Application Development Project

# Mini Soccer Penalty Kick Game

Tianpai Le, Yuehan She, Renkun Wang

AME 504 Mechatronics Systems Engineering

Viterbi School of Engineering

University of Southern California, Los Angeles, CA, 90007

# Table of Contents

**Introduction....4**

Project Idea....4

Context....4

Plan & Timelines....5

**Mechanical Design....6**

Overview....6

X-Translation Motion Design....7

Y-Rotation Motion Design....11

Z-Rotation Motion Design....14

**Electronics & Software Design....15**

**Manufacturing & Testing....20**

**Conclusion & Future Work....21**

**References....23**

# List of Figures & Tables

Table 1: Project Timeline.....5

Figure 1: Coordinate System Defined for the Project.....6

Figure 2: Linear Screw with Sliders and Sliding Rail (transparent).....7

Figure 3: MATLAB Application – Projectile Motion Launcher.....9

Table 2: Spring Comparison (Source: McMaster-Carr).....10

Figure 4 & 5: Y-Rotation Mechanism Design.....11

Figure 6: Geometry in the Y-rotation Mechanism.....12

Figure 7: MATLAB Application – Y-Rotation Geometry Calculation.....13

Figure 8 & 9: Z-Rotation Mechanism Design.....14

Figure 10: MATLAB Application: Projectile Motion Launcher.....15

Figure 11: Initial Loop Structure Design.....17

Figure 12: Improved Loop Structure.....18

Figure 13: Arduino Program Logic Block Diagram.....19

Figure 14 & 15: Product (2 views).....20

# Introduction

## Project Idea

The team plans to design and construct a mini soccer penalty kick game. The game will allow players to shoot the ball using a projectile launcher at the penalty spot towards the goal. There will also be a moving obstacle at the goal line acting as the goalkeeper. By controlling the power, horizontal angle, projectile angle, and activating the launcher at the right time, players can try their best to make the goal.

## Context

There are a lot of mini soccer games on the market, but not many of them can simulate the motion of shooting the ball as a projectile. Inspired by some popular slingshot video games (like Angry Birds), the team decided to bring it to real life. The whole setup will be multilayered and easy to play. The basic layout will have the goal area, shooting area and underground area for the mechanical and electrical components that are needed. The basic objective that we want to achieve is the player can control the shooting and the goalkeeper will move to stop the ball. In order to achieve that, we will first program the goalkeeper to move in left and right,up and down, four directions. Secondly, we will program the launch mechanism that allows the player to adjust the speed and direction.  Some other additional features we would like to add is the scoring system, by using an appropriate sensor and figure display, we are able to show the score when there is a goal.

## Plan & Timelines

The team's plan and timeline of the project is shown as follows (Table 1):

Table 1: Project Timeline

| **Timeline** | **Plan** |
|---|---|
| Week 1 (Mar 31~Apr 6) | ● Discuss the overall plan of the project<br>● Compare and select design ideas<br>● List items to be purchased |
| Week 2 (Apr 7~Apr 13) | ● Buy components<br>● Design and build X-translation mechanism components |
| Week 3 (Apr 14~Apr 20) | ● Design and build Z-rotation mechanism components<br>● Design and build Y-rotation mechanism components |
| Week 4 (Apr 21~Apr 27) | ● Assemble launcher components<br>● Write the Arduino code<br>● Run the Arduino code, and make adjustments according to feedback |
| Week 5 (Apr 28~May 1) | ● Final assembly with cases<br>● Run test<br>● Compile design files<br>● Work on presentation slides and report |

# Mechanical Design

## Overview

The mechanical components of the project are mostly integrated on the projectile launcher. Based on the design requirement of moving the launcher in three degrees of freedom (X-translation, Y-rotation, and Z-rotation, coordinate system defined by Figure 1), the primary goal would be forming three functional groups for each type of motion, and assembling them in a way that allows each group to move independently while minimizing size. This section will mainly focus on the the realization of all three motion mechanisms of the projectile launcher, with briefly going over other components of the product.

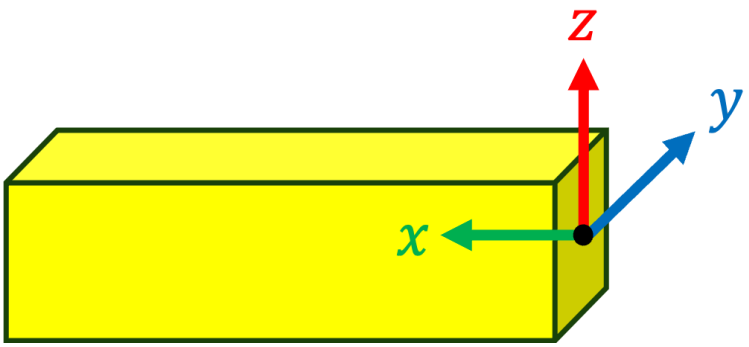


Figure 1: Coordinate System Defined for the Project

## X-Translation Motion Design

The X-translation motion would be responsible for controlling the launching force of the ball. Obviously using a compression spring would be an easy choice, for its linear relationship between force and distance, and easy visualization for players to sense the force changes. But that would lead to two questions: how to properly compress the spring, and what kind of spring to choose to meet our requirements.

