**Final Robot Project:**

**Tennis Ball Training Machine**

**A Report Prepared For:**

MTE100 and MTE121

**Prepared By:**

Group 8-28

Ben Boguslavsky – 21010767

Louie Shen – 21016126

Ethan Longo – 20994891

Nadish Madadi – 21016418

**Date of Submission:**

December 6, 2022

# Summary

The Tennis Ball Training Machine is a prototype robot. Its goal is to act as a convenient alternative to ball machines and coaches to help players train. The version demoed intakes Tetrix balls, can drive to a set position on the court and shoots the balls at random angles over a designated number of rounds set by a user.

Some constraints included the materials given; the size of tennis balls is too large for the given resources to handle; thus, the robot is a downscaled prototype.

The mechanical components consist of two major parts– the conveyor belt and the flywheels. There are sub-components within each main component, such as the color sensor in front of the conveyor belt, and the gears on the flywheels. The conveyor belt’s main function is to collect and store balls and move them into the flywheels. The function of the flywheels is to spin rapidly and launch Tetrix balls that come from the conveyor belt.

The software consists of 7 non-trivial functions, not including general rotation and driving a certain distance. The functions are responsible for sequences such as shooting or intaking, and others are responsible for driving to a specified location, getting user inputs, or moving the conveyor a fixed length.

Multiplexers proved to be an issue, due to their small transmission power and faulty communication with the Lego Mindstorm brick. This ultimately stopped the group from meeting a certain constraint during the demo.

The group’s schedule for the construction of this robot was structured to finish the project several days prior to the demo, however, issues came up as expected which delayed finishing the project, meaning the additional days in the schedule were used to fix the issues.

Table of Contents

[Summary i](#_Toc120983721)

[List of Figures and Tables iv](#_Toc120983722)

[1 Introduction 1](#_Toc120983723)

[2 Scope 1](#_Toc120983724)

[2.1 Functionality 1](#_Toc120983725)

[2.2 Inputs 3](#_Toc120983726)

[2.3 Environment Interaction 4](#_Toc120983727)

[2.4 Task Completion 5](#_Toc120983728)

[2.5 Changes 5](#_Toc120983729)

[3 Constraints and Criteria 7](#_Toc120983730)

[3.1 Changes Throughout the Project 7](#_Toc120983731)

[4 Mechanical Design 8](#_Toc120983732)

[4.1 Overall Design 8](#_Toc120983733)

[4.2 Drivetrain 9](#_Toc120983734)

[4.3 Conveyor Belt 10](#_Toc120983735)

[4.4 Flywheel Shooter 11](#_Toc120983736)

[4.5 Senor Attachment Design 11](#_Toc120983737)

[4.6 Motor Attachment Design 12](#_Toc120983738)

[4.7 Trade-Offs 13](#_Toc120983739)

[5 Software Design and Implementation 13](#_Toc120983740)

[5.1 Software Design Description 13](#_Toc120983741)

[5.1.1 Software Task List 13](#_Toc120983742)

[5.1.2 Program Breakdown into Blocks 14](#_Toc120983743)

[5.1.3 Data Storage 19](#_Toc120983744)

[5.2 Trade-offs 19](#_Toc120983745)

[5.3 Testing 19](#_Toc120983746)

[5.4 Significant Problem Solution 20](#_Toc120983747)

[6 Verification 20](#_Toc120983748)

[7 Project Plan 21](#_Toc120983749)

[7.1 Splitting Up Tasks 21](#_Toc120983750)

[7.2 Scheduled Plan vs Actual Progress 22](#_Toc120983751)

[7.3 Revisions to the Project Plan 23](#_Toc120983752)

[8 Conclusion 24](#_Toc120983753)

[9 Recommendations 24](#_Toc120983754)

[10 References 25](#_Toc120983755)

[11 Appendix A – RobotC Code 26](#_Toc120983756)

[12 Appendix B – Software Flowchart 35](#_Toc120983757)

# List of Figures and Tables

[Figure 1: Ball height relative to robot positions 2](#_Toc120985814)

[Figure 2: Robot with ramp 2](#_Toc120985815)

[Figure 3: Flywheel position after rotations 3](#_Toc120985816)

[Figure 4: Angle range of robot 4](#_Toc120985817)

[Figure 5: First version of robot with separate intake system 6](#_Toc120985818)

[Figure 6: Sweeping motion of original robot drive task 6](#_Toc120985819)

[Figure 7: Final conveyor belt setup 8](#_Toc120985820)

[Figure 8: Completed robot with all components 9](#_Toc120985821)

[Figure 9: Drivetrain of robot showing frame and placement of wheels 9](#_Toc120985822)

[Figure 10: Conveyer Belt and Drivetrain together 10](#_Toc120985823)

[Figure 11: Conveyer Belt with Tray Side View 11](#_Toc120985824)

[Figure 12: Flywheel System Spinning and Stationary 11](#_Toc120985825)

[Figure 13: Color and Gyro Sensor Placement 12](#_Toc120985826)

[Figure 14: Driving, Shooting, and Intake Motor Placement Top View 12](#_Toc120985827)

[Figure 15: Current Intake System (Left) and Initial Intake System (Right) 13](#_Toc120985828)

[Figure 16: Choosing rounds display menu 15](#_Toc120985829)

[Figure 17: numRounds function 15](#_Toc120985830)

[Figure 18: Ball height selection display menu 15](#_Toc120985831)

[Figure 19: ballHeight function 16](#_Toc120985832)

[Figure 20: moveConveyor function 16](#_Toc120985833)

[Figure 21: intakeSequence function 17](#_Toc120985834)

[Figure 22: cornerToCenter function 18](#_Toc120985835)

[Figure 23: shooting function 18](#_Toc120985836)

[Table 1: Finding ideal conveyor belt movement distance 20](#_Toc120985797)

[Table 2: Met and failed constraints 21](#_Toc120985798)

[Table 3: Schedule and completion of mechanical components 22](#_Toc120985799)

[Table 4: Schedule and completion of software 23](#_Toc120985800)

# Introduction

Tennis is a sport that usually requires multiple players for practice. This leads to players being unable to receive adequate training without resorting to paying for lessons. In today’s world, coaches are a great expense for the average human being. While ball machines do exist, there is no guarantee that there is one available at local community centers or outdoor public courts. Additionally, ball machines do not give a player a wide enough range of motions to practice thoroughly and effectively, since they only shoot the ball in one direction. Thus, a robot that provides more range of practice than a ball machine and is less of a financial struggle for players who cannot afford coaches is one that should exist in all public tennis court spaces. This is the problem that is tackled by “The Tennis Ball Training Machine”, which aims to provide a customizable experience for the user who wishes to efficiently practice tennis on their own. However, due to a lack of materials able to support real tennis balls, a downsized prototype of the robot is necessary, with Tetrix balls acting as tennis balls and a downsized court simulating a real tennis court.

