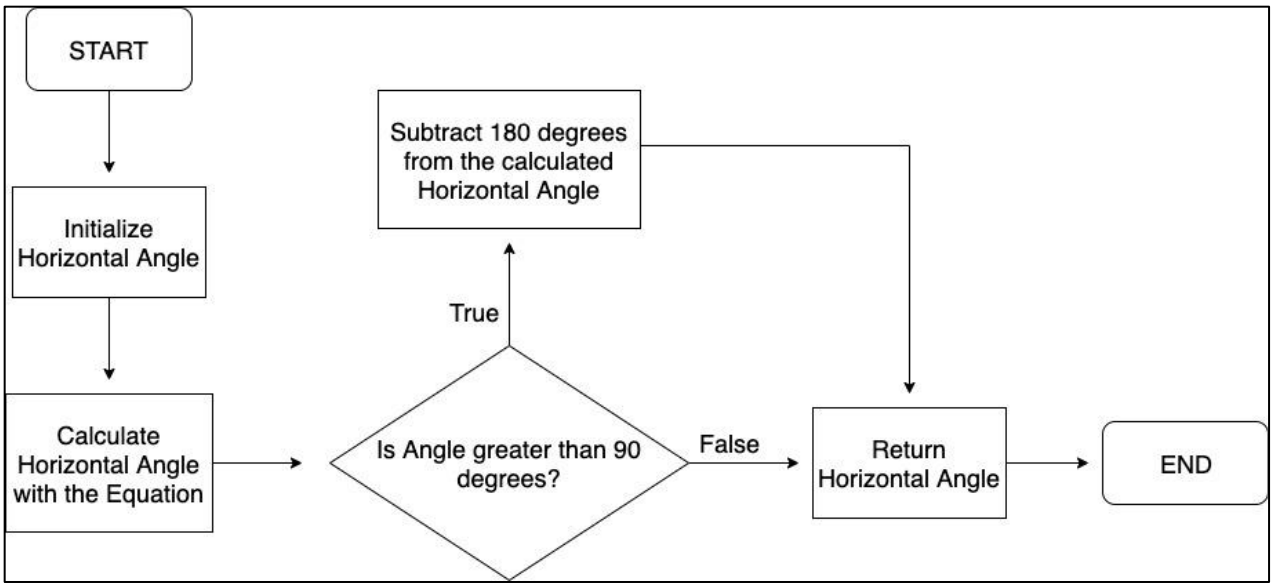

Figure 11: The flowchart for the calcHorAngle() function

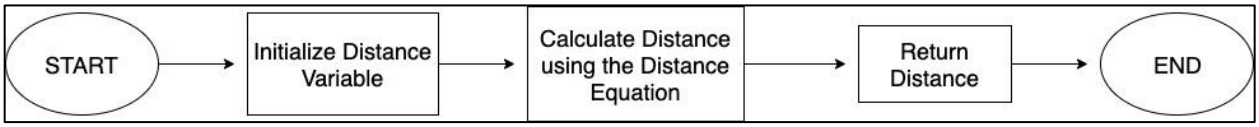

Figure 12: A flowchart for the calcDistance() function

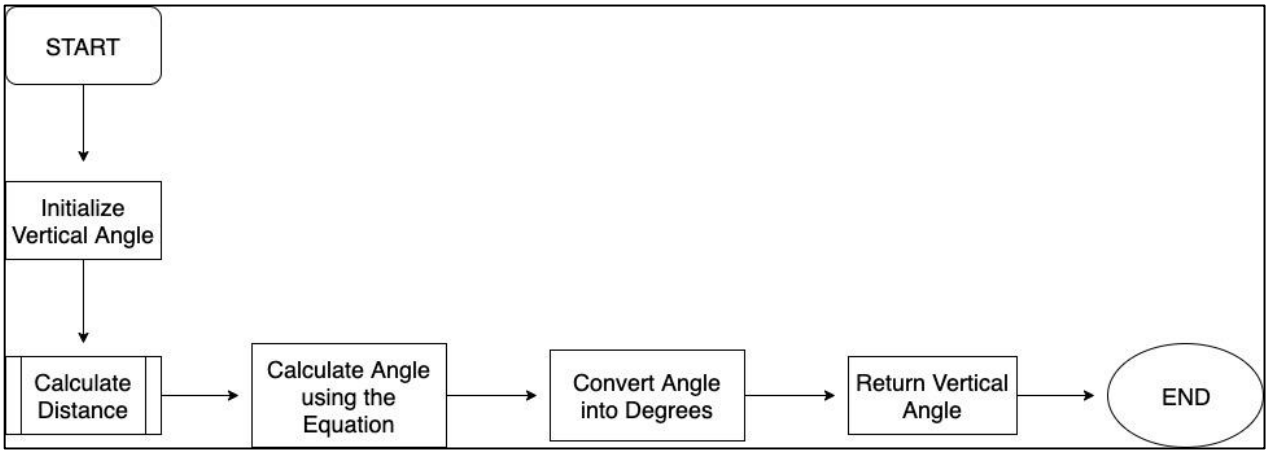

Figure 13: A flowchart for the calcVertAngle() function

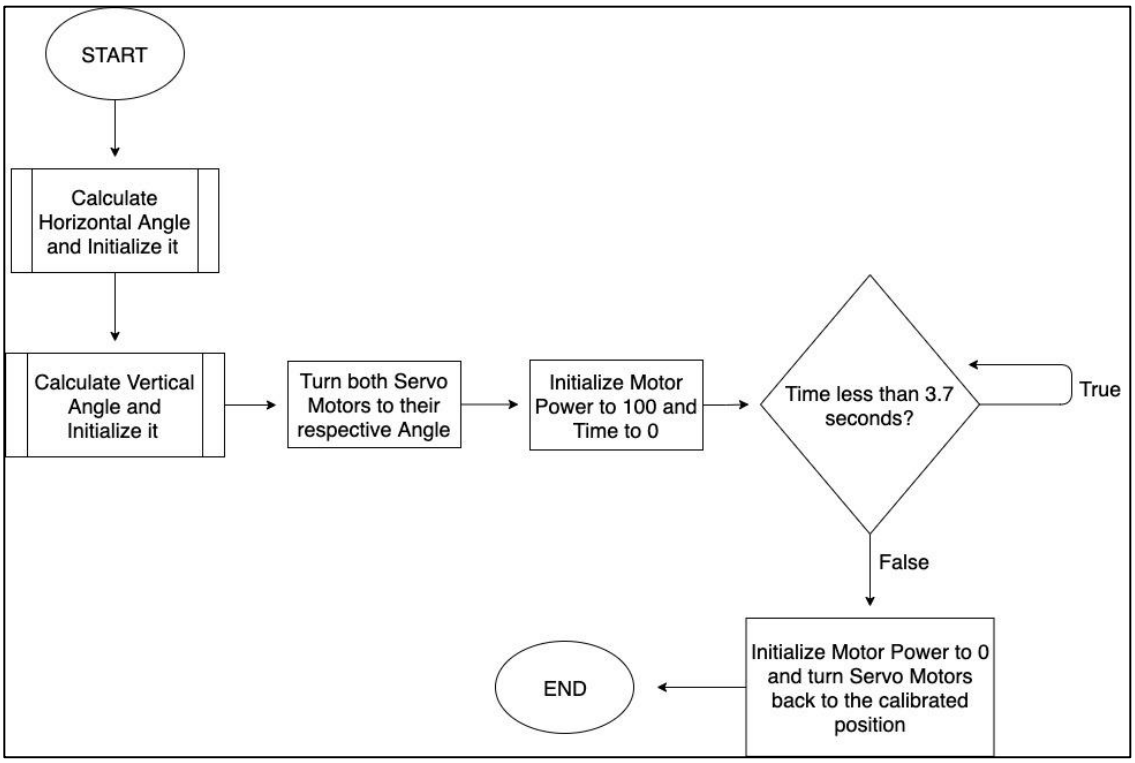