To compress the spring, we need an object that moves linearly. From previous labs, there is one thing ready to use: the linear screw. However, we cannot attach the spring to the nut directly because space isn't allowed. But instead, we can let the linear screw act as a guide which leads the spring to compress through a connecting component. Therefore, we designed a pair of sliders (Figure 2). The rear slider attached to the nut translates the screw's motion to a distance below its own axis, and the front slider has a free end. When the front axis is blocked by the trigger, by driving the linear screw using a motor, the rear slider follows the nut and completes the compression.

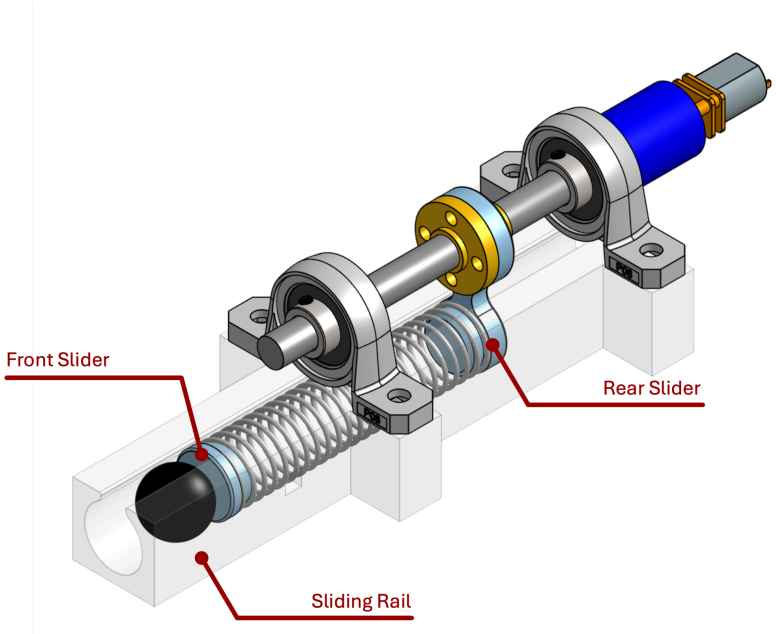


Figure 2: Linear Screw with Sliders and Sliding Rail (transparent)

Another concern is the compression spring tends to be unstable – it is susceptible to bending when compressed. So, it is important to restrict its motion. The solution is to design a U-shaped sliding rail (Figure 2) which houses both the spring and the sliders. Only a small portion on top is opened to ensure free movement of the rear slider. The rail also acts as the "barrel" of the ball. From subsequent testing it proves this design would guarantee the linear motion of the spring.

The next step is to choose the compression spring. The goal is to fit the spring's outside diameter with the ball's diameter as good as possible, and would allow for a reasonable shooting range. Based on energy equations and projectile motion analysis, we conducted the following calculations:

(1) The elastic potential energy provided by a spring is:

$$U = \frac{1}{2}k(\Delta L)^2$$

(2) Usually the fully compressed length of the spring will be provided by the manufacturer, therefore the difference between this length and the neutral length ($\Delta L$) would be:

$$\Delta L = L_n - L_c$$

(3) This difference can also expressed by the maximum load and spring rate:

$$\Delta L = \frac{F_{max}}{k}$$

(4) Substitute back into the the elastic potential energy equation:

$$U = \frac{(F_{max})^2}{2k}$$

(5) The energy will be converted to the kinetic energy of the ball with some reduction, assuming the conversion efficiency is 25%. From this equation, we can solve for the ball's initial velocity, $V$

$$K_E = \eta U = \frac{1}{2}mV^2$$

(6) Assuming the projectile angle from the ground is θ, we can finally calculate the ball position in X and Y directions:

$$x = x_0 + V_x t = x_0 + V\cos(\theta)t$$

$$y = y_0 + V_{y_0} t - \frac{1}{2}gt^2 = y_0 + V\sin(\theta)t - \frac{1}{2}gt^2$$

Using these equations above, we developed a MATLAB application in which we can enter spring properties (maximum load, spring rate) to generate a projectile motion map, which is shown in Figure 3.

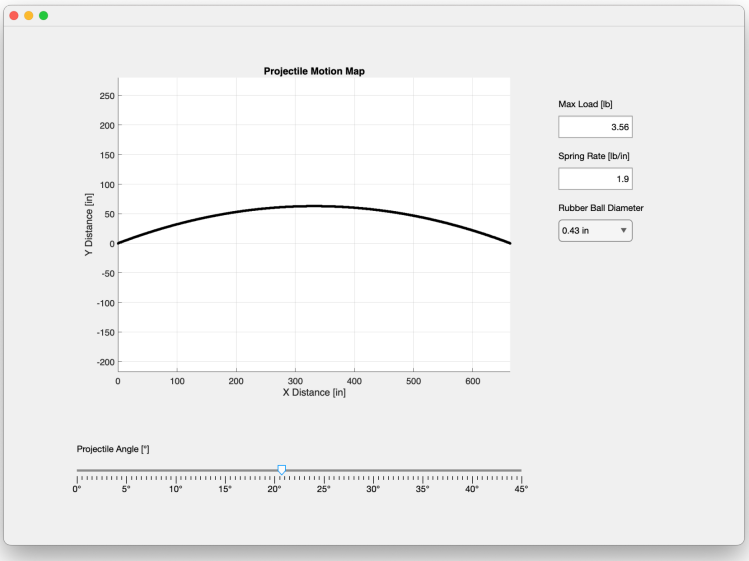


Figure 3: MATLAB Application – Projectile Motion Launcher

We targeted nine different types of compression springs and plugged all of their properties into a table, which is shown in Table 2 (next page), including the maximum shooting distance calculated by the MATLAB application.