# Scope

## Functionality

The tennis ball machine will perform three major tasks:

* Intaking the Tetrix balls
* Driving to a designated location on the court and back
* Shooting the Tetrix balls

Intaking balls is done through two identical parallel conveyor belts with built-in divider pegs to separate balls from each other. The two conveyor belts connect through a gear system attached to a large motor, which turns both conveyor belts the length of one ball space each time a ball is detected. This allows the intake system to follow an organized fashion once a new ball is detected.

The robot’s driving task is implemented through two large motors attached to wheels as well as a ball pivot to give the tennis ball machine stability. The robot moves to the center of the court horizontally and moves to a designated spot on the court vertically depending on the input provided by the user during the start up procedure.

This allows the player to train on different ball heights. For example, if the player wants to try hitting a ball that is higher than normal, the robot positions itself closer to the net so that the ball will reach the player at a higher angle. If a low shot was selected, the robot positions itself further away from the net so that the ball would reach the player at a lower height (see Figure 1). After completing the shooting task, the robot returns to its original position through the same process of moving horizontally first, followed by the same length vertically as it did to move forward.

Diagram

Description automatically generated

Figure : Ball height relative to robot positions

Shooting the Tetrix balls is done through two flywheels, each connected to a large motor. These motors attach to a two-layer gear system, which links to the flywheels. The ball hits a cardboard ramp attached to the robot giving the ball the height to go over the net (see Figure 2).

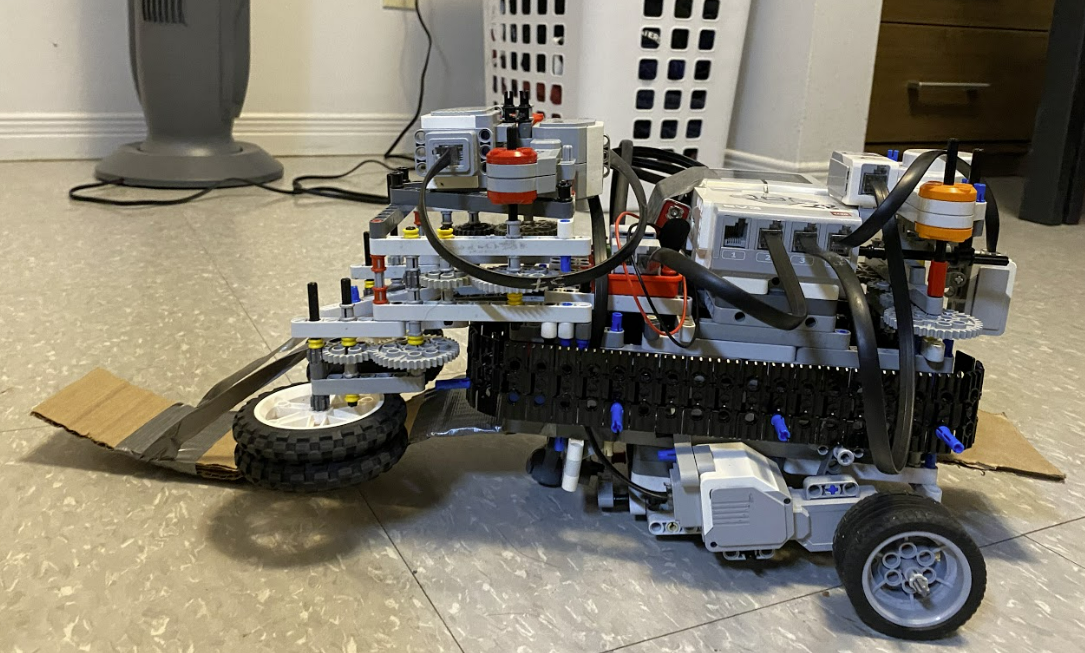


Figure : Robot with ramp

Each time before a ball is shot, the robot rotates within a constrained angle range using the driving motors to give the player variance in their practice. The conveyor belt system also moves one specific length each time it is required to move. This length ensures that when the conveyor belt moves, the pegs are in the same relative position as before, thus allowing organized storage of the balls, as well as organized output when the balls need to be shot through the flywheels. The robot turns the random angle first, then shoots by moving the conveyor one time to push a ball to the flywheels. The robot will then rotate back to its original position to stay within the constrained angle range.

## Inputs

The robot uses four different inputs:

* Color sensor
* Gyro sensor
* Motor Encoders
* User Input through the Lego Mindstorm brick

A color sensor is used to detect Tetrix balls. The sensor is positioned to sense in front of the conveyor belt before the user can insert a ball into the intake system. Since the Tetrix balls being stored by the robot are all yellow, the conveyor belt only intakes the ball if the color yellow is detected. This ensures that only Tetrix balls are permitted inside the robot, minimizing internal damage from other foreign objects. If another color is sensed other than yellow, the conveyor belt remains stationary and will continue to wait for a Tetrix ball to be detected. This process continues until a total of 3 balls are sensed and stored by the robot. Following the intake procedure, the robot will then proceed to the driving task.

The gyro sensor is used in the driving and shooting process of the robot. During driving, a 90-degree turn clockwise must be made for the robot to move from vertically to horizontally along the court. The robot rotates 90 degrees clockwise after reaching the appropriate spot to point the flywheels toward the player, as shown in Figure 3.

Diagram

Description automatically generated

Figure : Flywheel position after rotations

The gyro sensor is also used during the shooting process to track the amount the robot needs to rotate to turn a random angle between -30 and 30 degrees as shown in Figure 4. After the initial rotation, the gyro sensor detects whether the robot returns to the original zero angle before the shot was taken. This is to ensure that the robot’s rotations stay within the -30 and 30-degree range for every shot and rotation.

Diagram

Description automatically generated

Figure : Angle range of robot

Motor encoders are used in the conveyor belt and driving to ensure motors are consistent in the distance they move.

The Mindstorm brick button inputs are used in the start up of the robot. During start up, the user is prompted with a question asking how many rounds they want the robot to run for. This can be controlled with the left and right buttons, which will decrease or increase the number of rounds respectively. The enter button can be pressed when the user is satisfied with the set number of rounds. The number of rounds selected will communicate with the code, and the robot will repeat the intake, driving, shooting, and returning tasks for the set number of rounds.

