EMG Controlled Ball Throwing Mechanism — Design Documentation

Jaime Thrower 20971801
Jamie Kang 20956456

University of Waterloo
BME 261 Prototyping, Simulation and Design
EMG Controlled Mechanism Final Report

Submitted on 2024–08–06

I. Purpose of the Mechanism

The purpose of the mechanism is to throw a ball, which weighs approximately 100g, at a speed of 50km/h. This innovative device aims to assist forearm amputees in playing catch. The mechanism is designed to be triggered by the user flexing their arm, which activates the system to throw the ball that is placed in a basket attached to the mechanism. This allows amputees to participate in the enjoyable and social activity of playing catch, enhancing their recreational opportunities and quality of life.

II. Early Design Ideas

To be able to throw the ball at the desired speed, several mechanism ideas were considered in the early design phase. The mechanism would need to run on a single motor, and could not be too expensive. Although we had a feeling that we would not be able to throw the ball at the speed we wanted due to those two constraints, we aimed to get as close to the goal as possible. As suggested by the professor, a quick return (QR) mechanism was researched first. A QR mechanism involves a slider-crank interaction to transfer rotatory motion into transitional motion [1]. However, the QR mechanism was quickly deemed unsatisfactory for the following two reasons. First, the QR mechanism requires a gear to turn in one-direction, stopping before each quick-movement before activiation and restoration. The servo motors available to us were either continuous motors that were blind to its own angular position, or motors that were constrained to 180 degrees of motion. Second, the mechanism does not produce enough kinetic energy for the ball to be thrown anywhere near our desired speed. Thus, It was decided that some version of spring-loaded catapult mechanism would be better for our purpose, as the spring would be able to store enough potential energy which can be transferred to the ball as kinetic energy.

Two mechanisms were discussed for the release method of the spring-loaded catapult. Sketches drawn during brainstorming at this phase can be found in Section A of the appendix. At first, a drop cam method was brought up where the catapult would fire when the cam reached its drop point. This idea was quickly dropped due to the same reason as the QR mechanism, as it required the gear to be one-directional and angular position aware. The best method we found was a mechanism dubbed as the “choo-choo mechanism”. It is composed of two arms and a pusher; as the gear rotates, the pusher winds up the mechanism by pushing down on the smaller arm. The mechanism activates when the pusher crosses the angle in which the small arm is able to snap to the other side of the pusher, releasing all the potential energy stored in an instant. This mechanism works because it can be activated for both directions of rotation; which allows us to use the 180 degree servo motor to control activation as long as proper gear ratio is determined. The choo-choo mechanism is well explained in the diagram found in Section B of the appendix.

III. Material and Parts Selection

After the selection of the choo-choo mechanism design, the components were initially selected and tested with the assumption that our design could throw a 100g ball at the average women’s softball throwing speed of around 50 km/h. The most important main components that needed to be decided upon included the material of the throwing arm / the supporting columns, the springs, and the gears. The left side of Fig. 1 shows the initial dimensions chosen for the design; the spring attachment point of 3cm from the end of the throwing arm is later adjusted to 5cm.

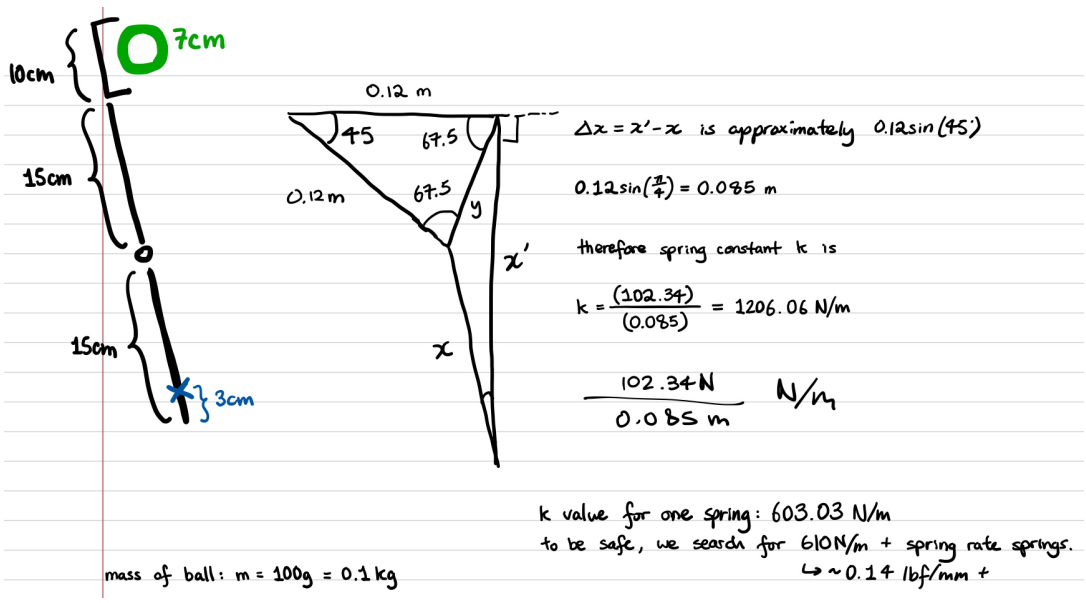

Fig. 1: A simple diagram of the initial dimensions, as well as the spring constant calculations using initial dimensions and force calcualted in Fig. 2.

The throwing arm's biggest constraint was that it needed to withstand the bending force from the gears stretching the springs. The material of choice to be tested was a SPF (spruce-pine-fir) wood plank with cross-sectional area of 0.75 inches by 1.5 inches, which was an easy source of acquisition from the local hardware store. Using the SolidWorks Simulation feature, we performed finite element analysis (FEA) using material properties of SPF wood according to [2]. The throwing arm was fixed at the pivot point and the attachment point of the choo-choo arm; a load of 120N was applied downwards from the spring attachment point. The load applied was slightly more than the calculated value from the desired speed in which we wanted to throw the ball, as can be seen in Fig. 2. This was to compensate for the change in spring attachment point mentioned earlier. Fortunately, as can be seen in Fig. 3, the results indicate that the SPF wood plank can be safely used with a factor of safety of 18.

mass of ball: $m = 100g = 0.1\,kg$ ↳ ~0.14 lbf/mm +

initial speed of ball: $v_0 = 50km/h = \frac{125}{9}\,m/s = 13.89\,m/s$

initial kinetic energy of ball: $KE = \frac{1}{2}mv_0^2 = \frac{1}{2}(0.1)\left(\frac{125}{9}\right)^2 = 9.645\,J$

potential energy stored in the spring: $E_{spring} = \frac{1}{2}kx^2$

torque exerted by the spring: $T = F_{spring}\,r = F_{spring}(0.12)$
where $r = 0.12m$ (distance from pivot to spring attachment point)

work done by spring as it moves through an angle $\theta = 45° = \frac{\pi}{4}$

$W = T \cdot \theta$

$(9.645\,J) = F_{spring}(0.12\,m)\left(\frac{\pi}{4}\,rad\right)$

$\Rightarrow F_{spring} = \frac{(9.645\,J)}{(0.12\,m)\left(\frac{\pi}{4}\,rad\right)} = 102.34\,N$

$\Rightarrow T_{spring} = F_{spring} \cdot r = (102.34)(0.12) = 12.28\,N \cdot m$

The choo choo mechanism is attached 10cm (0.10m) from the pivot point.
To balance the torque produced by the spring, the force $F_{choo\ choo}$ at 10cm must create an equal and opposite torque

$T_{choo\ choo} = F_{choo\ choo} \cdot 0.10 = 12.28\,N \cdot m$

Fig. 2: Calculation of the force provided by the spring at maximum deflection (when the throwing arm is parallel to the base plate).