Table 2: Spring Comparison (Source: McMaster-Carr)

| No. | Spring Name | Neutral Length [in] | Outside Diameter [in] | Max Load [lb] | Spring Rate [lb/in] | Adjustable Length [in] | Rubber Ball Diameter [in] | Max Distance @ 30° [in] |
|---|---|---|---|---|---|---|---|---|
| 1 | 9657K228 | 3.5 | 0.375 | 2.87 | 2.87 | 1 | 0.43 | 350-400 |
| 2 | 9657K226 | 3.5 | 0.438 | 4.77 | 3.25 | 1.47 | 0.4' | 900-1000 |
| 3 | 9657K115 | 3 | 0.5 | 6.15 | 3.77 | 1.63 | 0.5 | 800-900 |
| 4 | 9657K221 | 4.5 | 0.563 | 11.15 | 5.16 | 2.16 | 0.5 | 1600-1800 |
| 5 | 9657K126 | 3' | 0.593 | 7.49 | 4.59 | 1.631 | 0.5 | 1000-1100 |
| 6 | 9657K432 | 3 | 0.6 | 7.3 | 3.1 | 2.38 | 0.68 | 500-600 |
| 7 | 9657K449 | 3.5 | 0.6 | 7.3 | 2.7 | 2.79 | 0.68 | 600-700 |
| 8 | 9620K57 | 3 | 0.656 | 9.38 | 7 | 1.34 | 0.68 | 400-450 |
| 9 | 9657K124 | 3 | 0.688 | 3.56 | 1.9 | 1.871 | 0.68 | 200-220 |

From Table 2, it turns out most springs have maximum shooting ranges far beyond our expectations. So, we would like to choose a spring which gives relatively shorter shooting distance, while having larger compression length to make the force changes easier visible to users. Taking both into consideration, we finally chose spring No.9 for its being relatively soft and having reasonable compression length.

After plugging back this spring into the MATLAB application, we decided to set the maximum lifting angle (Y) of the launcher between 10 and 15 degrees, in which case the maximum projectile height of the ball would be less than 20 centimeters.

## Y-Rotation Motion Design

The Y-rotation motion, i.e. the rotation of the launcher about a preset y-axis, is responsible for setting the elevation angle of the launcher. Figures 4 & 5 show our design idea for controlling Y-Rotation. Our major design concept can be summarized as a process of energy transition from the rotational motion controlled by the motor (colored in purple in Figure 4 & 5), to the translational motion of the rollers (The non-transparent small circular-shaped components in Figure 5) along the track (what covers the rollers in Figure 4), then to the rotational motion of the top board where the launcher will be attached. In short, the motor will drive the rollers up and down and push the launcher-attachment top board upward. With one end of the top board fixed using a hinge, the board will move in a circular motion, elevating the launcher up and down. To limit the size of the final product, we chose to limit the elevation angle to below 10˚, as this limits the distance and the height that the ball will travel so we don't have to make the field and the goal too big.

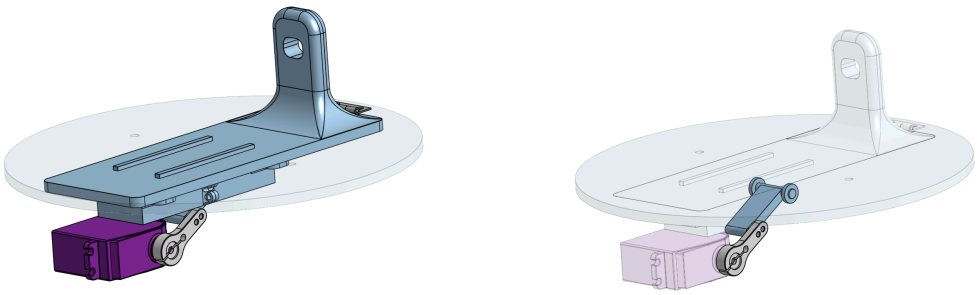

Figure 4 & 5: Y-Rotation Mechanism Design

We adopted this design idea mainly for considerations of size and weight-balance. With the motor supplying rotational energy and with the board desired to move in rotational motion, the simplest way to design the rotation control is to use the motor to directly drive the rotation of the board at the current location of the hinge. However, if the Y-rotation motion is designed this way, it will firstly cause the size of the circular board (The transparent circular component in Figures 4 & 5) that holds the top board to significantly increase as it needs to hold a large and heavy servo motor, and secondly lead to the imbalance of weight as the motor needs to be attached to one side of the board for this design to work, which can tip over the entire launcher system with its

weight. Our current design sets the position of the motor towards the center, which effectively solves these problems.