The next prompt, which follows the round number selection, will ask the user the desired height of the ball when it is launched from the robot. There are three different settings: high ball, normal ball, and low ball which correspond with the up, enter, and down buttons respectively. This input will tell the code what factor to multiply the vertical drive distance with to reach the appropriate distance from the net.

After the round and height input tasks are finished, the robot will begin the intake task.

## Environment Interaction

The robot is placed in a 1.35 by 1.35-meter enclosed space with one of the walls simulating the net and the player hitting the ball back toward the robot’s side of the court, being the 1.35 by 1.35-meter area.

The robot will start in the left corner, where the intake process takes place. The motor encoder of the conveyor belt motor is used to ensure a fixed distance is traveled by the conveyor belt each time it needs to store a ball. After the intake, the robot drives to the middle of the court and moves to the designated spot on the court using the large motors connected to the wheels. During the driving process, the encoders in the motors connected to the driving wheels are used to calculate the amount the motors need to spin to reach the correct distance horizontally and vertically.

The robot then begins the shooting process. The two large motors connected to the flywheels will turn on and continue to stay on while the motor connected to the conveyor belt will feed the balls into the spinning wheels. The distance moved by the conveyor belt is very specific and is kept track of using the motor encoder of the motor that powers the belt.

After the shooting is complete, the robot will drive back to the starting point through the driving motors and check if any more rounds need to be played. If there are, intaking will start again and the process will repeat without including the user input.

## Task Completion

The robot will recognize when the user input is finished when the enter button is pressed on the first prompt, and either the up, enter, or down buttons are pressed afterwards (this is for the height of the ball). After the user presses two appropriate buttons, the robot waits for 3 yellow balls to be sensed by the color sensor for the intake process. Once 3 balls have been detected, the intake task is complete. Next, the robot moves towards the center of the court and stops at a certain motor encoder value, marking the end driving task. Following this, the shooting task takes place. Once the robot has shot 3 balls and returned to the zero angle, the shooting task is complete. The robot will once again do the driving task and will use the motor encoders to travel back to the starting point. This process repeats until the number of rounds is reached (all rounds have been played). Once the robot reaches the start point, the robot will shut down the program.

## Changes

The robot went through many iterations of changes. Initially, the robot was going to be able to collect balls independently. This means that instead of feeding the balls into the robot’s intake manually, the robot would be able to pick the balls up itself to get ready for the shooting task. It would have done this through a separate intake system attached to the conveyor belt. This separate intake system would be positioned lower to collect the balls from the ground into the larger conveyor belt system (see Figure 5).

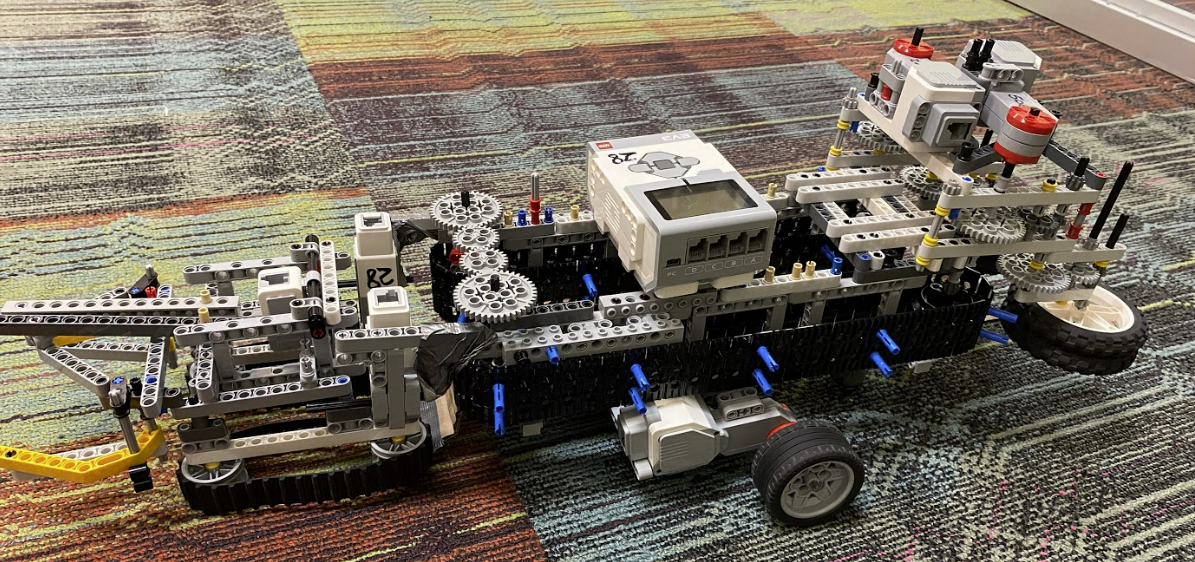


Figure : First version of robot with separate intake system

For this version of the robot, the robot begins with the user input task with the balls already loaded into the conveyor belt. The robot would then drive from the corner to the designated spot on the court and wait a few seconds for the player to get ready on the other side before shooting.

After the robot has shot all the balls within the conveyor belt system and the player has hit them back, the robot begins a sweeping motion (see Figure 6) with the separate intake system turning on. With an attached funnel to the front of the robot, ensuring the balls reach the intake system, the robot would turn on the conveyor belt and move one divider whenever a ball is detected by the color sensor to pass through the separate intake system. Once the sweeping motion covers the entire area of the robot’s side of the court, the robot returns to the designated shooting area on the court to begin shooting once again, assuming that the number of rounds has not been reached.

A picture containing diagram

Description automatically generated

Figure : Sweeping motion of original robot drive task

Aside from the original design requiring a separate intake system and a sweeping motion task, the original plan was also to angle the conveyor belt vertically. With this type of design, the robot would have an easier time shooting the balls over the tennis net. This would eliminate the need for a cardboard ramp to launch the ball over the net and help maintain maximum ball speed due to a lack of friction along the ramp. However, since Lego pieces are restricted to the angle they can produce, the separate intake system was unable to pass balls up the conveyor belt system and into an angled shooting mechanism.

In addition, the size of the original robot design increased drift significantly when driving and made it increasingly difficult to attach parts in a stable manner, ultimately leading to the decision to condense the size and functionality of the robot. Thus, the robot needed to be changed in scope to fit within the restraints of the materials provided and fulfill the intended goal of the project.

# Constraints and Criteria

The criteria list tests to ensure that all tasks were completed accurately and with precision. The list of tasks includes: the conveyor belt moving the length of exactly one divider peg when the color yellow is detected from the ball, the robot driving to the correct positions, the Tetrix balls shooting out one at a time (three times before the robot begins to drive again), the random angle turning between -30 and 30 degrees consistently, and finally, the robot continuing repeat these actions until the number of rounds the user inputted is reached.