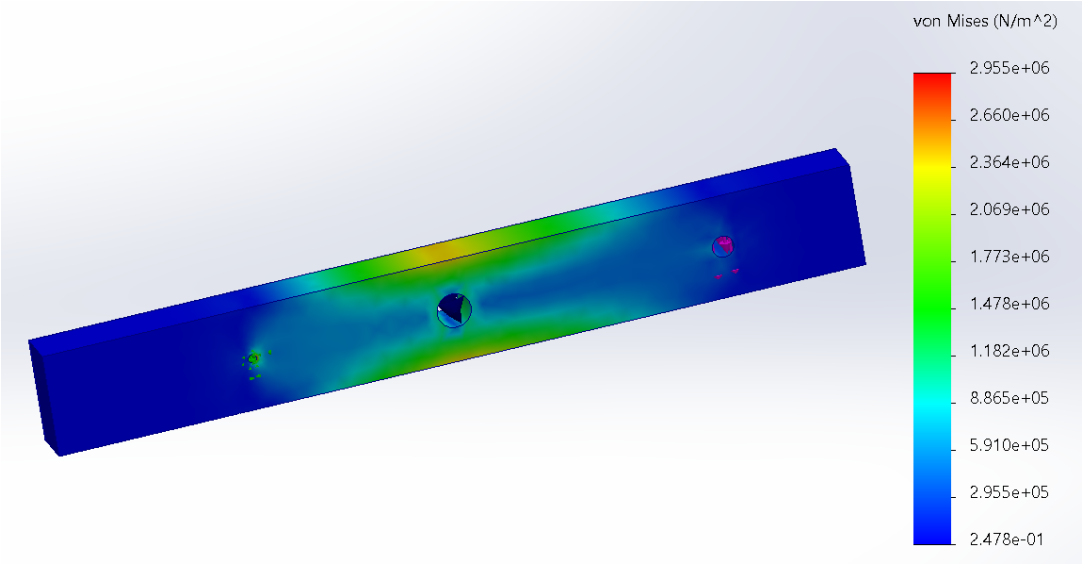

Fig. 3: Screenshot of the von Mises stress results of FEA on the throwing arm.

Next, the gears were chosen. The choo-choo mechanism requires the driven gear (the gear directly connected to the short choo-choo arm and the pusher) to have a 360 degree range of motion running CCW, and about 10–15 degrees more CW (due to the non-perpendicular direction of pull from the throwing arm). Since the servo motors available only had a range of 180 degrees, it was important that our gear ratio was could at least double the range. We also wanted to avoid printing our own gears, as we were advised not to do so during class. After searching for gears that were in our budget, we chose a set of plastic gears with a gear ratio of 8/3. The driving gear directly coupled to the motor has 96 teeth, and the driven gear which

is connected to the choo-choo mechanism has 36 teeth.

With the gear ratio now fixed and the dimensions of the gears known, we could finally calculate the torque that would be required for the motor to power our machine. First, the positions of the components were sketched out in SolidWorks. Each component was later made in context of this sketch. Using the Equations feature, the lengths of each component were related to calculate the various mechanical values. The sketches and equations can be found in Section C of the appendix. Using an initial spring constant of 1220 N/m (calculated from the estimated dimensions in Fig. 1), we checked if it was possible to throw at the speed we desired with the motors availabe to us. Unfortunately, it turned out that for our desired speed the motor would need roughly a maximum torque of 4.24 N·m. The maximum torque that the servo motors available could provide was 2.42 N·m, so we had to lower our standards quite a bit. We found a set of springs of which their combined spring constant is around 630 N·m, which had the suitable unstretched length, and extension range for our machine. This brought the maximum torque required from the motor to 2.19 N·m, and the speed of the ball at launch around 24.9 km/h. Although we were disappointed that it was half the speed that we were aiming for, we decided that this speed would be sufficient for the throwing machine.

The arms of the choo-choo machine were 3D printed, however we did not run any FEA prior to printing due to the complexity of the loading situation. It was also very cheap to 3D print, thus we decided to test in real life instead. When tested during the fabrication process, the 3D printed arms posed zero issues, and the first copy of the arms successfully have made it onto the final product.

Clevis pins were strategically chosen to connect the choo choo mechanism arms together because they facilitate quick and easy assembly and disassembly. This is particularly beneficial during the prototyping and testing phases, where frequent adjustments are necessary to perfect the design. The clevis pins provided the required rotational freedom for the choo choo mechanism, ensuring the frictionless operation of the arm and gear system. Furthermore, the strength and durability of steel clevis pins ensure they can withstand the mechanical stresses and repeated use inherent in a catapult. Their secure connections were reinforced by cotter pins, which prevented unintentional disassembly, ensuring the reliability and safety of the mechanism. 3D printed spacers and washers made space between the choo-choo mechanism arms so that the cotter pins did not impede the rotation of the arms.

Bearings were added to the rotating joint so that friction could be avoided as much as possible during the launching, so that all of the potential energy from the springs could be transferred to the ball. The rotating joint itself was made with a steel rod to support the force from the springs. The springs were connected to the throwing arm via a threaded rod, in which the rings at the ends were hooked onto the ends of the rod and the rope guide attached to the base.

The base was made out of wood to utilize wood screws which would make fixing our machine to the base easier. Casings around the supporting columns were designed to be 3D printed do prevent the columns from collapsing and moving about. The gear holder and the motor casing was also designed to be 3D printed. The gear holder was specially designed so that the gears would be at the correct location during assembly, by having a long base that touches the supporting columns. This way, by aligning the columns and the long base during assembly, no measuring would be necessary to position the gears properly. The gears themselves were mounted onto the gear holder via sholder bolts, so that the gears could rotate properly.

Finally the ball basket to hold the ball was designed with as many holes as possible to reduce weight so that it had little effect on the throwing motion. All of the holes in the 3D printable parts designed were for a no. 6 wood screw, which we decided we would use to secure the parts together.