Based on this design idea, we conducted some geometric calculations to solve for the dimensions of the entire mechanism that could reach our desired range of launching angles. Similar to the projectile motion along the X-direction, we will derive some equations based on the geometry shown in Figure 6.

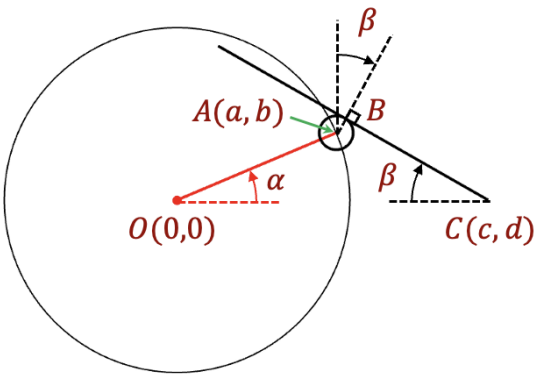


Figure 6: Geometry in the Y-rotation Mechanism

As illustrated in Figure 6, the origin (point O) represents the pivot of the servo's rotating shaft, point A is the center of the roller, point B is the tangential point between the roller and the track (represented by the think black line), and point C is the fixed connection point (where the hinge is located). The angle α is the lifting angle of the shaft, and the angle β is the actual lifting angle of the launcher.

Here the only fixed parameters are the origin, the full length of the track, and the diameter of the roller (denoted as $r$). Other parameters including points C, linkage length (thick red line, denoted as $L$), and the offset value of the angle α, are all free to design within some geometric constraints. Our end goal is to derive an expression for the angle β and optimize its range. The full derivation is shown on the next page.

(1) Express the coordinates of point A

$$a = Lcos(\alpha)$$

$$b = Lsin(\alpha)$$

(2) Express the coordinates of point B

$$B_x = a + rsin(\beta)$$

$$B_y = b + rcos(\beta)$$

(3) Solve for the angle β using the following equation

$$tan(\beta) = \frac{B_y - d}{c - B_x}$$

Where point C coordinates $(c,\ d)$ will be treated as an input

This set of equations above enabled us to make another application on MATLAB (Figure 7). Apart from the geometric traits mentioned above, we included the peripheral dimensions of the servo, the width of the rotating shaft (left small circle), and invalid track length (because this region is blocked by the Z-rotation drivebar, painted in green). They would become the additional limits of setting up the dimensions.

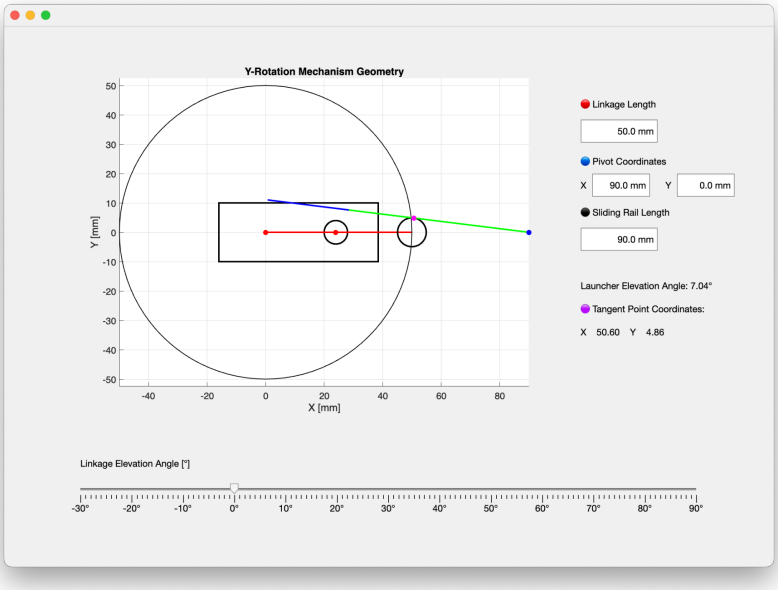


Figure 7: MATLAB Application – Y-Rotation Geometry Calculation

## Z-Rotation Motion Design

The Z-rotation motion is responsible for setting the horizontal shooting angle of the launcher, i.e. determining whether the launcher is shooting left or right. Figures 8 & 9 show our design idea for controlling Z-Rotation. Our design idea has a very straightforward design concept, where a servo motor (in Figures 8 & 9, colored in purple) will be attached at the center of the circular rotation board and the motor will directly control the rotation. The angular motion is limited to the range between -15˚ to 15˚ with 0˚ pointing straight towards the goal. This is to make sure that the ball is not shooting too wide so that we do not have to make the goal too wide or make the field too short.

To deal with the track installed underneath the rectangular board for achieving Y-rotation, an upside-down Ω-shaped bracket goes around the track and connects the motor and the circular rotation board. This ensures that the Y-rotation and Z-rotation do not interfere with each other.

Another design consideration worth mentioning is that the motor mount (The part attached under the motor in Figure 8) has a wide base. This can work to prevent the entire launcher system from tipping over due to the weight on the edge of the circular board. An alternative solution to this problem would be to add a supportive part under the edge of the circular board. However, this solution is not ideal because it will reduce the smoothness of Z-rotation due to the existence of friction.