Figure 14: A flowchart of the shotMechanics() function and how it works

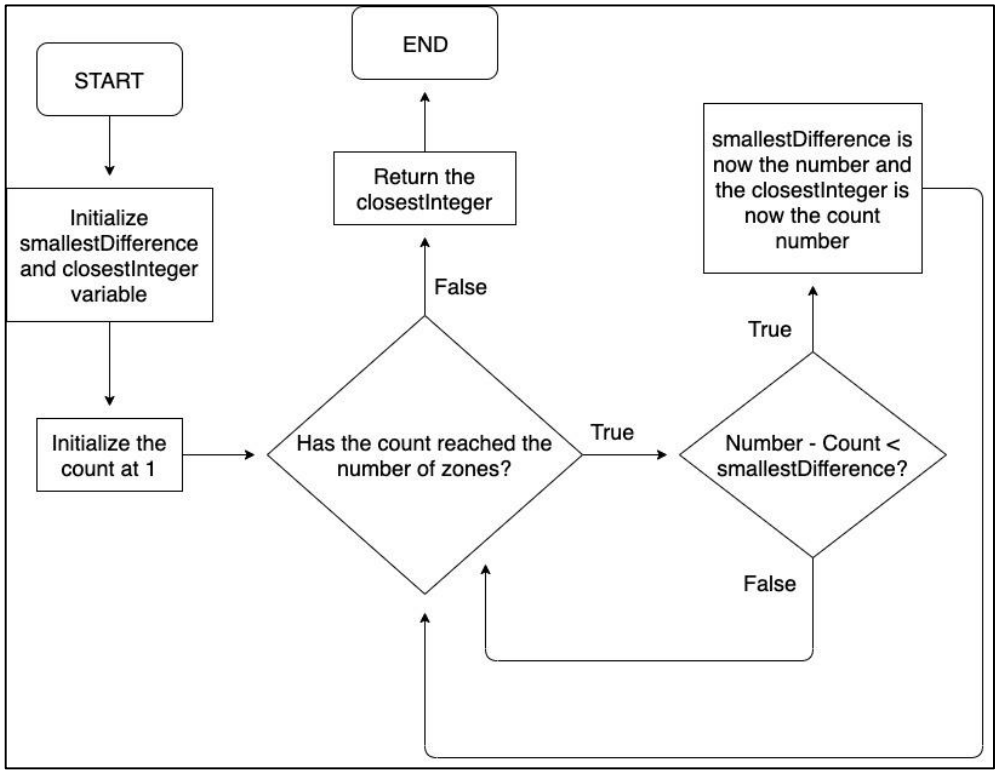

Figure 15: A flowchart of the roundingAlg() function and how it works

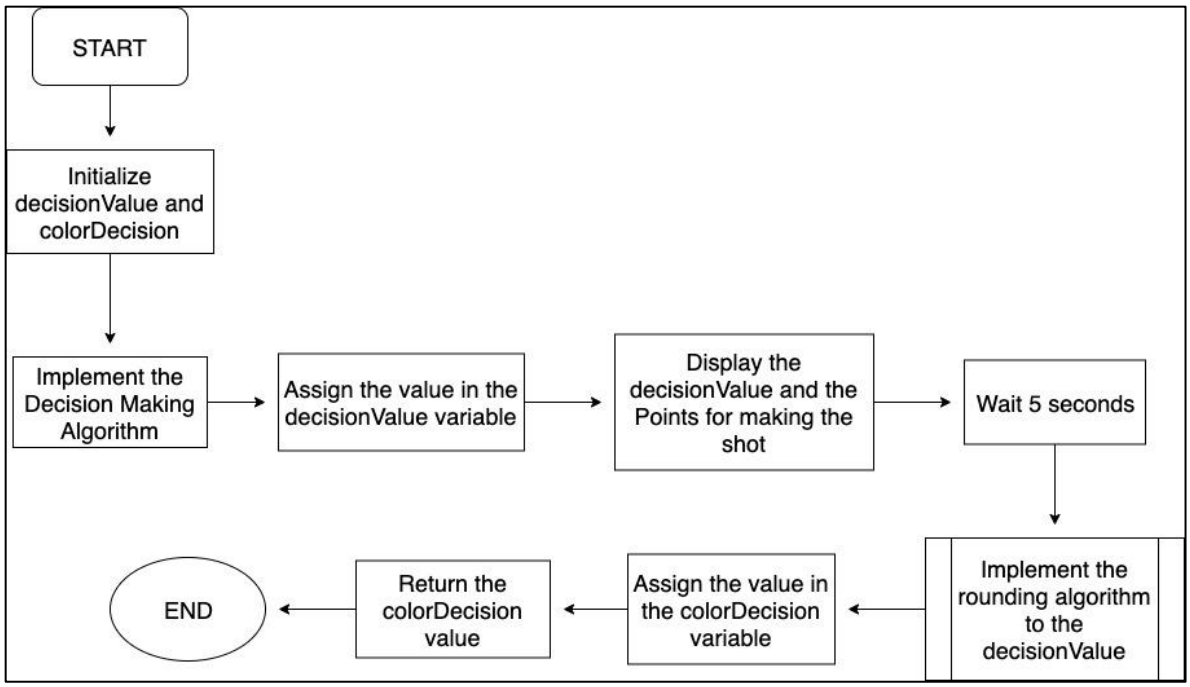

Figure 16: A flowchart of the makeDecision() function and how it works

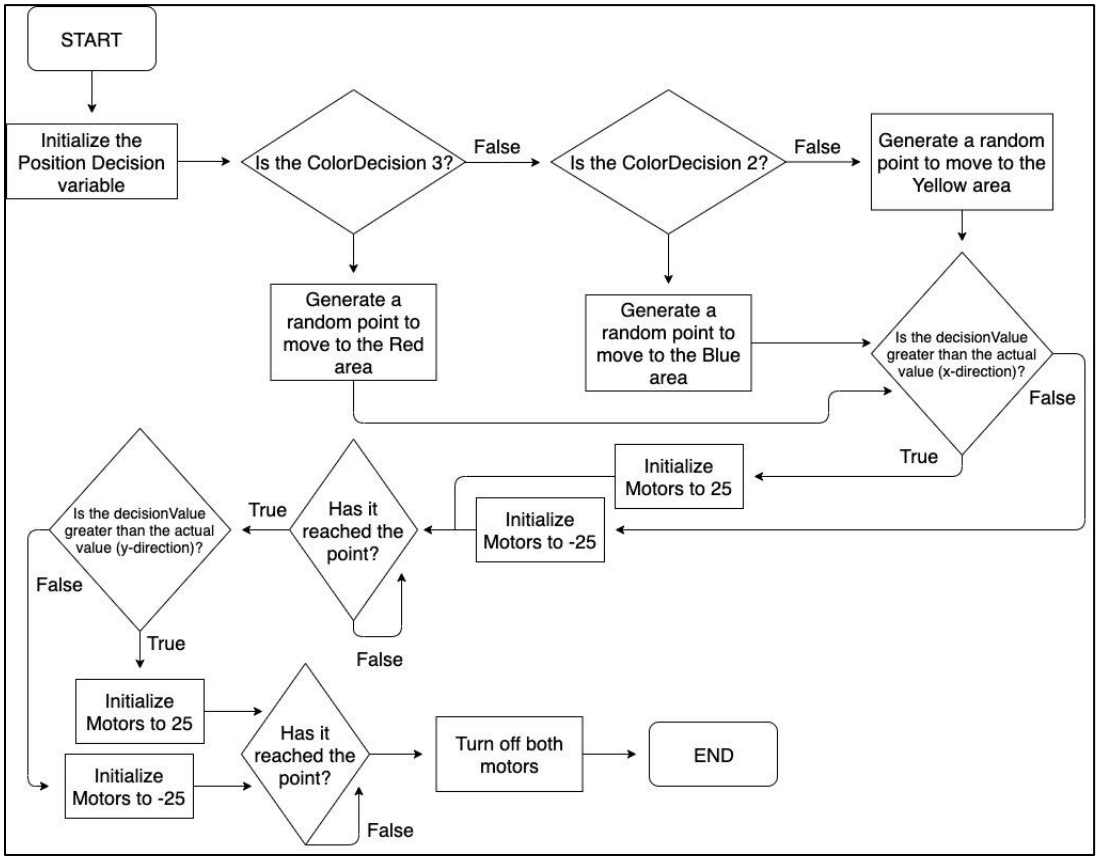