The original design, as well as the current design of the robot, is presented with a set of constraints along with its criteria. The constraints include objects on the court that may block the path of the robot, the weight of the robot causing drift when turning, the conveyor belt jamming due to a lack of power and working communication of multiplexers, and lastly the conveyor belt only being able to hold a maximum of 3 balls due to the size of the robot.

## Changes Throughout the Project

Within the span of designing the initial robot to the final design, changes were made to the criteria and constraints. The initial criteria were to include the sweeping motion in the intake process and moving the conveyor belt once the ball reached the end of the separate intake system. This was changed due to the robot being unable to bring the balls from the low intake system into the large main body conveyor belt. As a result, the sweeping motion was unable to be implemented and had to be removed from the criteria.

Within the requirements and constraints, there were some notable ones that helped guide the development and testing of the robot. Particularly, criteria and constraints regarding the conveyor belt were important in deciding how many balls the robot can hold with the given parameters. Since the original size of the robot was too large, the final design was downsized so only 3 balls can fit within the robot. The robot is also downsized overall as tennis balls are too heavy and large to be handled by Lego. Furthermore, the robot requires the Tetrix balls to be organized and spaced properly. Thus, when setting up the pegs to divide the conveyor belt, additional space was given on top of the length of one ball to allow the Tetrix balls to be comfortably organized (see Figure 7).

A picture containing floor, indoor

Description automatically generated

Figure : Final conveyor belt setup

Although there are some criteria and constraints that guided the building process, some of them also did not provide any value. For example, the constraint of the robot drifting while turning was not something that provided value when building and testing the robot. Since the provided wheels would sometimes slip on the smooth surface of the E2 floor, changes to this constraint are not possible. The conveyor belt occasionally getting jammed is also a problem that is unsolvable since the multiplexer provides minimal power in comparison to the motor connected to the Mindstorm brick. The other components of the robot require even greater power than the conveyor belt so switching between the multiplexer and brick is a not viable decision. In short, the conveyor belt is not able to have the power to always run smoothly, occasionally stutter, and pause rotating entirely. These are constraints that had no solution.

# Mechanical Design

## Overall Design

The three systems necessary for the robot to function are the conveyor belt system, the flywheel system, and the drivetrain. The drivetrain provides movement to the robot, allowing it to move around the court and rotate when needed. Secondly, the conveyor belt system consists of an intake system via a color sensor and pushes Tetrix balls toward the flywheel system. The last component of the robot, the flywheel system, acts as an outtake system. The flywheel spins at a fast enough speed so the balls in the robot leave one at a time, acting in conjunction with the conveyor belt. The conveyor belt system is attached to the drivetrain and the flywheel system is attached on top of the conveyor belt system to reduce the length of the robot. Figure 8 shows the complete robot setup.

A picture containing indoor, cluttered, messy

Description automatically generated

Figure : Completed robot with all components

## Drivetrain

The robot drivetrain consists of a 26-hole by 14-hole rectangular frame with 3 bar high elevation from the tray to the ground (see Figure 9). The large frame creates enough room for up to 3 balls to fit within the robot each round. The decision to elevate the frame ensures enough space for wheel placement and a lack of conflict between the conveyor belts. The robot’s weight was a major hurdle to overcome as it is both long and back heavy. To overcome this, two large motors with wheels were attached to the front of the drivetrain with a pivot ball placed further back to counteract the issue of weight distribution. The distance between the two front driving wheels and the pivot ball is 14 holes long, ensuring a rigid base.

A picture containing projector

Description automatically generated

Figure : Drivetrain of robot showing frame and placement of wheels

## Conveyor Belt

The conveyor belt is constructed around a 4-gear system prioritizing torque over speed. It also contains 2 large Lego belts with pegs installed every 5 spaces away (see Figure 10). A large motor is used to power this gear system, approximately a 1:1 ratio, and rotate both conveyor belts at a constrained distance. 4 gears are used to compensate for the large distance within the drivetrain. The conveyor belts are held up by 3 rectangular Lego pieces on each side running the length of the robot. Axles are passed through the ends of each system of 3 rectangular pieces with sprockets attached to them. The conveyor belt must be close to the tray for the pegs to hold the Tetrix balls so spacing between the two is extremely minimal. The conveyor belts are purposefully attached with some slack since Lego motors, even the large ones, cannot handle the inward stress and provide the necessary power to rotate the conveyor belt. Furthermore, this decision ensures the axles do not bend and cause friction along the rectangular frame during rotation.

A picture containing toy

Description automatically generated

Figure : Conveyer Belt and Drivetrain together

Additionally, the conveyor belt cannot function properly without a tray underneath. The decision to use cardboard as the tray of the robot meant friction could be kept minimal especially due to the slow turning of the conveyor belt. The tray makes up for large spaces within the drivetrain that hinder the conveyor belt’s functionality. The tray begins extruded slightly from the front of the robot to ensure balls can be inputted accurately and extend all the way towards the back and under the flywheels to ensure balls can be outputted correctly (see Figure 11).

A picture containing indoor

Description automatically generated

Figure : Conveyer Belt with Tray Side View

## Flywheel Shooter

The flywheel shooter or in other words the outtake system of this robot consists of a two-level gear system powered by two large motors for each flywheel (see Figure 12). As the power output of a large motor cannot spin fast enough to shoot balls from the robot at a professional training level, a gear system was necessary. By compounding the gear system, the robot was able to spin up to approximately 1300 rpm with a gear ratio of 8:1 [1]. Unlike the conveyor belt, speed and time are necessary components of the flywheel.

A picture containing indoor, floor, cluttered

Description automatically generated

Figure : Flywheel System Spinning and Stationary

## Senor Attachment Design

The color sensor is placed at the front of the robot to ensure proper detection prior to the insertion of Tetrix balls. It acts as a precondition for the rest of the robot to function. Next, the gyro is attached to the Lego EV3 brick due to a lack of space on the robot and a lack of non-moving components. Non-moving areas such as the brick would ensure gyro drift is kept minimal. See Figure 13 for sensor placement.