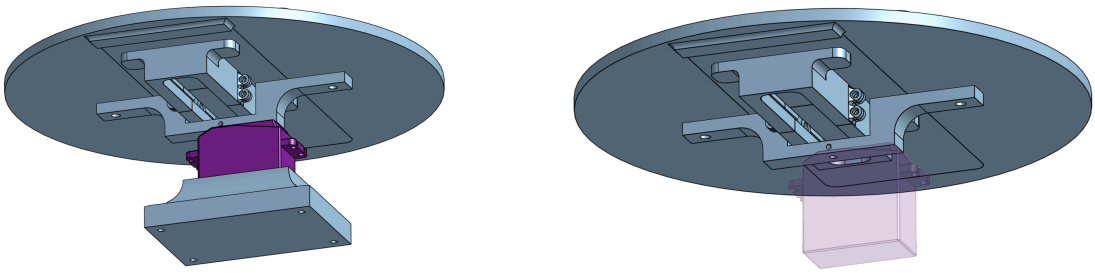

Figure 8 & 9: Z-Rotation Mechanism Design

# Electronics & Software Design

There are two main goals in the electronics and software design of this project: (1) Allow the system to perform all three types of motions (2) Allow users to control the system by remote control.

The full circuit schematic is shown in Figure 10. It consists of four motors: one DC motor for the motion in the X-direction, and three servos for motions in the Y and Z directions plus the goalkeeper. An L293D motor driver is used to drive the DC motor. In addition, an IR receiver is placed to receive the signal from the remote control.

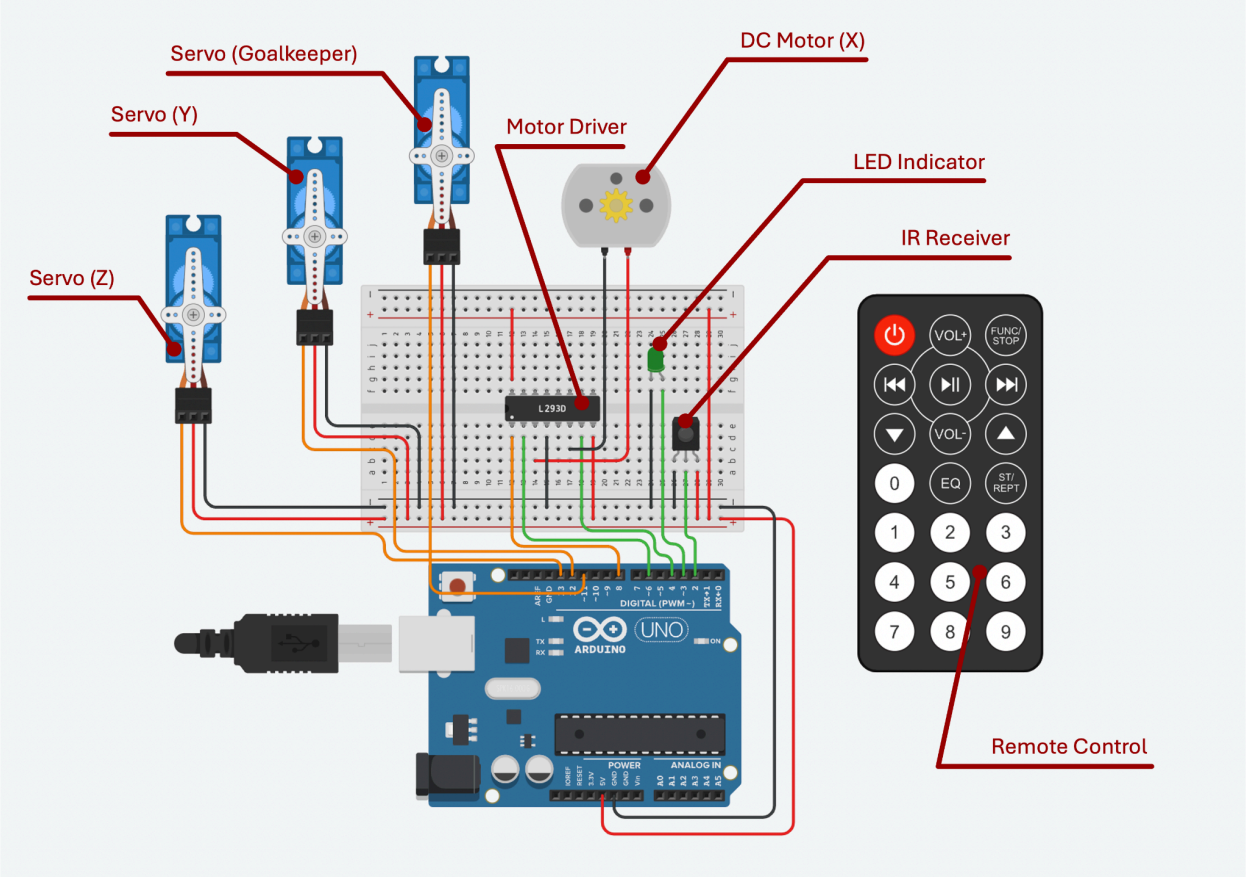


Figure 10: MATLAB Application: Projectile Motion Launcher

After building the circuit, the next step is to set up the control logic. The first thing to solve is to obtain the real-time positions of all four motors, because we want each one of them to move back and forth within a designated range (for motion in the X-direction it would be a distance, and for other motions it would be angles). Among them, servos are relatively easier to deal with, because they have built-in feedback control systems which can read positions themselves. Therefore, the only thing we need to do is to give position commands, and they will automatically turn to the desired positions.