Figure 17: A flowchart of the moveToPos() function

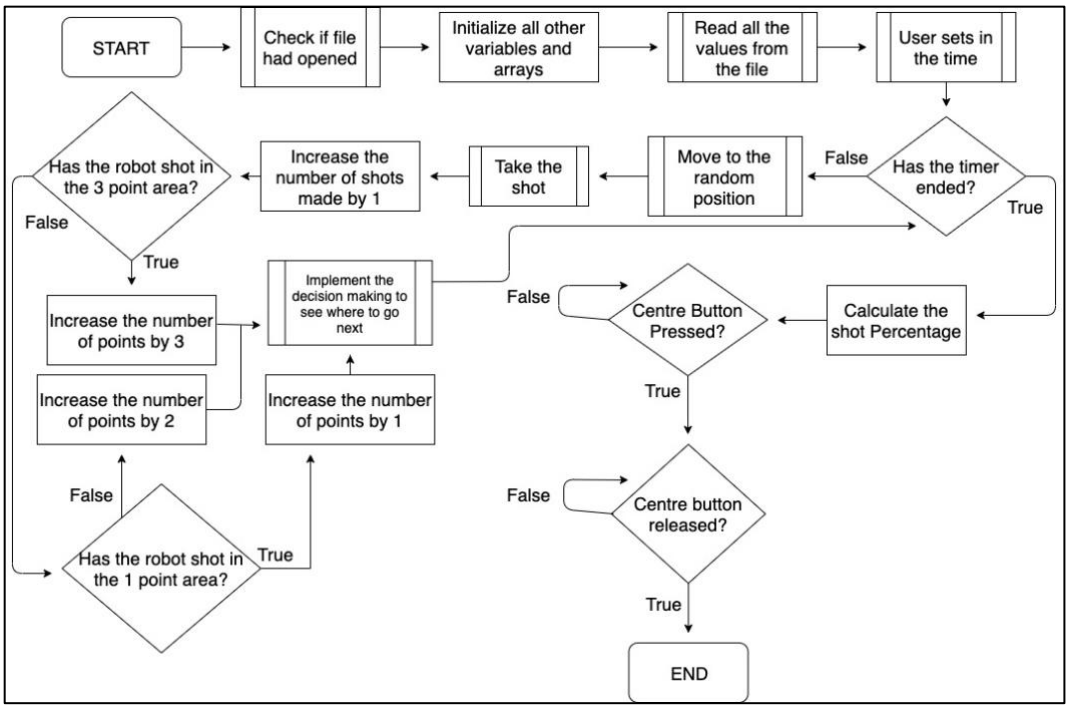

Figure 18: A flowchart of the decisionMaking() function and how it works

See Appendix A for all source code relating to these functions and the source code of main, as well as global constants and libraries used throughout the code.