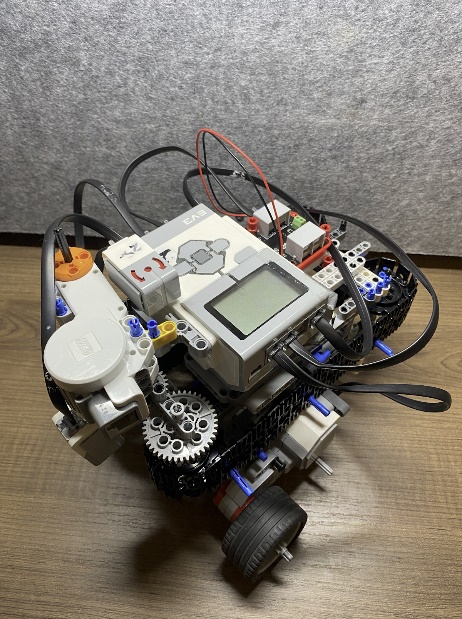


Figure : Color and Gyro Sensor Placement

## Motor Attachment Design

Motors are placed in a specific manner due to the use of a multiplexer. Two large motors are used for driving and two large motors for flywheels. The motors attached to the wheels are at the bottom of the robot, 3 holes away from the tray to ensure there is no friction along them. The flywheel motors are placed on top of the flywheel system to ensure the flywheels remain in one position and reduce energy lost through rigorous vibrations leading to it coming off the robot. Lastly, the final large motor is used towards the intake system but due to its low power output, it is attached to a multiplexer towards the front of the robot. See for Figure 14 motor placement.

A picture containing electronics, camera

Description automatically generated

Figure : Driving, Shooting, and Intake Motor Placement Top View

## Trade-Offs

Initially, a separate intake system was built, as shown in Figure 15, but due to the speed of the intakes and the color sensor being unable to detect ball color at such speed, the conveyor belt itself acts as the intake system. While this allows for balls to be inputted correctly, the trade-off is that the robot is less efficient, needing more human interaction than previously.

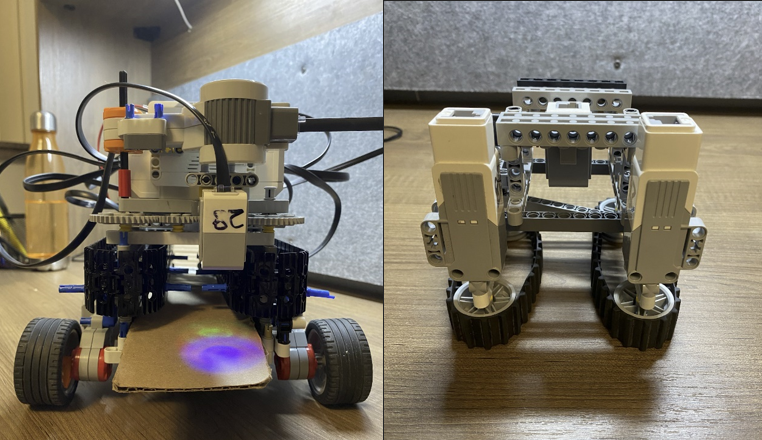


Figure : Current Intake System (Left) and Initial Intake System (Right)

# Software Design and Implementation

## Software Design Description

### Software Task List

The robot’s tasks are divided into 3 steps: Startup, Operation, and Shutdown:

#### Startup (After robot is powered on):

* User inputs number of rounds to play (enter is pressed to confirm number of rounds)
* User inputs the desired height for the balls to be shot throughout the rounds

#### Operations

* Ball is placed under color sensor. If yellow is detected, the conveyor belt moves and intakes the ball
  + Exceptional Operation: If the color detected is not yellow, the conveyor does not move.
* Once three balls are collected inside the robot, robot moves towards the center position of the court, based on the user’s initial height input
  + If “High” height, robot moves to center, but closer to the net
  + If “Normal” height, robot moves to the exact center of the court
  + If “Low” height, robot moves to center, but further away from the net.
* Robot faces flywheels towards the net and turns on flywheels
  + Robot rotates random angle, then moves the conveyor so that the ball is in the flywheel and gets launched
  + Robot returns to initial zero angle, and repeats the above processes two more times
  + Robot drives back to original bottom left corner
* The “Operations” take place for the number of rounds inputted in the startup

#### Shutdown

* Once the number of rounds has been complete, the program ends

#### Changes from Earlier Tasks

The original robot design included a separate set of small conveyor tracks to intake the balls into the conveyor. Based on this, the initial intake task involved the intake conveyors turning on in sequence with the conveyor belt. This made the intake task more complex due to very specific timing requirements.

With the large size of the robot and high chance of drift, it is difficult to have the robot drive around the entire court and collect tetrix balls on its own (this was the original task the robot would follow before returning to the middle and launching the tetrix balls). So, a change was implemented only so that the robot drives to the center, however at a selected height. This minimizes the time of driving, thus minimizing drift when arriving to the center or corner of the court.

### Program Breakdown into Blocks

The code is broken down into 7 non-trivial functions, not including the rotate() function and the driveDist() function:

* numRounds
* ballHeight
* intakeSequence
* moveConveyor
* cornerToCenter
* randomAngle
* shooting

#### numRounds() Function

This function is responsible for the user input of how many rounds is desired to be played. It takes no parameters, but it returns an integer “rounds” which is used in a loop to iterate the program’s operations (on task list) a certain amount of times. The function works within a while loop as seen in Figure 16, constantly prompting the user to press a button if the enter button is not pressed; right button to add one to “rounds”, left to subtract one from “rounds”, or the enter button to confirm the number of rounds and thus exit the while loop. Figure 17 shows the display shown in the display for this function.

A picture containing text, monitor, microwave, indoor

Description automatically generated

Figure : Choosing rounds display menu

#### Diagram Description automatically generated

Figure : numRounds function

#### ballHeight () Function

ballHeight takes to parameters, and prompts the user to press a button once, which selects the height of the balls when being shot. The function shows the options: up button for high, down button for low, and enter button for normal height (see Figure 18). Then, an integer variable in the function is set from 1 to 3. Based on the value, the function returns a double which is used as a multiplier for how far up the court the robot should drive when moving to the center. The value of the doubles that are returned are 0.5 for low, 1 for normal, and 1.5 for high (see Figure 19).

A picture containing text, monitor, microwave, oven

Description automatically generated

Figure : Ball height selection display menu

Diagram

Description automatically generated

Figure : ballHeight function

#### moveConveyor(int MotorPow) Function

An integer parameter for the motor power of the conveyor belt which stores the balls is passed into the moveConveyor function. The belt moves exactly 15.4 cm when the function is called (see Figure 20), a value determined through tests. This allows the pegs in the belt to move at precise intervals and separate the balls evenly when intaking the balls or moving them to the flywheels. No values are returned.