However, such functionality doesn't apply to the DC motor. So, some additional effort needs to be made to compensate for that. One option is to set up an open-loop control system by designating time limits. For example, if the motor pushes the indicator forward by 5,000 milliseconds, it should also move back for 5,000 milliseconds after launching the ball to return the indicator to its original position. However, since the motor would compress the spring in one direction and recover in the other, the actual time of moving the indicator along the same distance in these two directions would vary and cause a time gap. This gap would keep expanding play by play, and eventually resulting in inaccuracy.

Another possible option is to use microswitches. Although it is still unable to obtain the real-time positions of the indicator, it can set up absolute position limits, thus preventing the motor from continuing running in the same direction after touching the boundary. But the problem is they don't fit our current design, for the space occupied by their lever arms would reduce the valid compression length of the spring.

Finally, we decided to offer the control fully in the hands of the user, which means the user would control the turning direction of the motor using two buttons, and the absolute position of the indicator will not be recorded. It is not a perfect approach, since the indicator won't be able to return to its original position after each play. But it can still be considered reliable for the limited time and would be a solid stepstone for future improvements.

The second problem to address is to allow real-time stop of each motion and move forward to the next step. For example, when the launcher is rotating in the Z-direction, it should be able to stop any time when the user presses the "pause" button, and immediately switch to rotation in the Y-direction. One important thing in this problem is the delay() function cannot be used, because the entire system will stop running during delay and no command will be read by the controller. The alternative is Arduino's built-in timekeeper within the millis() function [1], which not only keeps time but refreshes itself every time being called. However, even when we use millis() instead of delay(), there is still one remaining obstacle: when the system is engaging in one motion, it will be an infinite loop until the user presses the "pause" button (Figure 11). This would never work with the goalkeeper – because it needs to operate all the time, no matter if a button has been pressed (except the "power" button which is designed to open/shut down the entire system). Obviously, we couldn't stay in this loop without doing anything else. Even if we don't activate the goalkeeper, we still need to constantly check all button states in each motion loop apart from the main function, and repeating this process will result in a huge downgrade in both reaction time and program efficiency.

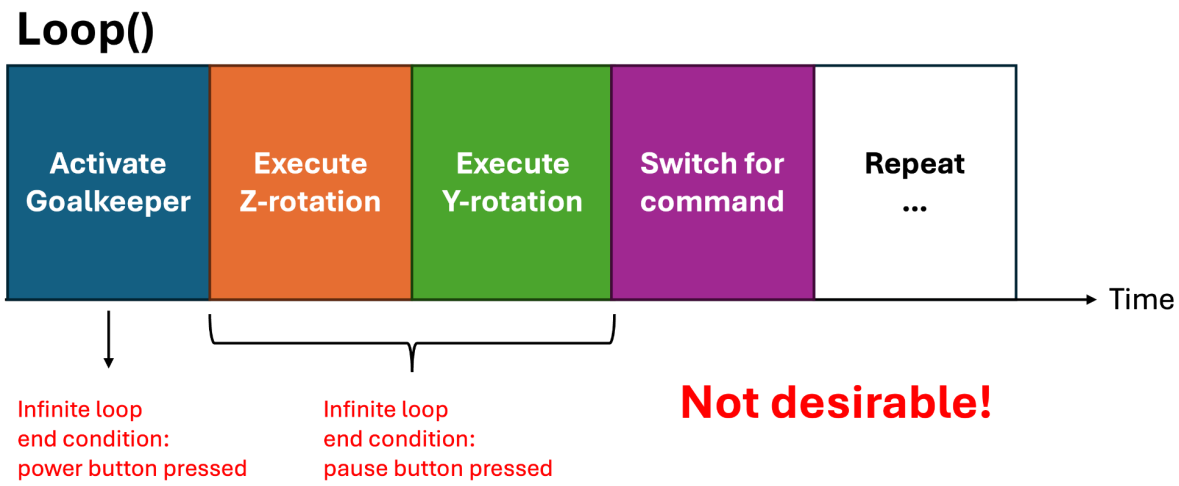


Figure 11: Initial Loop Structure Design

An alternative approach is to limit the duration of each motion to a very short time. It will lead to those functions executed by turn. But if the entire loop goes fast enough, this discontinuity will be almost impossible to visualize, and from the user's perspective everything is executed simultaneously. In fact, this method has been applied to a variety of programs much more complicated than this project. For this project, we choose to set the time limit of each motion to 10 milliseconds (Figure 12), which is still a considerable amount of time compared to the rotation speed of the servos (which is 0.15~0.2 seconds per 60 degrees). Thus, the improved loop structure using this method is shown as follows.