Data Storage

The main data storage method used in the program was from a file input. The file contained information obtained from the shooter that the robot was trying to imitate. Once the data was read in, the program would calculate the points per shot required to best mirror the human.

While shooting, the robot would constantly update its points per shot and the total points and store them in variables. It would then use these values to keep them as close as possible to the human it was attempting to imitate.

Other data used to shoot such as the vertical and horizontal angles were only temporarily stored until the robot complete the shot mechanics. They would later be reset based on the next position.

Software Design Decisions

The largest software decision made was based on the limitation of the 4 sensor ports. Unexpectedly, the operation of the servo motors required one sensor port to be used as opposed to using a motor port. Unfortunately, this meant that a sacrifice was needed as we had proposed for 2 ultrasonic sensors, a touch sensor, and a color sensor. In the end, we decided to use ultrasonic sensor limitations to detect what zone the robot was in, as seen in Appendix A, sacrificing the color sensor.

Testing Procedure

Table 2: Software Tests Ran

Test Ran	Why?	Expected Behavior	How Behavior was Ensured
Setting Timer	To ensure that the buttons would result in the correct incrementation of the time.	Left: -1, Right: +1 Down: -5, Up: +5 Enter: EXIT LOOP	Testing each button individually, and testing boundaries (1 min and 30 mins). Tested enter button.
File Input -> Array	To ensure that the information from the file was read in successfully and inserted into a data array.	The time would be stored in a variable, the shots made/missed were stored in an array, and the point value of each shot was stored in another array.	Outputting the file values to the display on the robot in an easy to read format.
Movement	To ensure that the shooting mechanism was not too heavy for the rails bordering the court. It also tested the direction of the motors.	The robot would move around the court with ease in the x and y directions.	Tested all directions of the motors. Tested moving motors at the same time. Tested moving to an exact position.
Decision Making	To ensure that the robot would move to the correct zone based on its previous shots and data from the file.	The robot would move to a point in the correct zone and stop. Based on touch sensor feedback (via physically pressing the sensor), it would proceed to the next zone.	Outputting the current points per shot values to the display and checking the zone it should move to based on hand calculations.
Shooting Mechanics	To ensure that the angles were calculated correctly, and that the robot was able to position the shooting mechanism correctly and release the ball with consistency.	The shooting mechanism would rotate in the horizontal direction, then elevate in the vertical direction. After it was positioned, it would shoot the ball.	Manually moved the robot to different locations on the board and ran the function to see the accuracy of the algorithms used to calculate the angles.

These were all the tests ran before putting the program together and running the entire program to make sure it worked in sequence. The reader should note that the shooting mechanics test was not

passed as the angles were not calculated correctly and we did not allow ourselves enough time to properly debug the calculating functions.

Problems Encountered and Resolutions

One of the problems encountered related to data storage as the robot has a limited memory capacity. Through trial and error, we determined that we could not store more than 120 values in an array. We originally had planned to store more than this amount, however, we limited it to 75 values based on NBA shooting attempt statistics [2].

The largest problem encountered was the lack of functionality of the TETRIX Standard Servo motors. The night before the demonstration, when testing the Servo motors, they only rotated in every direction, meaning the position of the shooting mechanism was unable to reset after shooting. Unfortunately, due to a miscommunication in testing times, this problem was solved the morning of demo day. Once the new servo motors and connecting platform were successfully installed, there was not enough time to debug the functions calculating the vertical and horizontal shooting angles, which would send the angle to the servo motors.

Criteria (Requirements) and Constraints

Requirements

As stated previously in the preliminary design report, two specific requirements, set by the designers, will have to be met to meet the safety requirements and for the project to work successfully. These specific requirements are further explained below.

Requirement #1

The first requirement pertains to the distance at which the design must shoot from. **The distance at which the design shoots should be 53.24 cm or less**. This was set concerning the dimensions of an official NBA court and was also stated previously in the preliminary design report. This criterion helped to set restrictions on the project design as the dimensions of the design has to be within the dimensions of the court otherwise the shooter cannot be placed on the court.

One change that was made to set a justifiable measurement is to half the size of the length so instead of the length being set at 94 cm, it has now become 47 cm, as shown in Figure 19 below.

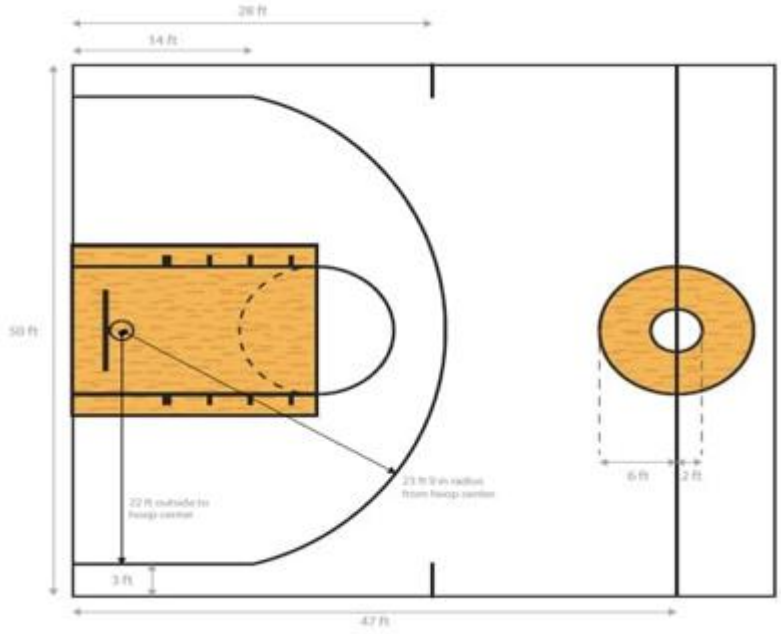

Figure 19: NBA Half Court and Dimensions *[3]*

Another change that was made was the placement of the backboard to meet the distance requirement. For the design to meet this requirement, the backboard had been placed farther away from the edge of the board. This is different from the original idea as it was decided to place the backboard right on the edge of the board.