Diagram

Description automatically generated

Figure : moveConveyor function

#### intakeSequence() Function

IntakeSequence’s primary function is to determine whether the color sensor detects yellow, and if yellow is detected, to move the conveyor belt using moveConveyor(). It uses no parameters and has no return values. The function allows 3 balls to be inputted, calling moveConveyor() each time a ball is detected. Once three balls have been stored in the robot, intakeSequence() ends, as shown by Figure 21.

Diagram

Description automatically generated

Figure : intakeSequence function

#### cornerToCenter(float length, float width) Function

This function’s purpose is to drive the robot to its required position when it shoots the ball. The two passed parameters are the dimensions of the court the robot is in. In the demo, the court is a square field, so the program uses the same value for both parameters in this function. The width, however, is also multiplied by the return value of ballHeight(), in order to move the robot closer or further from the net. Initially, the robot drives the distance of width multiplied by the return value of ballHeight(), then rotates ninety degrees clockwise, and drives half the length of the court so it is centered. The function ends after a second ninety degree turn which faces the flywheels towards the net (see Figure 22). This function does not return any values.

Diagram, text

Description automatically generated

Figure : cornerToCenter function

#### RandomAngle() Function

RandomAngle() uses the built-in RobotC random() function to randomly generate a number from 0 to 60. This generated number is used as a random index for an array containing all numbers from -30 to 30, which is a range of angles the robot turns to during the shooting sequence. No parameters are required for the function; however, the return value is the random angle that can be used to tell the robot how much to turn and in which direction.

#### Shooting() Function

Shooting() takes no parameters. Also, it uses the RandomAngle and the moveConveyor function within itself. After turning on the flywheels to maximum power, selecting a random angle using RandomAngle, the robot rotates that given angle and calls the moveConveyor function to pass a ball to the flywheels and shoot it. After this, the robot rotates the negative angle of the previous rotation to return to it’s zero angle. It repeats this process two more times, as seen in Figure 23, and then the flywheels are turned off.

Diagram

Description automatically generated

Figure : shooting function

### Data Storage

Generally, there is no data storage required aside from the random angles, the height, and the number of rounds. The RandomAngle function is used within the shooting function, and the values it returns is stored as a variable “chosenAngle” in the function. Height is stored as a variable “height” in the main program since it is used when passed in the driveToCenter function in main. Also, the number of rounds retrieved from numRounds is stored in a variable “rounds”, which is used in the for loop in main to repeat the general operations the required amount of times.

## Trade-offs

There were two major trade-offs involving a slight change in code.

The first trade-off was the speed of driving and rotation of the robot. Due to the size and weight of the robot affecting drift when driving and turning, the speed of rotation and driving was lowered significantly in the driveDist, rotate, and cornerToCenter functions when they were applied in the main code.

Another trade-off involved the responsiveness of the color sensor. When checking if yellow is detected during the intake sequence, the color sensor would detect yellow hundreds of times when only one ball is under it during a given time period. Thus, a wait timer of two seconds was added between each time the color sensor detects yellow. Although the user would now have to wait two seconds before inserting the next ball, this wait timer would ensure that the intake sequence would not finish early (detecting yellow three times too early), and all three balls could be detected correctly and easily.

## Testing

Testing was broken down based on each function and was also done before all components of the robot were assembled together (The conveyor system was not yet connected to the flywheel system).

Testing of the user input of rounds and height were tested individually on the brick. This was done by outputting the values returned onto the display after running each of the functions that receive a user input. The expected results were simply a display of the intended number, which is what resulted in the end. These tests were important to ensure the program repeated the correct number of times and that the robot positioned itself properly on the court.

Then, the intake sequence was tested. The initial test was to determine how close the ball must be to the color sensor for the sensor to accurately detect the color yellow. This was done by having the brick display whether it detects yellow or not when the ball was in a different position relative to the color sensor. After determining the optimal ball position, the next procedure to test was the motion of the conveyor itself and ensure that the moveConveyor function correctly moved the conveyor the proper distance. Table 1 shows the total displacement of the ball-holding pegs from their previous position based on the distance travelled by the conveyor belt in the moveConveyor function, with a displacement of zero centimeters being the ideal expectation.

Table : Finding ideal conveyor belt movement distance

|  |  |
| --- | --- |
| **Distance Travelled by Conveyor (cm)** | **Displacement of Pegs (cm)** |
| 16 | 0.7 |
| 15.25 | 0.3cm |
| 15.4 | 0 |

After determining the distance of travel required for the conveyor and the sensitivity of the color sensor, both features were tested together to ensure the conveyor moves and stops at the correct position after yellow is detected by the sensor. The expected outcome was that the intake sequence could be run several times without the position of the pegs changing so that storage of the balls could be continuously repeated, and this was the outcome that was achieved.

Next, the flywheels were tested. What had to be done was to move the conveyor (using the moveConveyor function) and ensure the balls going through the conveyor could reach the flywheels and get launched. The expected outcome was that the pegs push the balls to the wheels, however the balls could not always reach the flywheels and so the conveyor belt would get jammed with more motion. The solution to this was not a software issue, but rather a physical issue, which was solved by bending the tray that supports the balls near the flywheels so the balls could roll into the flywheels.

Finally, after assembling the full robot, to test the accuracy of rotation and distance when driving to the center of the court, tape was carefully placed on the floor to mark specific distances (the high, normal, and low center points of the court the robot operates in). Then, the cornerToCenter function was run several times at all heights to ensure that the robot moved to the desired positions each time. Luckily, the desired outcome was achieved, with the robot accurately moving to the correct position, although with slow rotation and driving speeds.

All functions and processes were tested once more all together with the full structure of the robot to ensure that everything ran as needed.

## Significant Problem Solution

There were significant problems with the software, although the multiplexer caused some difficulties when running the shooting function. It was observed that the conveyor would not move instantly when needed in order to feed a ball to the flywheels. Since a solution to a hardware issue was not possible, implementing a longer wait time between the conveyor belt movements (five second wait time after turning a random angle) seemed to solve the problem.

# Verification

During the demo, the robot was able to meet most of the updated constraints that were set for it. Table 2 the constraints the robot was going to meet and whether it passed the constraint.

Table : Met and failed constraints

|  |  |
| --- | --- |
| **Constraints** | **Met?** |
| No drift when turning | yes |
| Robot can shoot 3 Tetrix balls every time | no |
| Smooth reading of the yellow Tetrix ball by the color sensor | no |
| No jams in the conveyor belt | yes |

One of the most important constraints the robot had to meet for the demo was to have little to no drift when rotating. During testing, there were many times when the robot went off its course due gyro drift and the drift accumulated over time as the robot is quite heavy. Also, the floor the robot was tested on affected how much it turned. There was more friction on the carpet, so the robot was more accurate on a smooth floor. To fix this issue, the robot was programmed to rotate slowly. This minimizes drift, meeting the constraint for the demo.