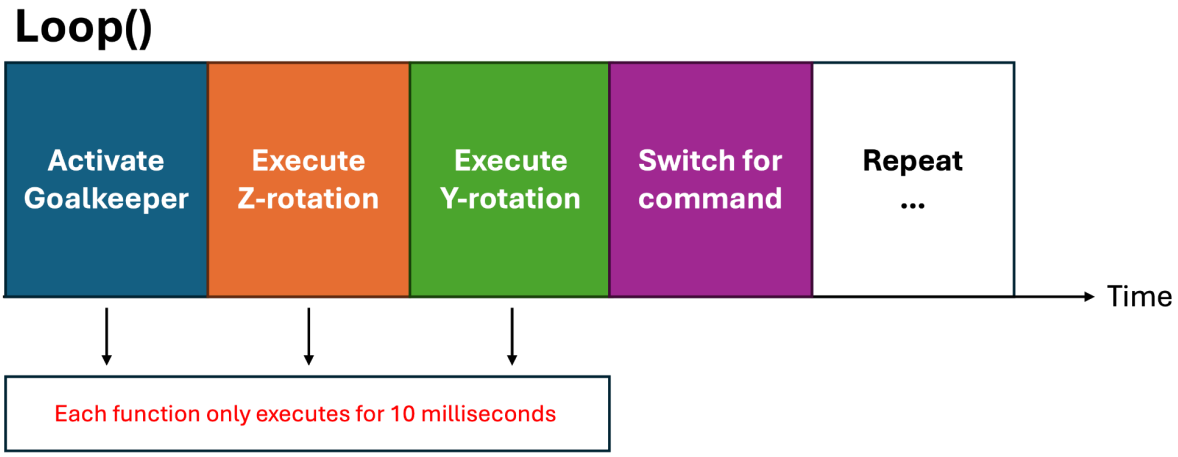


Figure 12: Improved Loop Structure

The last piece of code would be setting up the remote control and controlling the servo. Fortunately both of them are included in Arduino libraries, which are <IRremote.h> and <Servo.h>, respectively [2][3]. Especially for <Servo.h>, it can automatically transform the input angles into PWM. So, instead of looking for the servo's technical properties and doing mathematical operations to obtain PWM, we can just input the position after targeting the correct servo, which would save a lot of time.

Therefore, the complete logic of the Arduino program can be constructed in the block diagram as shown in Figure 13. And the process can be described as follows:

(1) The entire system is started through the "power" button, with the goalkeeper and Z-rotation motion activated, goalkeeper speed can be adjusted by number buttons "1"~"3".
(2) When the "pause" button is pressed for the first time, Z-rotation motion stops and Y-rotation motion starts.
(3) Pressing the "pause" button for a second time will stop Y-rotation motion, and the motor for X-translation will start to push the indicator forward and compress the spring.
(4) During this process, the user can press "up" and "down" buttons to move the indicator in either direction, and press the "pause" button to stop the motor at any time.
(5) The user will manually pull the trigger to launch the ball.
(6) Whenever the "power" button is pressed when the system is running, it will shut down immediately, and pressing it again will restart the system and return to step (1).

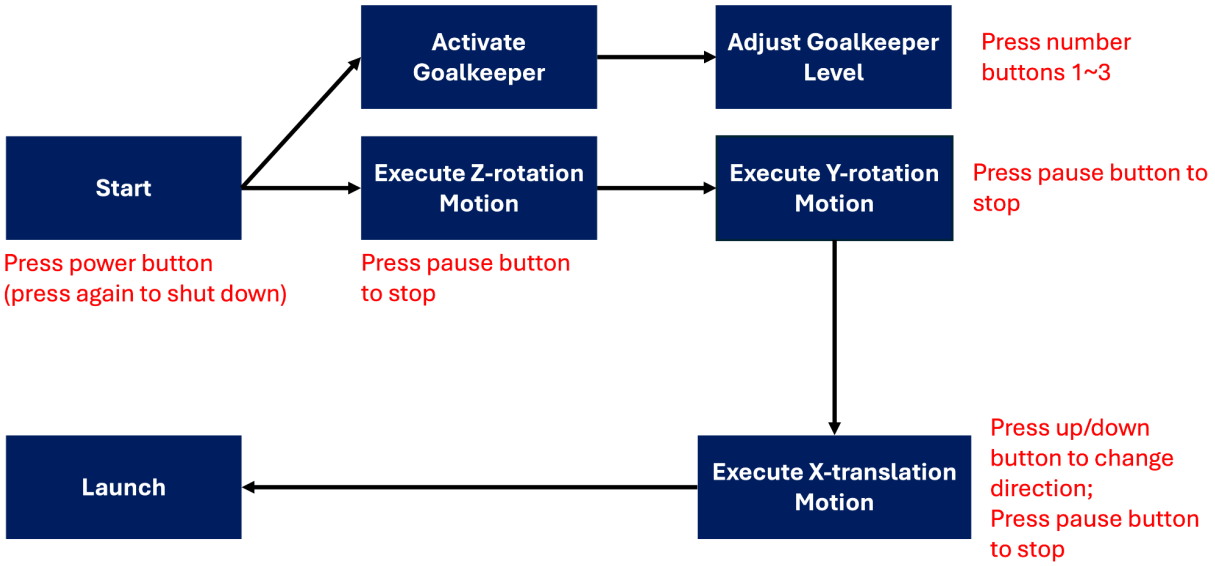


Figure 13: Arduino Program Logic Block Diagram

# Manufacturing & Testing

Most of the parts in our project are made by our own, we use 3D printing to manufacture the launcher part, except for the X-translation mechanism. X-translation is achieved by linear screw, however, the front slider, rare slider and slider rail are designed and manufactured by ourselves. All of the Y-rotation and Z-rotation mechanism parts are 3D printed, due to the inaccuracies of printing, we still need to sand all the parts in order to perfectly assemble. The outside cases and holding blocks are made by wood and cut by ourselves using the saw in the lab. The motor mounts are also 3D printed and use the drill to make holes for the wires to go through. The fully assembled product is shown in Figure 14 and 15.