Requirement #2

The second requirement pertains to the success of the final design. To determine whether the design is a success or not, **the robot should have a score within the range of ± 20% of the score of the opponent**. This was set keeping in mind that the design will not be able to match the exact number of points made by the opponent. A better way to look at this would be by making the design score within a certain range. This is the most important criterion of the design as it determines how successful the final design is.

For this specific criterion, only one change had been made in terms of deciding the success criteria. At first, it was decided that the robot and opponent's score would be compared based on the number of baskets made, but it was then changed to compare the number of points as any NBA game winner is decided based on how many points the team has scored, not how many baskets a team makes. Not only that, because this is a game, it had to have made sure that the design will not always win every round against the opponent. If that was the case, then the opponent will not wish to continue as they will not have any hope to try and win the other rounds. Hence why the decision of this requirement came in place to ensure the success of the final design.

Constraints

As stated previously in the preliminary design report, two specific constraints, set by the designers, are to be met to meet the safety requirements and for the project to work successfully. These specific requirements are further explained below.

Constraint #1

To keep the interest of the game and to make sure that the design is not affected in any way, **the design must not run for more than 30 minutes**. This constraint was set to keep the design as efficient as possible during its testing and showcase during demo day. When compared to the other constraints and criteria of the final design, this constraint is not as valuable, yet it is still important to set this constraint as it makes sure that design will run without any decrease in power from the battery. The battery is shown in Figure 20 below.

Figure 20: LEGO MINDSTORMS EV3 Rechargeable DC Battery *[4]*

At first, it was decided to let the maximum amount of time, set by the user, be the amount of time that the EV3 battery can run for, however, the maximum amount changed to 30 minutes due to a gradual decrease in power occurring well before the EV3 battery runs out. This is necessary as the power level of the battery will be able to provide more power to run the motors on the robot [5]. Figure 3 above shows this EV3 battery.

Constraint #2

For the design to stay stable, not topple over, and be able to make a shot, **the shooting arm must not be over 3 kilograms in weight**. The weight of this design was always kept in mind as a constraint for a variety of reasons. If the weight of the design was not established, then the design can either topple over or will not be able to stay stable before, during, or after the design takes the shot. For the design, the designers used beams on the sides to keep the shooting arm balanced and make sure that it moves properly in the x and y directions.

If the weight of the shooting arm exceeds 3 kilograms, then the beams will bend towards the inside. Not only will the design not look aesthetically pleasing, but this will decrease the shooting accuracy of the design causing the design not being able to meet the shooting requirement but also change the entire shooting mechanism of the design, making it valuable to the final design. No changes were made to this constraint as the weight had been set from the start. The entire implementation of the final design relied on this specific constraint so changing the value of the weight would then change the entire final design hence why no change had been made.

Verification

For the final design, only two constraints were set by the designers in order to make sure that the demo of the project is a success. These constraints have already been stated and further explained in the Constraints and Criteria section of this report.

Updated List of Constraints

1. **The final design must run for an inputted time of 30 minutes or less**
2. **The shooter of the final design must weigh 3 kilograms or less**

In order to meet these constraints, many steps and precautions were taken to make sure that we can accomplish meeting these constraints. In the end, both constraints were met successfully in time for the demo of the project. It was imperative that our design constraints were met as they are used to determine the success of the final design. If they were not met in time for the demo, our design would not have been successful.

Constraint #1

As stated above and in the Constraints and Criteria section, the first constraint that was set by the designers was for the final design to run for no more than 30 minutes. The explanation as to why this constraint was set can be read in the Constraints and Criteria section as well. To meet this constraint, the restrictions of the time were set in the setTimer function of the design code.

Using while loops and if statements, the setTimer function was made for the user to input the time at which the design would run for. However, if the user inputted a time greater than 30 minutes or a negative time, then the code will exit and will have to be compiled again. This helped to meet the constraint as it required the user to input a time between 0 to 30 minutes so that the design can proceed to start shooting.

Constraint #2

As stated above and in the Constraints and Criteria section, the second constraint that was set by the designers was for the shooter of the final design to weigh 3 kilograms or less. The explanation as to why this constraint was set can be read in the Constraints and Criteria section as well. In order to meet this constraint, each part of the shooter, which includes the shooting mechanism and the mount, was weighed after being 3D printed.

All the parts of the shooter were 3D printed so it was essential to make sure that each part was dimensioned to not only meet the criteria that was set but also to make sure that a single part will not weigh more than it should. As the cumulative weight of the shooter cannot exceed 3 kilograms, it was made sure that each part wouldn't weight a lot. In the end, the shooter placed on the final design weighed 2.7 kilograms, successfully meeting this constraint.

Project Plan

The project plan had been held back many times due to inefficient parts, delays in 3D printing the parts, and laser cutting certain parts (for eg – the racks used for moving the robot in the X and Y directions). The team had also predicted that the design would be mechanically and software intensive.

For that reason, the team planned on having a balance of planning the mechanical and software aspects of the ShotBotTM. However, the team failed on following the plan as the focus was mainly on the hardware aspect, completely ignoring the software design until the end. This led to the robot's physical system being built at a very delayed time, causing the team to use a set of code that had not been debugged accurately.

Even though the tasks were split into different sub-teams with their respective leaders, each team member had equally taken part to work on a certain task, until the milestones (planned by the team) were met and were accomplished on time.

Task Assignment

As stated in the Preliminary Robot Report, the team for the ShotBot™ had been divided into sub-teams, each of them with their own sub-team leader. Abhimanyu lead the design tasks and proposed design revisions and changes to the ShotBot™. Ansh lead the communications and technical tasks, such as the reports and design logs for the mechanical and software part of the project. Balen had led the team in the software element of the project by creating the additional trivial functions needed for the ShotBot™. Neil led the team in completing the tasks and milestones planned for the ShotBot™ and ensured the completion of those tasks at the right time.

In short, the project had been subdivided into 4 different sub-teams, namely the Design, Communications, Programming, and Mechanical sub-teams. Each sub-team has its lead:

- Abhimanyu Dev Singh – Design Lead:

Abhimanyu was the head of designing for the ShotBot™ and implementation of the calibrating mechanism into the robot. He was also responsible for the shooting mechanism on the robot and implementing it in the ShotBot™ [1].

- Ansh Sahny – Communications Lead:

Ansh was the head of communications, who had documented everything with the help of pictures and sketches. He had kept track of all the overall activities of the team, by recording designs and work done by the team [1].

- Balen Seeton – Software Lead:

Balen was the mastermind of the software that had helped the ShotBot™ work. He had planned and implemented the necessary decision-making and robots' shooting mechanism's code [1].

- Neil Fernandes – Project Coordinator:

Neil had taken responsibility for the team and made sure that the milestones planned and established in the GANTT chart had been completed on time. He also ensured there is coordination between the members of the ShotBot™ team [1].

The team had also initially planned on building a main court (from the 4th to the 6th of November) on which the colors red, yellow and blue, on which the robot would have made certain decisions and move to a strategic position from which it would shoot using the color sensor. Unfortunately, the idea was scrapped and was replaced by a new plan in which the team decided to hardcode certain positions on the color regions.

Revisions to Project Plan

The team had to finish the shooting mechanism on the 16th of November, but due to a lack of parts and materials, the shooting mechanism was fully functional (with the system responsible for moving the robot) only on the 20th of November. This caused the team to have only one day to debug the code for the overall robot system. Therefore, the team could not finish debugging the code on time.

Despite the circumstances, the team completed the milestones that were stated in the Preliminary Design Report for the ShotBotTM, namely creating the design report, programming the decision – making code, designing and building the shooting mechanism, and even testing and debugging the system. The team is currently finishing up the last milestone, which is this report. This timeline is seen in Figure 21, as previously shown in the Preliminary Design Report [1].

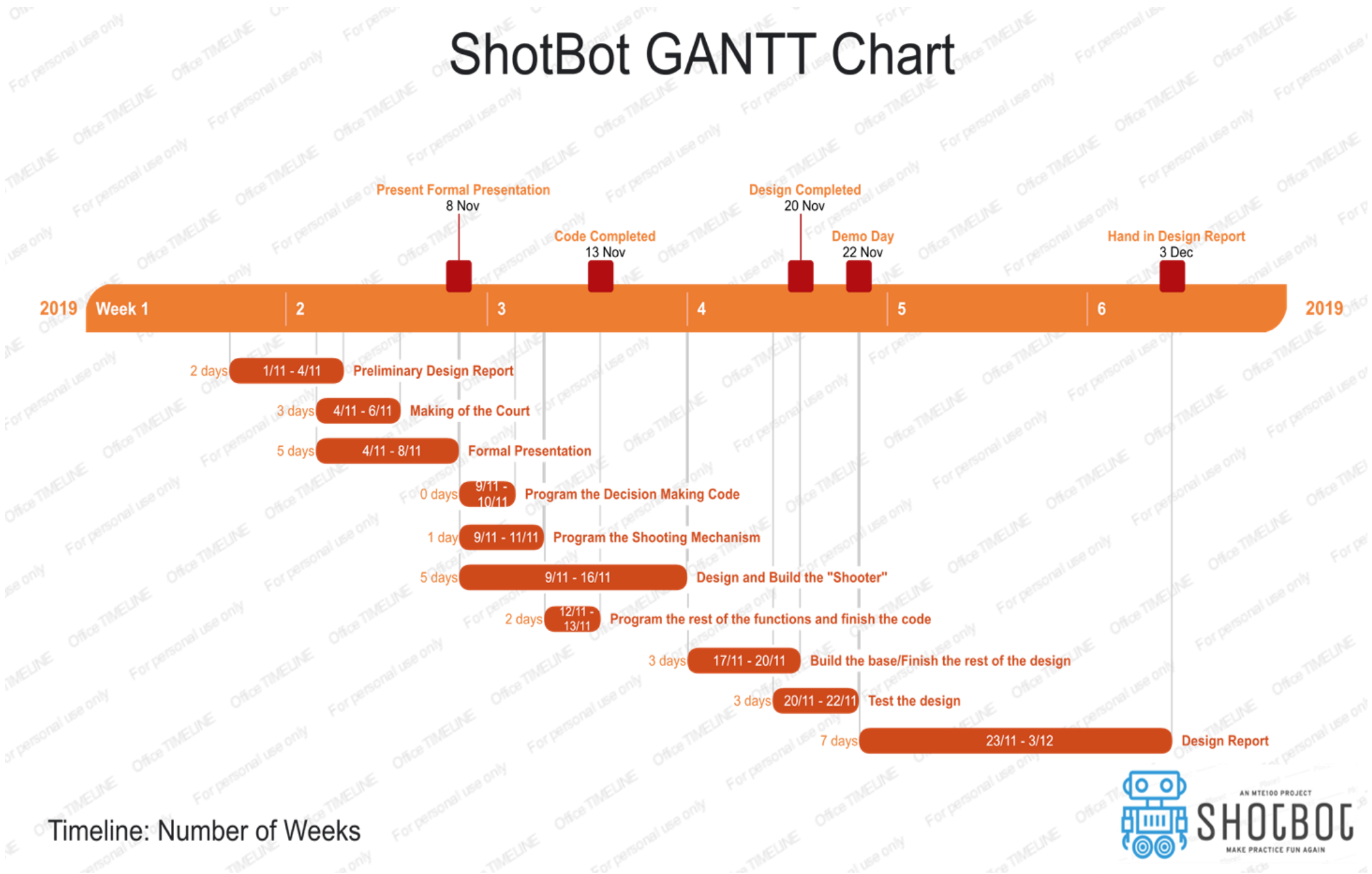

Figure 21: GANTT chart for the robot [1]

Conclusion

In Conclusion, ShotBot™ is a robotic shooting partner designed to make shooting practice fun again, and in return, improve the shooting percentage of the opponent/user. The key task of the ShotBot™ is to input a list of data containing the number of shots taken, anticipated points for each shot taken, the number of successful shots, total points scored, and the duration of the round (all done by the user) and in return replicates the shots and tries to shoot with similar statistics as the opponent(CITE). The design for this product was based on a rack and pinion-based shooting mechanism, with rails and gears to move the robot along the x and y axes. The robot always used ultrasonic sensors to detect its location on the court -. It would move to a designated point in a location decided by a decision-making algorithm based on how to approach an average. Unfortunately, the design was not capable of meeting all the criteria as it was not consistent enough to finish within 20% of the user's points every game.

Recommendations

Mechanical Design

Although the mechanical design was fully thought out before printing laser cutting, 3D printing, and assembling, there were small flaws in the design. One problem with the mechanical system was the inability to drive at a constant speed across the court. At some points on the rails, the robot struggled to maintain its velocity. This could have been solved by replacing the LEGO axles with wooden axles for less flexibility. An increase in laser cutting accuracy could have also partially solved this problem.

Software Design

The software design was planned out in advance of the programming; however, it involved many spontaneous changes to the source code throughout the process. If more time was allotted, the software could have been made more robust by adding an extra function for file input and processing the data and implementing it into the array. In turn, with the use of this function, a more complete main would have been used as opposed to a function that served as the main except for the configuring of sensors.

References

[1] B. Seeton, N. Fernandes, A. Sahny and A. D. Singh, *ShotBotTM – MTE100/GENE121 Final Project Prelimnary Design Report,* Waterloo, 2019.

[2] Basketball Reference, "NBA Team Regular Season Records for Field Goal Attempts," 9 November 2019. [Online]. Available: https://www.basketball-reference.com/leaders/team_fga.html. [Accessed 12 November 2019].

[3] Basketball Goal Store, "Basketball Court Building Guide - Part 2," Basketball Goal Store, 21 April 2016. [Online]. Available: https://basketballgoalstore.com/blog/basketball-court-building-guide-part-2/. [Accessed 1 November 2019].

[4] Amazon, "LEGO MINDSTORMS EV3 Rechargeable DC Battery 45501," Amazon, [Online]. Available: https://www.amazon.com/LEGO-MINDSTORMS-Rechargeable-Battery-45501/dp/B00E1P57TE. [Accessed 1 November 2019].

[5] T. Bickford, "Robot Batteries, an Overview," May 2016. [Online]. Available: http://www.mainerobotics.org/uploads/8/3/4/4/8344007/robot_batteries_an_overview_-_google_docs.pdf. [Accessed 1 November 2019].

Appendices

Appendix A

The Source Code of the setTimer Function:

```
float setTimer()
{
  int count = 0;
  while (count >= 0 && count <= 30)
  {
    displayString(7, "Shooting Time");
    displayString(8, "%d:00", count);

    while (!getButtonPress(buttonAny))
    {}

    if (getButtonPress(buttonLeft))
    {
      while(getButtonPress(buttonLeft))
      {}
      count--;
    }
    else if (getButtonPress(buttonRight))
    {
      while(getButtonPress(buttonRight))
      {}
      count++;
    }
    else if (getButtonPress(buttonUp))
    {
      while(getButtonPress(buttonUp))
      {}
      count += 5;
    }
    else if (getButtonPress(buttonDown))
    {
      while(getButtonPress(buttonDown))
      {}
      count -= 5;
    }
    else
    {
      break;
    }
  }
  return count;
}
```

The Source Code of the calcHorAngle, calcDistance, and calcVertAngle Functions:

```
int calcHorAngle(float xDist, float yDist)
{
  int angle = 0;
  angle = atan2(30-yDist, -(20-xDist)) * 180/PI;
  if(angle > 90)
  {
    angle = angle - 180;
  }

  return (int)angle;
}

float calcDistance(float xDist, float yDist)
{
  float dist = 0;
  dist = sqrt(pow((30-yDist),2) + pow((20-xDist),2));
  return dist;
}

float calcVertAngle(float xDist, float yDist)
{
  int theta = 0;
  float distance = calcDistance(xDist, yDist);

  float arcsinInput = pow(INITIAL_VELOCITY,2.0)/(9.8 * distance);
  theta = asin(arcsinInput) * 180/PI;
  return theta;
}
```

The Source Code for the shotMechanics and roundingAlg Functions:

```
void shotMechanics(float xDist, float yDist)
{
  float horAngle = calcHorAngle(xDist, yDist);
  float vertAngle = calcVertAngle(xDist, yDist);

  setServoPosition(S4, 1/*Servo Number*/, -10);
  setServoPosition(S4, 2/*Servo Number*/, 10);

  motor[motorC] = 100;
  time1[T1] = 0;
  while(time1[T1] < 3700)
  {}
  motor[motorC] = 0; // LAUNCH POINT

  setServoPosition(S4, 1, 0);
  setServoPosition(S4, 2, -45);
}


int roundingAlg(float number)
{
  float smallestDifference = ZONES * MAX_SHOTS_TAKEN; // Max Difference Possible
  int closestInt = 0;
  int i = 1;

  for(i = 1; i <= ZONES; i++)
  {
    if(fabs(number - i) < smallestDifference)
    {
      smallestDifference = fabs(number - i);
      closestInt = i;
    }
  }

  return closestInt; // Closest Integer from 1 to 3
}
```

The Source Code for the makeDecision function:

```
int makeDecision(int numOfShots, float & currentPps, float desiredPps)
{
  float decisionValue = 0;
  int colourDecision = 0;

  decisionValue = desiredPps * (numOfShots + 1) - currentPps * (numOfShots); // DECISION MAKING ALGORITHM
  displayString(4,"DecisionValue: %f", decisionValue);
  displayString(5, "DesiredPPS: %f", desiredPps);
  wait1Msec(5000);

  colourDecision = roundingAlg(decisionValue);

  return colourDecision;
}
```

The Source Code for the moveToPos Function:

```
void moveToPos(int cDecision, float xDist, float yDist, float & xDecision, float & yDecision)
{
  int pDecision = 0;

  if(cDecision == 3)
  {
    pDecision = random(NUM_RED - 1);
    xDecision = RED_POINTS[0][pDecision];
    yDecision = RED_POINTS[1][pDecision];
  }
  else if(cDecision == 2)
  {
    pDecision = random(NUM_BLUE - 1);
    xDecision = BLUE_POINTS[0][pDecision];
    yDecision = BLUE_POINTS[1][pDecision];
  }
  else
  {
    pDecision = random(NUM_YELLOW - 1);
    xDecision = YELLOW_POINTS[0][pDecision];
    yDecision = YELLOW_POINTS[1][pDecision];
  }

  if(xDecision > xDist)
  {
    motor[motorA] = 25; // MOTOR IN X-DIR
    while(SensorValue[S1] < xDecision)
    {}
  }
  else
  {
    motor[motorA] = -25; // MOTOR IN X-DIR
    while(SensorValue[S1] > xDecision)
    {}
  }
  motor[motorA] = 0;

  if(yDecision > yDist)
  {
    motor[motorB] = -25; // MOTOR IN Y-DIR
    while(SensorValue[S2] < yDecision)
    {}
  }
  else
  {
    motor[motorB] = 25; // MOTOR IN Y-DIR
    while(SensorValue[S2] > yDecision)
    {}
  }
  motor[motorB] = 0;

}
```

The Source Code for the decisionMaking Function:

```
void decisionMaking()
{
  TFileHandle inFile;
  bool fileOkay = openReadPC(inFile, "shotData.txt");

  if(!fileOkay)
  {
    displayString(5,"Error - File Not Available");
    wait1Msec(5000);
  }
  else
  {
    //INITIALIZATION STARTS
    int i = 0;
    int shotResult[MAX_SHOTS_TAKEN];
    int shotPoints[MAX_SHOTS_TAKEN];

    for(i = 0; i < MAX_SHOTS_TAKEN; i++)
    {
      shotResult[i] = 0;
      shotPoints[i] = 0;
    }
    //INITIALIZATION ENDS

    //READING IN FROM FILE STARTS
    int time = 0;
    int shotsTaken = 0;
    int pointsScored = 0;
    float pointsPerShot = 0;
    readIntPC(inFile, time);

    i = 0;
    while(readIntPC(inFile,shotResult[i]) && readIntPC(inFile,shotPoints[i]))
    {
      if(shotResult[i] == 1)
      {
        pointsScored += shotPoints[i];
      }

      if(i > MAX_SHOTS_TAKEN)
      {
        break;
      }

      i++;
    }
    shotsTaken = i + 1;
    pointsPerShot = (float)pointsScored/(float)shotsTaken;
    //READING IN FROM FILE ENDS

    //DECISION MAKING PROCCESS
    i = 0;
    int finalDecision = 0;
    int robotPoints = 0;
    float robotPointsPerShot = 0;
    int shotsMade = 0;

    float count = setTimer();
    time1[T2] = 0;
    while(time1[T2] < count * 60 * 1000)
    {

      float xDec = 0;
      float yDec = 0;
      moveToPos(finalDecision, SensorValue[S1/*Ultrasonic 1*/], SensorValue[S2/*Ultrasonic 2*/],xDec,yDec);
      shotMechanics(xDec, yDec);
```

```
      time1[T1] = 0;
      while((SensorValue[S3] == 0) && (time1[T1] < 2000))
      {}

      if(time1[T1] < 2000)
      {
        shotsMade++;

        if(SensorValue[S2] < 11) // Y-dir
        {
          robotPoints += 3;
        }
        else if((SensorValue[S2] >= 15.5 && SensorValue[S2] < 18.5)&&(SensorValue[S1] >= 15.5 && SensorValue[S1] < 24.5))
        {
          robotPoints += 1;
        }
        else
        {
          robotPoints += 2;
        }
      }
      robotPointsPerShot = (float)robotPoints/(i + 1);

      finalDecision = makeDecision(i, robotPointsPerShot, pointsPerShot);
      i++;
    }

    float shotPercentage = (shotsMade/i + 1) * 100;
    displayBigTextLine(4, "SHOTBOT Shot Percentage:");
    displayBigTextLine(6,"%f%%", shotPercentage);
    while(!getButtonPress(buttonAny))
    {}
    while(getButtonPress(buttonEnter))
    {}

  }
}
```

The Source Code for the main Function:

```
task main()
{
  SensorType[S1] = sensorEV3_Ultrasonic;
  wait1Msec(100);
  SensorType[S2] = sensorEV3_Ultrasonic;
  wait1Msec(100);
  SensorType[S3] = sensorEV3_Touch;
  wait1Msec(100);
  SensorType[S4] = sensorI2CCustom9V; // HORIZONTAL & VERTICAL MOVEMENT
  wait1Msec(100);

  decisionMaking();

}
```

Global Constants and Libraries Used:

```
#include "PC_FileIO.c"
#include "EV3Servo-lib-UW.c"

const int MAX_SHOTS_TAKEN = 75; // Record for FGA in a Game / 48 * 30
const int ZONES = 3;

const int NUM_RED = 7;
const int NUM_BLUE = 5;
const int NUM_YELLOW = 2;

const int RED_POINTS[2][NUM_RED] = {{4,8},{36,8},{10,6},{30,6},{7,4},{33,4},{20,4}};
const int BLUE_POINTS[2][NUM_BLUE] = {{8,20},{30,20},{12,17},{27,17},{20,14}};
const int YELLOW_POINTS[2][NUM_YELLOW] = {{19,17},{21,17}};

const float INITIAL_VELOCITY = 18.55;
```