A constraint the robot had to meet but was unable to was to shoot 3 Tetrix balls every round. During the demo, the robot was only able to shoot 2 Tetrix balls instead of 3. This was because the color sensor detected the same yellow ball twice when one was being inputted. This was more of a human error that could have been solved by adding a greater wait time.

The other main constraint the robot had to meet was to have no conveyor belt jams. The battery used for the multiplexer was significantly weaker than the power supplied by the brick. This caused the conveyor belt to occasionally jam. To fix this, greater motor power was used. This still was not enough, so a large motor was used instead of a medium motor. The large motor and its high output power were enough to overcome the resistance from the battery connected to the multiplexer, so the robot was able to meet this constraint during the demo.

# Project Plan

## Splitting Up Tasks

Everyone in the group oversaw a certain aspect of the project. Ben and Louie mainly focused on the software while Ethan and Nadish mainly focused on the mechanical side of the robot. Ben and Louie created 4 functions together and the main code while Ethan and Nadish built the conveyor belt system and drivetrain. Although everyone had their own specific part of the robot to work on, everyone was helping each other when needed. Everyone in the group participated in the creation of the flywheels as this was the most challenging aspect of the robot to create as needed. Nadish and Ethan both created the ball height function and color sensor function. Also, Ethan and Louie built the walls of the court that held the Tetrix balls in, so they do not fly out.

## Scheduled Plan vs Actual Progress

As soon as the group was formed, the members immediately planned out a set schedule so all deadlines could be met along the way. The initial plan was to finish both the mechanical components and the software of the robot by November 19th. This was planned to ensure the group had enough time to improve or fix any aspects of the robot prior to the demo.

As the project continued, some issues arose. These issues consisted of both mechanical and software issues, but whenever there was an issue, this would affect both the software and mechanical components of the robot. Any difficulties that were met throughout the project were accounted for when the schedule was made by leaving enough time between the final deadline and the demo.

Table 3 and Table 4 are the two pre-planned schedules as well as the actual date of completion for the software and mechanical.

Table : Schedule and completion of mechanical components

|  |  |  |
| --- | --- | --- |
| **Mechanical Components and Assembly Schedule** | | |
| Part | Scheduled completion | Actual completion |
| Drivetrain | October 31st | October 30th |
| Motorized Drivetrain | November 4th | November 5th |
| Conveyor belt | November 9th | November 8th |
| Gear systems for flywheels & conveyor belt | November 11th | November 8th |
| Fly wheels system | November 17th | November 20th |
| Tray & Ramp | November 19th | November 24th |

Table : Schedule and completion of software

|  |  |  |
| --- | --- | --- |
| **Software Schedule** | | |
| Part of code | Scheduled completion | Actual completion |
| All 6 functions | November 10th | November 8th |
| Fix functions for Bugs | November 16th | November 18th |
| Main Program | November 19th | November 24th |

## Revisions to the Project Plan

Throughout the course of the project, there were many revisions that were made. They were mainly in the mechanical building of the robot due to some limited supplies and minor issues that created major issues. This caused the group to fall behind quickly in the pre-planned schedule. Some big changes that were made that caused the group to fall behind are listed below:

·        Making the conveyor belt shorter

·        Removing the intake system

·        Adding a gear system to the flywheels

·        Making the conveyor belt horizontal instead of angled

After the conveyor belt was built, it was noticed that the robot was going to be too big for the small 1.35-meter by 1.35-meter size court. The robot would have had trouble turning and the weight of the robot was too much to handle for the driving motors. The group then made the conveyor belt about half the original size and that solved a lot of problems. Overall, this issue caused a minor setback when trying to work with a massive robot.

One of the biggest issues was attaching the intake system to the conveyor belt. The intake system consisted of two little tracks that were each attached to a medium size motor. The two tracks were attached above by a few bars that kept the intake system stable. Although the intake system was stable, there was not much to attach the conveyor belt to. This caused the intake and conveyor belt system to be unstable when connected. The group spent significant time attempting to stabilize it but was unable to. So, the decision to remove the intake system altogether went through.

The flywheels turned out to be slow after some testing. To fix this, a 2-layer gear system was added to the flywheels. It took a lot of time to create, which delayed the part completion. Once complete and reinforced the flywheels were able to shoot the Tetrix ball with significant power.

Even after making the conveyor belt smaller, it was still quite large and heavy. The initial idea was to have the conveyor belt at an approximate 50-degree angle to the ground, but this proved to be unattainable. One issue is that there was not any way to support the conveyor belt to keep it angled with limited supplies. There was also no way to attach the driving wheels to the base of the conveyor belt system. Overall, these difficulties resulted in a lot of time spent on repairs, which caused a heavy setback on the whole project completion.

The robot’s purpose was to launch a Tetrix ball over a certain height on a wall, but since the robot was horizontal, it would shoot horizontally instead of at an angle. This defeated the main purpose of our robot. To fix this, a small cardboard ramp was attached in front of the robot. When the robot shoots the Tetrix ball, the ball hits the ramp and fly over the marked line on the wall. This took more time to construct than anticipated.

# Conclusion

When it comes to tennis training, costs remain high in the current world, forcing it to be inaccessible to the average human being. The Tennis Ball Training Machine is a solution to such a problem by replacing the constant need for coaches. Furthermore, the robot can conduct a variety of training regimes. While some criteria and constraints were not met, the overall robot is able to perform at an adequate level. Some important aspects of the mechanical side of the robot are its high-power launching system via a 2-level gear system and its precise conveyor belt working together with the flywheels. This ultimately ensures fast and rigorous training for the player. For the software side of the robot, the drivetrain is programmed to produce random angles and move to a different ball height every game, adding an element of variation to the player’s training. With such a small robot being able to produce such high-intensity training modules, it proves to be both affordable and convenient for those in need of tennis training.

# Recommendations

As aforementioned, in terms of the mechanical design of the robot, a separate intake system would prove more beneficial due to its lack of human interaction. Adding an intake system provides a means for the robot to be more efficient in its procedure without needing long wait periods before balls are inputted by humans. As for the software side, going hand in hand with the intake system, the robot could have swept the entire court and stored the balls on the floor on its own, completely removing the need for human interaction. If this project were used in the industry, coaches would not be required for every training session, reducing additional costs during training. The players themselves could set up consistent, yet intense training regimes.