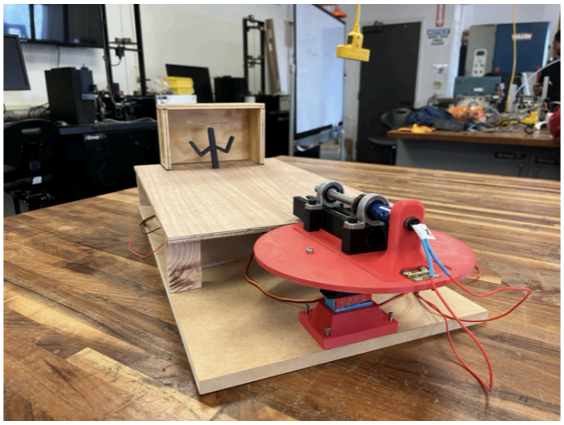

Figure 14 & 15: Product (2 views)

During our testing, several problems were encountered. First of all, Z-rotations are not stable, this is because the servo motor was directly connected to the bracket, after a few tests, the screw became loose and the whole system fell down. By adding a support and servo horns, we successfully fixed the problems. The second problem is the Y-rotation mechanism. After a few tests, we found that the motion is stuck since the roller track is not smooth enough. The reason is that the tracks are printed separately and the meeting point has a tiny height difference, after sanding two parts, this is easily solved. The last problem we face during testing is the goalkeeper. During the test, the goalkeeper did not move as we expected. After diagnostics, we found out that the battery power is not enough to power three motors simultaneously, so we have added another battery as a temporary fix.

# Conclusion & Future Work

Overall, there are a lot of positives to take from this project. In terms of mechanical design, each type of motion of the launcher is well constrained, with the compression spring being prevented from bending, and the roller and track design worked out perfectly for rotation around the Y-axis. Meanwhile, despite not being super compact, the launcher is still limited to a reasonably small size. Most importantly, all the functionalities of the launcher have been achieved, which proved our design concept is reasonable enough to meet the requirements.

We also learned a lot on the calculation and software side. Results from testing showed that our calculation in both spring choice and geometric dimensions for Y-rotation have been verified. It also proved that choosing numbers from calculation would be much more effective than randomly assigning them, especially for those open-ended design cases like what we have encountered. Coming to software, we gained a lot of knowledge about relating Arduino programs with our existing components, including controlling servos by relating their angular positions with PWM, setting up IR receiver modules, and enabling remote control using button commands. With the help of improved loop structures and built-in libraries, we successfully reduced the numbers of both wire connections and lines of code.

For the potential future work in this project, we are looking into some improvements to current functions. There are three potential improvements we have identified that are insignificant to the overall function but would make the product perfect if implemented. Firstly, there is an approximately 5% chance that the goalkeeper does not start automatically when the game is turned on. This is caused by the low quality of the motor that drives the movement of the goalkeeper and we will update this part with a better motor to solve this problem. Secondly, there's still an insignificant weight imbalance occurring for the launcher. Potential solutions to this problem include making the battery mount base wider, choosing a lower-weight motor with a similar level of performance capability as our currently used ones for the launcher rotation and elevation, etc. Lastly, whenever we turn off the entire system, the launcher elevation will not return to a desired, perfectly flat position but remain at a slightly elevated angle. We figured out that this issue is caused by the fact that the Arduino servo code library we use to control the

motors has a preset default position that cannot be changed by the user side. Therefore, to improve on this, we can either spend time experimenting to calibrate the equilibrium position in the preset or test on a different servo motor library that allows us to control the default position by ourselves.

We are also looking into adding potential new functions for future work to make our product more advanced. Firstly, we are looking into an electronic shooting trigger that allows the customer to shoot the ball without hand-pulling the trigger. We chose a hand-pulling trigger for our final product as, based on our experiment, the hand-pulling trigger is easier to control and harder to damage. However, with the entire system being electronically controlled and the trigger being the only exception, we would still like to search for potential options to make this happen. Secondly, we are looking into adding an auto-scoring system by adding an IR sensor at the goalline and LCD boards to show the score. This will make this game more competitive and fun. Lastly, we are thinking about a ball-returning system that allows the ball to be returned to a location conveniently reachable for the players. This can be fully structural without the need for electronic systems, just like those in most table hockey game sets.

# References

[1] Arduino.cc, “millis(),” [Online]. Available: https://www.arduino.cc/reference/en/language/functions/time/millis/.

[2] Arduino.cc, “Servo,” [Online].Available: https://www.arduino.cc/reference/en/libraries/servo/.

[3] Arduino.cc, “IRremote,” [Online]. Available: https://www.arduino.cc/reference/en/libraries/irremote/.