The final CAD assembly can be found in Section D of the appendix.

IV. Fabrication Process

The fabrication of the mechanism involved several precise steps, beginning with the preparation of the main catapult arm, which was made from 1.5”x0.75”x12’ SPF wood. A 30cm long piece was cut from this lumber using a vertical saw, and three

holes were drilled: a 5/16” hole for securing a 2.25” long steel threaded rod with hex nuts to hold the springs, a 1/2” hole for four 3/16” inside diameter bearings that aligned with the supporting columns, and a 1/8” hole for a clevis pin that connected to the choo choo arm. The supporting columns, made from 1.5”x0.75”x12’ SPF wood, were cut to 16cm long, and each had a 1/2” hole drilled to house two bearings. A 3” long, 3/16” diameter steel rod passed through these bearings to create the arm’s rotation point. It was then secured from falling out by the 3D-printed casing, which also served to stabilize the columns near the top. The base, made from a 12”x12”x0.75” particle board, was drilled to accommodate the attachment of support columns using angle brackets and to screw the gear holder onto the base using #6 wood screws. The choo choo mechanism’s long and short arms, as well as the contact rod and ball basket, were 3D printed from PLA. The short choo choo arm was joined to the long arm and small gear using clevis pins of specific lengths, with the driven gear requiring additional holes for these pins. The 96-tooth driving gear was coupled to the motor with M3 bolts and nuts. Holes were drilled as accurately as possible so that they aligned properly with the preexisting holes in the motor plate. Rope guides and springs were attached under the catapult, with the springs being placed on the threaded rod. Assembly was finalized by securing clevis pins with cotter pins and using washers to reduce friction between PLA and wood parts. All components were sourced from vendors such as Home Depot, Amazon, McMaster-Carr, and EMS Store, with detailed specifications ensuring a precise and functional build.

Along the process, we came across a few blocks. One problem faced in the testing phase was when our 3D-printed gear holder was not sturdy enough to keep the gears in place under the stress of the mechanism. As the catapult arm was drawn back and tension from the springs was applied, the gears were pulled apart and could no longer mesh together. This issue prevented the mechanism from reaching its actuation point. Although we fixed this issue by redesigning and printing a sturdier gear holder (which is what can be seen in Fig. 9), it subsequently broke when tested with two springs. This was due to the nature of 3D printing; the method in which the gear holder was printed layer by layer made it weak to tensile force normal to the layers. This problem was fixed by filling the cracks in the PLA with epoxy. At first this was done only in little amounts, but due to consequent breaks in layers where epoxy was not applied, we decided to strengthen the entire part by pouring epoxy inside the part itself, fully saturating the mostly hollow inner section. This worked, but to be safe for the demonstration we decided at this point to use an elastic band instead of the springs to first show off the functionality of the catapult. The catapult was activated with the springs only at the end of the demonstration.

Another small problem was that the cotter pin was not possible to be used for the attachment of the small choo-choo arm and the pusher to the small gear. Someone had mistakenly taken our bag of cotter pins, and there was no possible way to get them back before the deadline. A replacement was quickly thought out; a shoulder bolt and a heat sink nut was used in place to secure the clevis pin in place. This worked perfectly, and it remained secure just as effectively as the cotter pins.

Photos of the fully assembled machine can be found in Section E of the appendix.

V. Power Systems

The motor was powered by connecting it to a wall outlet using a USB wall adapter. A USB cable from the adapter provided the necessary power, with the red line (5V power) connected directly to the motor. The black line (ground) was connected to both the motor and the Arduino’s ground pin. This setup ensured that the motor received a stable power supply from the wall outlet, while maintaining a common ground with the Arduino for proper operation and control.

The circuit was powered by connecting the Arduino’s 5V power pin to the breadboard’s power rail, ensuring a stable and regulated power supply for the components. The Arduino UNO itself was powered through a USB cable connected to a laptop, which provided the necessary voltage and current for the system. This setup allowed for easy and reliable power delivery during the development and testing phases, enabling smooth operation of the catapult mechanism.

VI. Code Design and Arduino Integration

The Arduino code is designed to control the servo motor based on an input signal from electrodes attached to the bicep, which indicates muscle flexing. This signal, digitized via a comparator, is read from pin 2 using digitalRead. The purpose of the code is to move a servo motor through three specific angles whenever the bicep flex signal is HIGH. The first and second movement triggers the choo-choo mechanism to activate the catapult which then throws the ball. The first activation triggers the mechaism in the clockwise direction, and the second activation triggers in the counterclockwise direction. The servo then resets to the initial position on the final activation and waits for the next signal to repeat the cycle.

The code begins by including the Servo library and creating a servo object. It defines several constants and variables, including the pin connected to the signal from the electrodes (`signalPin`), a debounce delay to avoid false triggering (`debounceDelay`), and an array of positions (`posArray`) that the servo will iterate through. The setup function initializes serial communication, sets the signal pin as an input, attaches the servo to pin 9, and moves the servo to its initial position of 10 degrees.

In the loop function, the code uses a flag (`wasLow`) to track if the signal pin was LOW for the required duration. If this flag is true and the signal pin has been HIGH for at least the debounce delay, the servo moves from the current position to the next position in the posArray. The position index is incremented and wrapped around using a modulo operation. The code then prints the current position value to the serial monitor and resets the wasLow flag. If the wasLow flag is false and the signal pin has been LOW for the debounce delay, the flag is set to true.

The code includes two helper functions to check if the signal pin is HIGH or LOW for the specified duration. These functions use a while loop to monitor the pin state and record the start time when the pin first goes HIGH or LOW. If the pin remains in the same state for the duration of the debounce delay, the function returns true. Otherwise, it returns false.

The moveServo function is responsible for moving the servo from one position to another. It takes two arguments, fromPos and toPos, and moves the servo in increments of 1 degree. If the current position is less than the target position, the servo moves forward; otherwise, it moves backward. During the movement, the function prints progress to the serial monitor and includes a short delay to ensure smooth motion. After the movement is complete, it prints the final positions to the serial monitor.

The full code can be found in Section F of the appendix.

VII. Final Product

The final product encountered issues with the initial springs due to the 3D-printed gear holder's insufficient strength. Consequently, we substituted the springs with a rubber band for activating the mechanism with the motor. While this adjustment allowed the mechanism to work perfectly and launch the ball, the ball was thrown slower than initially intended. When manually pulling back the catapult with the springs, the ball was launched at the desired speed, demonstrating the potential of the design. The use of the elastic band provided reliable operation, although it resulted in a slower ball launch compared to the original goal.

Although activating the mechanism with real EMG signals had the potential to work, as demonstrated during tests on another classmate, their reliability and reproducibility were inconsistent. This inconsistency was likely due to the low-quality EMG instruments available, which did not provide accurate and reliable readings across different users. Additionally, factors such as muscle fatigue, changes in skin conditions, variations in electrode placement, and fluctuations in the user's physical state or stress levels could further affect the EMG signals. This variability meant that the system could not be guaranteed to function correctly under all conditions.

To ensure the success and dependability of the term project, especially during the final oral exam, we opted to use a fake EMG signal. This decision prevented any potential issues or compromises that could arise from the unpredictable nature of

real EMG signals. Using a simulated EMG signal provided a controlled and stable input, ensuring that the mechanism would reliably trigger the ball-throwing action during demonstrations, thereby effectively showcasing the project's functionality and objectives.

References

[1] R. P. Podhorodeski, S. B. Nokleby, and J. D. Wittchen, "Quick-return mechanism design and analysis projects," *International Journal of Mechanical Engineering Education*, vol. 32, pp. 100–114, 2 2004. DOI: 10.7227/ijmee.32.2.2.

[2] MakeItFrom.com, *Spruce-pine-fir (spf, softwood)*, Accessed: 2024-07-19. [Online]. Available: https://www.makeitfrom.com/material-properties/Spruce-Pine-Fir-SPF-Softwood.

[3] Apalrd, *Apalrd's choo-choo analysis spreadsheet*, Accessed: 2024-07-19. [Online]. Available: https://www.chiefdelphi.com/t/paper-apalrds-choo-choo-analysis-spreadsheet/146469.

A Brainstorming Sketches

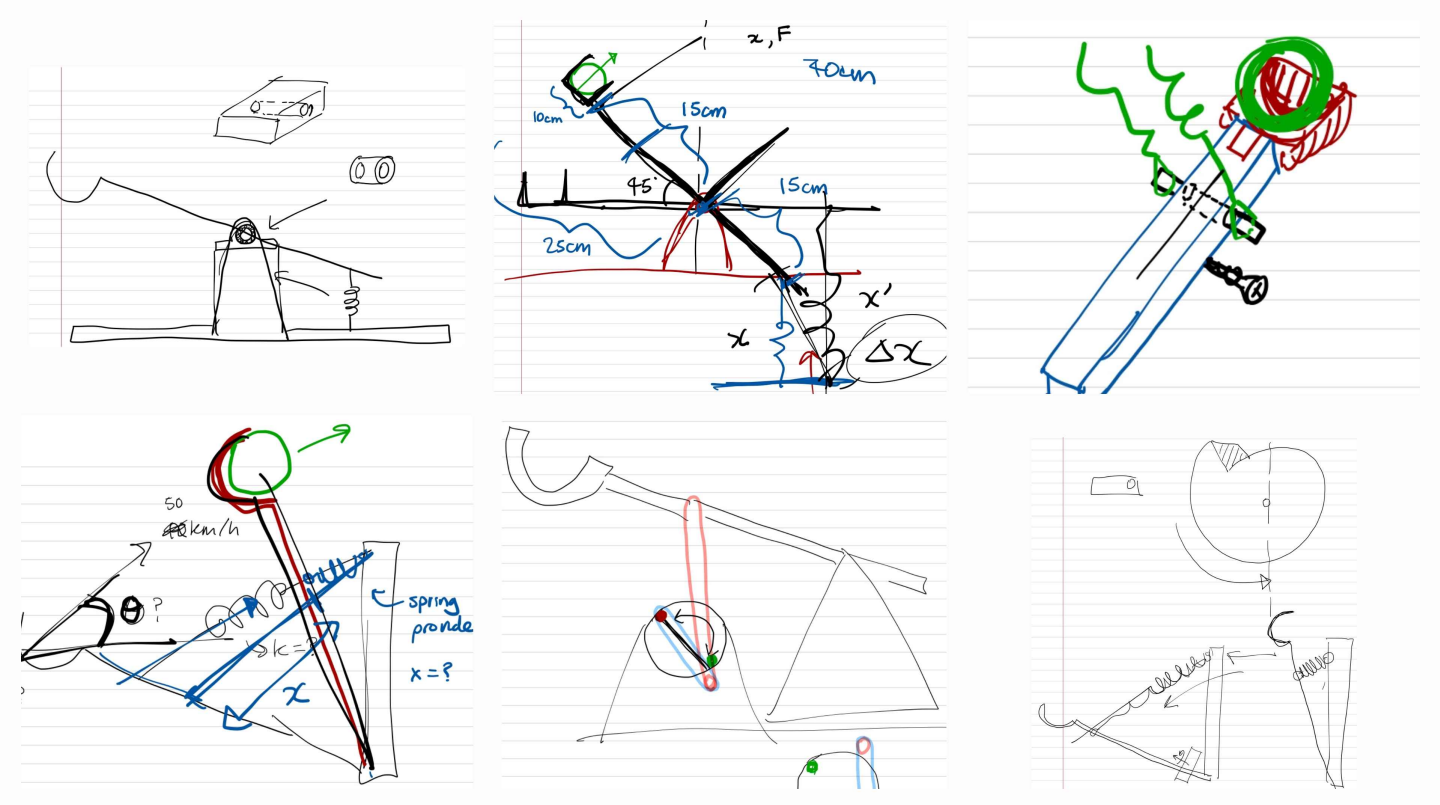

Fig. 4: A collage of sketches from early brainstorming phase.

Fig. 4 shows some of the most early ideas for our throwing mechanism. The bottom right figure shows our thought with using cams as a possible mechanism, while some other drawings such as bottom left and top right shows different spring placement considerations.

B Choo-choo Mechanism Diagram

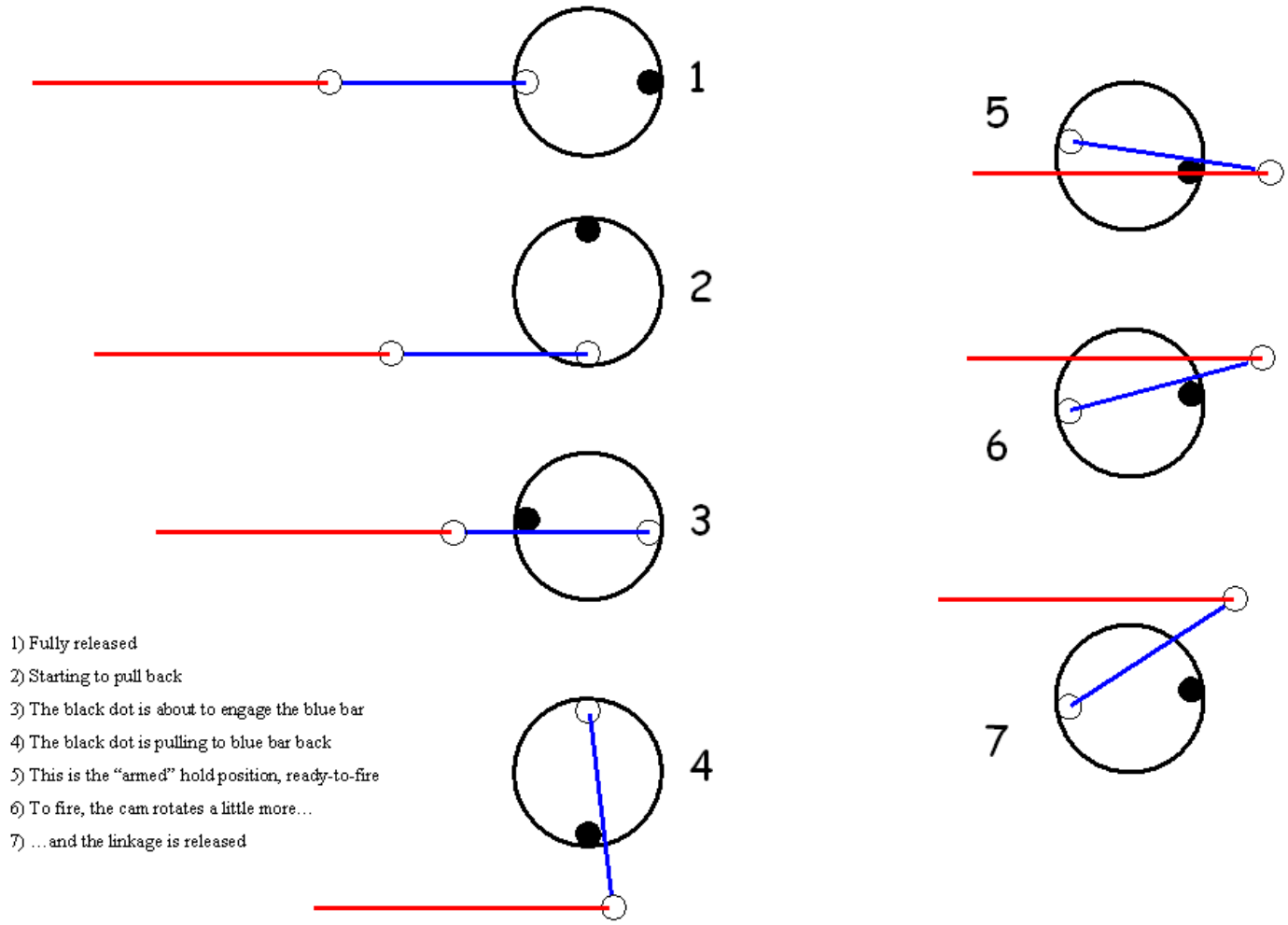

Fig. 5: A graphic of the step-by-step process of the activation of the choo-choo mechanism. Source: [3]

The choo-choo mechanism was first used in Winnovation 2010, then was popularized by Team JVN in the 2014 Build Blitz [3].

C SolidWorks Dimensions and Calculations

All values in Figs. 6 and 7 are shown in meters.

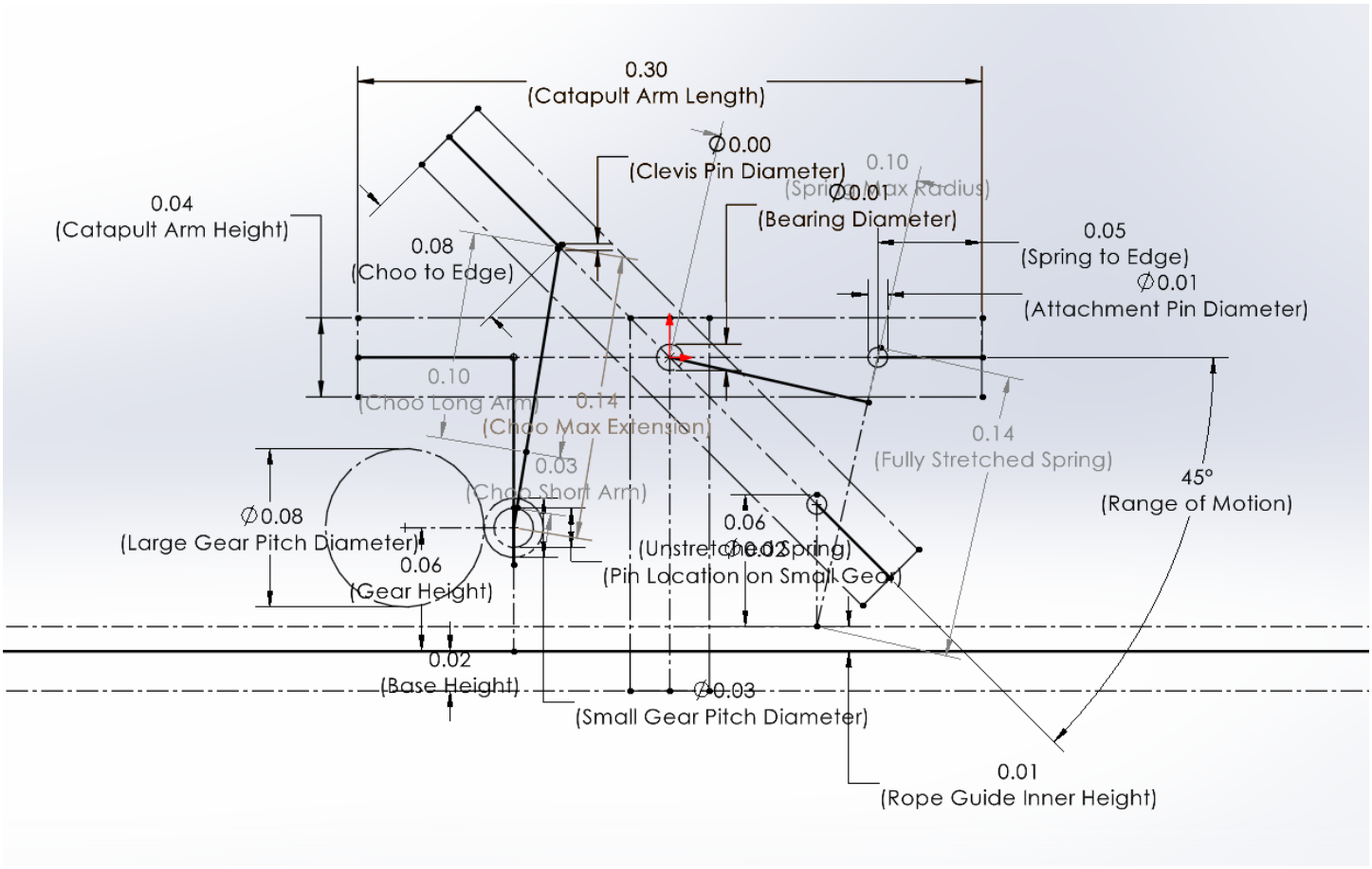

Fig. 6: Sketch "Dimensions" on Right Plane of CAD Model Assembly.

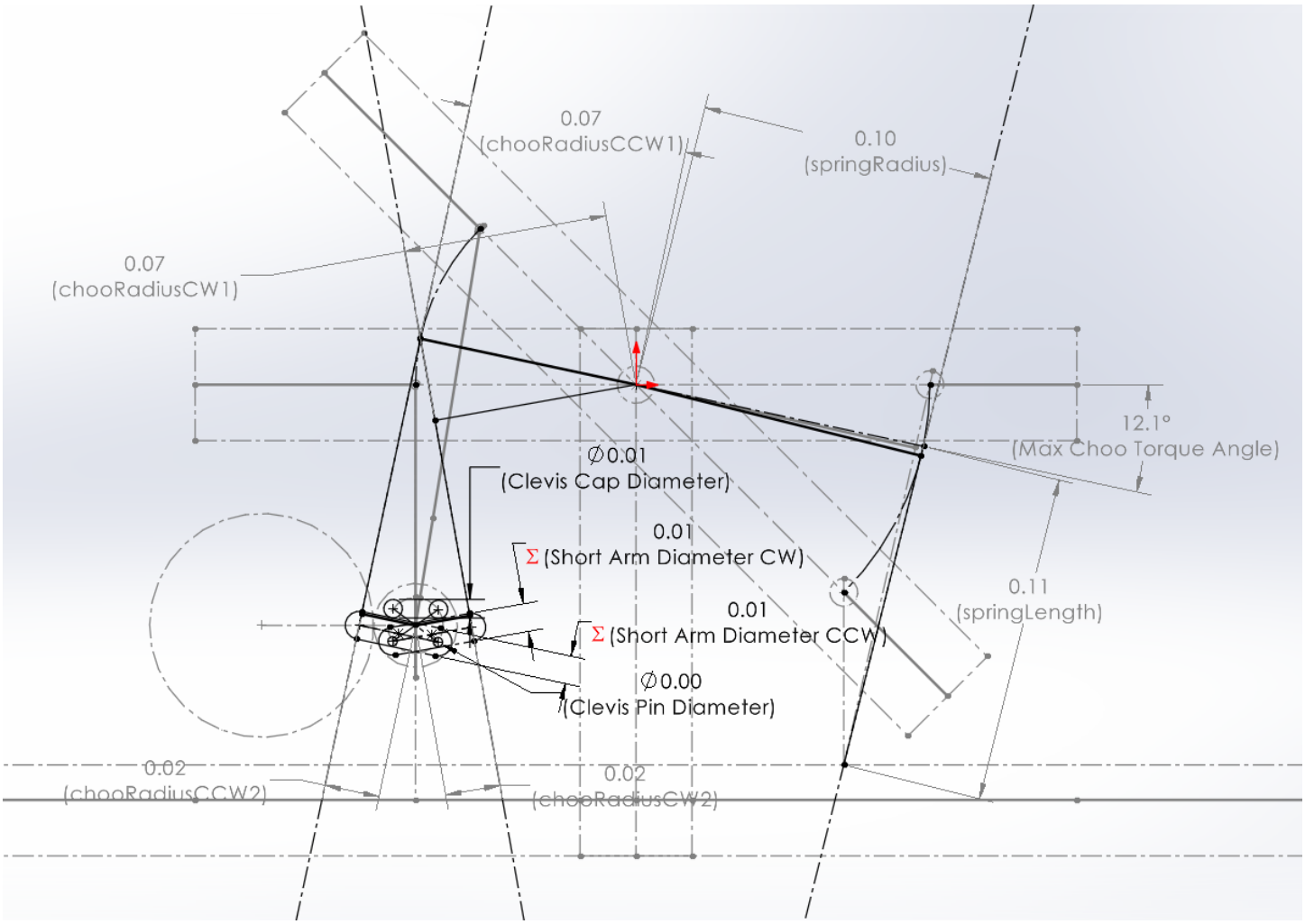

Fig. 7: Sketch "Calculations" on Right Plane of CAD Model Assembly.

```
"springRateNewtons"= ( 2 * 4.4482216153 * 1.8 ) / 1in
"springExtension"= "springLength@Calculations" - "Unstretched Spring@Dimensions"
"springForce"= "springRateNewtons" * "springExtension"
"springTorque"= "springForce" * "springRadius@Calculations"
"Short Arm Diameter CCW@Calculations"=3 * "Clevis Pin Diameter@Calculations"
"chooTorqueMain"= "springTorque"
"chooForceCCW"= "chooTorqueMain" / "chooRadiusCCW1@Calculations"
"chooTorqueDrivenCCW"= "chooForceCCW" * "chooRadiusCCW2@Calculations"
"chooTorqueDriverCCW"= "chooTorqueDrivenCCW" * ( 96 / 36 )
"Short Arm Diameter CW@Calculations"=3 * "Clevis Pin Diameter@Calculations"
"chooForceCW"= "chooTorqueMain" / "chooRadiusCW1@Calculations"
"chooTorqueDrivenCW"= "chooForceCW" * "chooRadiusCW2@Calculations"
"chooTorqueDriverCW"= "chooTorqueDrivenCW" * ( 96 / 36 )
"springMaxExtension"= "Fully Stretched Spring@Dimensions" - "Unstretched Spring@Dimensions"
"springMaxForce"= "springRateNewtons" * "springMaxExtension"
"springMaxTorque"= "springMaxForce" * "Spring Max Radius@Dimensions"
"maxChooTorqueAngle"= "Max Choo Torque Angle@Calculations"
```

Fig. 8: SolidWorks Equations, relating variables from Figs. 6 and 7 to determine various mechanical values.

D Finalized Design

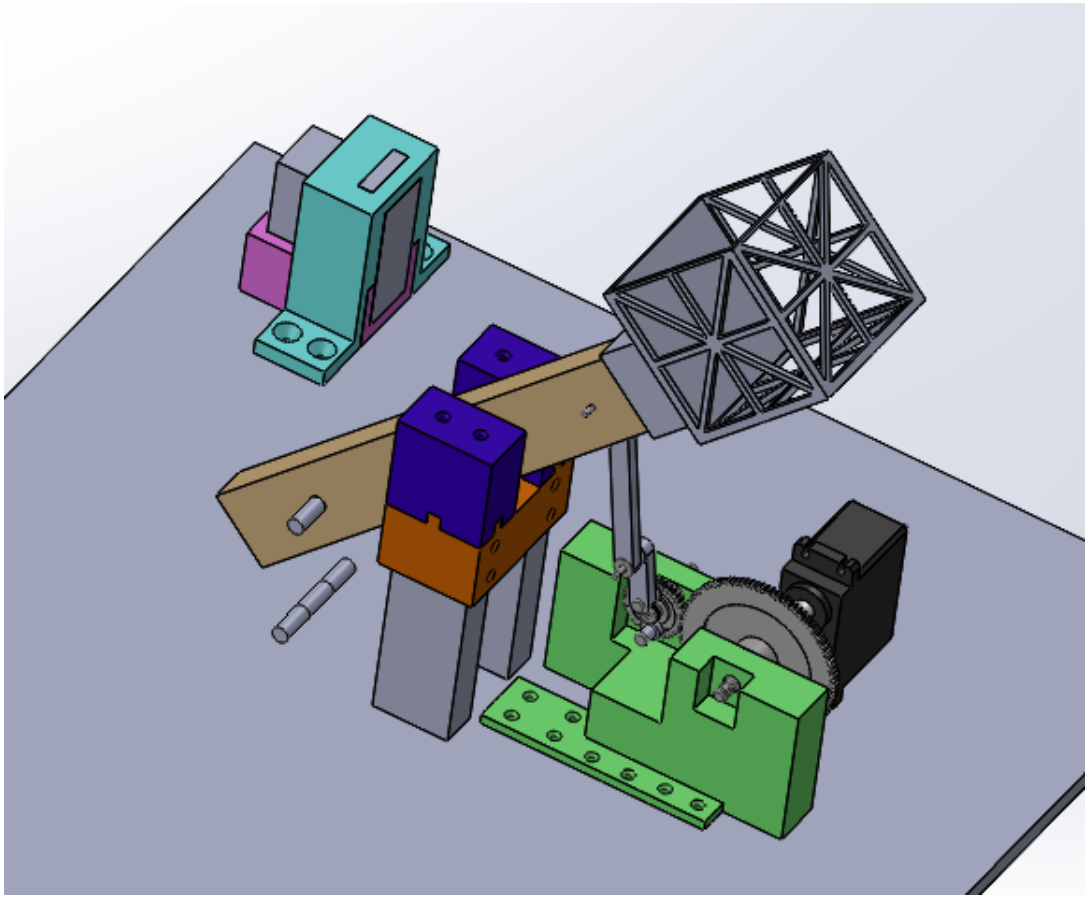

Fig. 9: A screenshot of the final CAD assembly, ready to be printed and assembled.

The final CAD of the finished design can be found in Fig. 9. The motor casing is located at the corner, as the CAD model found for our model of servo motor had incorrect dimensions and required a simplier model to be made ourselves. In the actual assembly, this casing is in the proper location around the motor.

E Pictures of the Finished Machine

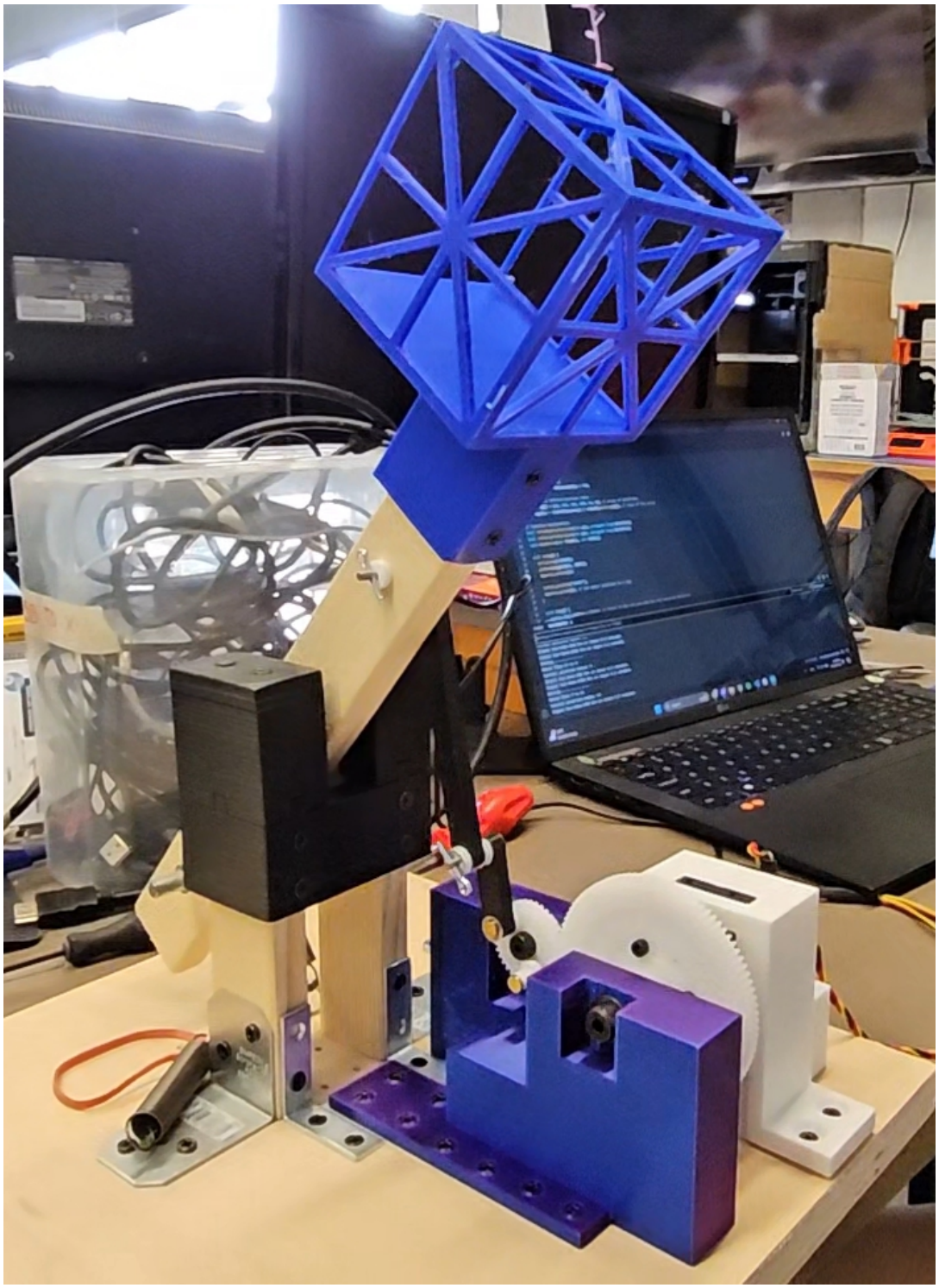

Fig. 10: A picture of the machine, after being assembled fully.

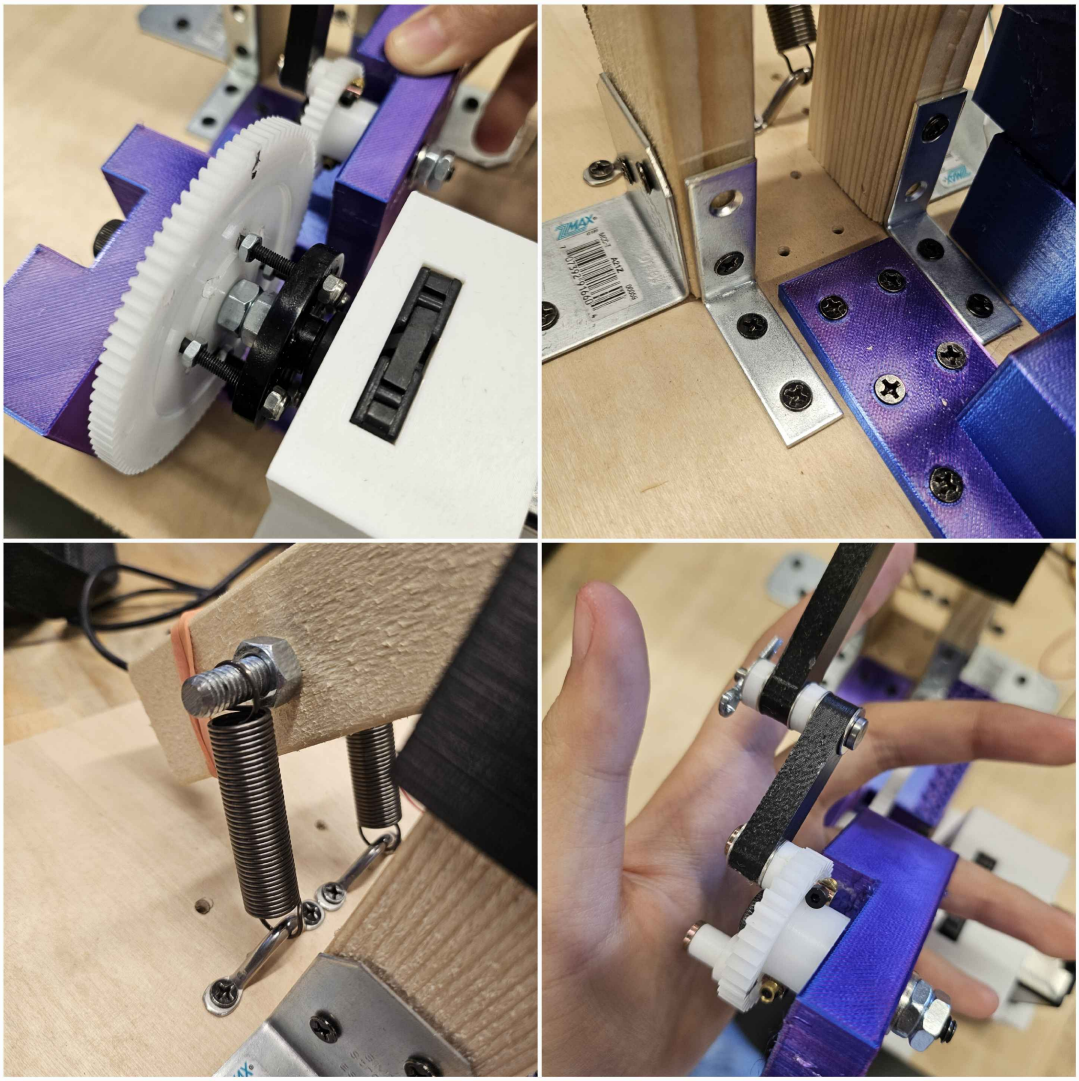

Fig. 11: A collage of close-up pictures of the various components of the machine.

F Full Arduino Code

```
#include <Servo.h>
Servo myservo;

const int signalPin = 2;
const unsigned long debounceDelay = 500;

int pos = 0; // Initial position index
int posArray[] = {10, 135, 0}; // Array of positions
int arraySize = sizeof(posArray) / sizeof(posArray[0]); // Size of the array

// Function Declarations
bool isPinHighForDuration(int pin, unsigned long duration);
bool isPinLowForDuration(int pin, unsigned long duration);
void moveServo(int fromPos, int toPos);
```

```
void setup() {
    Serial.begin(9600);
    pinMode(signalPin, INPUT);
    myservo.attach(9);

    Serial.println("START");
    myservo.write(posArray[0]); // starting position of 10 deg
}

void loop() {
    static bool wasLow = false; // Track if the pin was LOW for the required duration
    static int fromPos = 0;
    static int toPos = 0;

    if (wasLow && isPinHighForDuration(signalPin, debounceDelay)) {
    fromPos = posArray[pos];
    pos = (pos + 1) % arraySize; // Move to the next position in the array and wrap around
    toPos = posArray[pos];
    moveServo(fromPos, toPos);
    Serial.print("Current position value: ");
    Serial.println(posArray[pos]);
    wasLow = false; // Reset the flag after incrementing the position
    }

    if (!wasLow && isPinLowForDuration(signalPin, debounceDelay)) {
    wasLow = true; // Set the flag when pin has been LOW for the required duration
    }
}

bool isPinHighForDuration(int pin, unsigned long duration) {
    unsigned long highStartTime = 0;
    while (digitalRead(pin) == HIGH) {
    if (highStartTime == 0) {
        highStartTime = millis(); // Record the time when pin goes HIGH
    } else if (millis() - highStartTime >= duration) {
        Serial.println("Signal has been HIGH for at least 0.5 seconds.");
        return true; // Pin has been HIGH for the specified duration
    }
    }
    return false; // Pin was not HIGH for the specified duration
}

bool isPinLowForDuration(int pin, unsigned long duration) {
    unsigned long lowStartTime = 0;
```

```
    while (digitalRead(pin) == LOW) {
    if (lowStartTime == 0) {
        lowStartTime = millis(); // Record the time when pin goes HIGH
    } else if (millis() - lowStartTime >= duration) {
        Serial.println("Signal has been LOW for at least 0.5 seconds.");
        return true; // Pin has been HIGH for the specified duration
    }
    }
    return false; // Pin was not HIGH for the specified duration
}

void moveServo(int fromPos, int toPos) {
    Serial.print("Moving");
    if (fromPos < toPos) {
    for (int currPos = fromPos; currPos <= toPos; currPos +=1) {
    Serial.print(".");
    myservo.write(currPos);
    delay(10);
    }
    }
    else {
    for (int currPos = fromPos; currPos >= toPos; currPos -=1) {
    Serial.print(".");
    myservo.write(currPos);
    delay(10);
    }
    }
    Serial.println();
    Serial.print("Moved from ");
    Serial.print(fromPos);
    Serial.print(" to ");
    Serial.println(toPos);
}
```