# References

|  |  |
| --- | --- |
| [1] | B. E. Channel, "Spinning a Lego Wheel FASTER," 8 September 2018. [Online]. Available: https://www.youtube.com/watch?v=s3BsDF6UjCQ. [Accessed 3 December 2022]. |

# Appendix A – RobotC Code

#include "mindsensors-motormux.h"

//get number of rounds

int numRounds()

{

int rounds = 0;

while(!getButtonPress(buttonEnter))

{

displayString(2, "RIGHT: +1 Round");

displayString(3, "LEFT: -1 Round");

displayString(5, "ENTER to submit");

if(getButtonPress(buttonRight))

{

while(getButtonPress(buttonRight))

{}

rounds++;

}

if(rounds > 1)

{

if(getButtonPress(buttonLeft))

{

while(getButtonPress(buttonLeft))

{}

rounds--;

}

}

displayString(11, "Number of Rounds is: %i",rounds);

}

eraseDisplay();

displayString(7, "Rounds: %i", rounds);

wait1Msec(3000);

return rounds;

}

//Ball Height Input

float ballHeight()

{

eraseDisplay();

displayString(2, "Up - high ball");

displayString(4, "Middle - normal ball");

displayString(6, "Down - low ball");

int x = 0;

while(!getButtonPress(buttonUp) && !getButtonPress(buttonEnter) && !getButtonPress(buttonDown))

{}

while(getButtonPress(buttonUp) || getButtonPress(buttonEnter) || getButtonPress(buttonDown))

{

if(getButtonPress(buttonUp))

x = 1;

else if(getButtonPress(buttonEnter))

x = 2;

else if(getButtonPress(buttonDown))

x = 3;

}

if (x == 1)

{

displayString(8, "up was pressed");

eraseDisplay();

return 1.5;

}

else if (x == 2)

{

displayString(8, "enter was pressed");

eraseDisplay();

return 1;

}

else if (x == 3)

{

displayString(8, "down was pressed");

eraseDisplay();

return 0.5;

}

return 1;

eraseDisplay();

}

//rotate

void rotate(int motorPow, int angle)

{

resetGyro(S3);

wait1Msec(1000);

if(angle > 0)

{

motor[motorA] = -motorPow;

motor[motorD] = motorPow;

while(getGyroDegrees(S3) < angle)

{}

motor[motorA] = motor[motorD] = 0;

}

else if(angle < 0)

{

motor[motorA] = motorPow;

motor[motorD] = -motorPow;

while(getGyroDegrees(S3) > angle)

{}

motor[motorA] = motor[motorD] = 0;

}

motor[motorA] = motor[motorD] = 0;

}

//Drive Distance

void driveDist(float distance, int motorPow)

{

const float ENCODERTODIST = 180/(2.75\*PI);

float encoderNum = distance \* ENCODERTODIST;

nMotorEncoder[motorA] = 0;

motor[motorA] = motor[motorD] = motorPow;

while(nMotorEncoder[motorA]<encoderNum)

{}

motor[motorA] = motor[motorD] = 0;

}

//move conveyor

void moveConveyor(int motorPow)

{

const int BALL\_DIST = (15.4 \* 180)/(PI\*4.15);

MSMMotorEncoderReset(mmotor\_S2\_1);

MSMMotor(mmotor\_S2\_1, -motorPow);

while (MSMMotorEncoder(mmotor\_S2\_1) > -BALL\_DIST)

{}

MSMotorStop(mmotor\_S2\_1);

MSMMotorEncoderReset(mmotor\_S2\_1);

}

//intake sequence

void intakeSequence()

{

const int CAPACITY = 3;

int ballcount = 0;

const int INTAKEPOWER = 30;

displayString(1,"Intaking Balls");

while(ballcount < CAPACITY)

{

MSMMotorEncoderReset(mmotor\_S2\_1);

while(SensorValue[S4] != (int)colorYellow)

{}

moveConveyor(INTAKEPOWER);

ballcount++;

displayString(2, "%d", ballcount);

wait1Msec(2000);

}

}

//robot from corner to center function

void cornerToCenter(float length, float width)

{

resetGyro(S3);

driveDist(width/2, 25);

wait1Msec(1000);

rotate(10, 90);

wait1Msec(1000);

driveDist(length/2, 25);

wait1Msec(1000);

rotate(10, 90);

wait1Msec(1000);

}

//Random Angle Picker

int RandomAngle()

{

int listOfAngles[61];

for (int i = 0; i < 61; i++)

{

listOfAngles[i] = i - 30;

}

int randomNumber = random(60);

int randomAngle = listOfAngles[randomNumber];

displayString(2,"%d", randomAngle);

return randomAngle;

}

//shooting sequence

void shooting()

{

MSMMotorEncoderReset(mmotor\_S2\_1);

motor[motorB] = -100;

motor[motorC] = 100;

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

{

MSMMotorEncoderReset(mmotor\_S2\_1);

int chosenAngle = RandomAngle();

displayString(5, "%d", chosenAngle);

wait1Msec(1000);

rotate(15, chosenAngle);

wait1Msec(5000);

MSMMotorEncoderReset(mmotor\_S2\_1);

moveConveyor(30);

wait1Msec(3000);

MSMMotorEncoderReset(mmotor\_S2\_1);

rotate(15, -chosenAngle);

eraseDisplay();

}

motor[motorB] = motor[motorC] = 0;

}

//MAIN

task main()

{

SensorType[S2] = sensorI2CCustom;

MSMMUXinit();

SensorType[S3] = sensorEV3\_Gyro;

wait1Msec(50);

SensorMode[S3] = modeEV3Gyro\_Calibration;

wait1Msec(50);

SensorMode[S3] = modeEV3Gyro\_RateAndAngle;

wait1Msec(50);

SensorType[S4] = sensorEV3\_Color;

wait1Msec(50);

SensorMode[S4] = modeEV3Color\_Color;

wait1Msec(50);

int rounds = numRounds();

float height = ballHeight();

const int COURTDIMENSION = 135;

for (int currentRound = 0; currentRound < rounds; i++)

{

intakeSequence();

MSMMotorEncoderReset(mmotor\_S2\_1);

cornerToCenter(COURTDIMENSION ,COURTDIMENSION\*height);

MSMMotorEncoderReset(mmotor\_S2\_1);

shooting();

MSMMotorEncoderReset(mmotor\_S2\_1);

cornerToCenter(COURTDIMENSION, COURTDIMENSION\*height);

MSMMotorEncoderReset(mmotor\_S2\_1);

}

}

//END OF PROGRAM

# Appendix B – Software Flowchart

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated