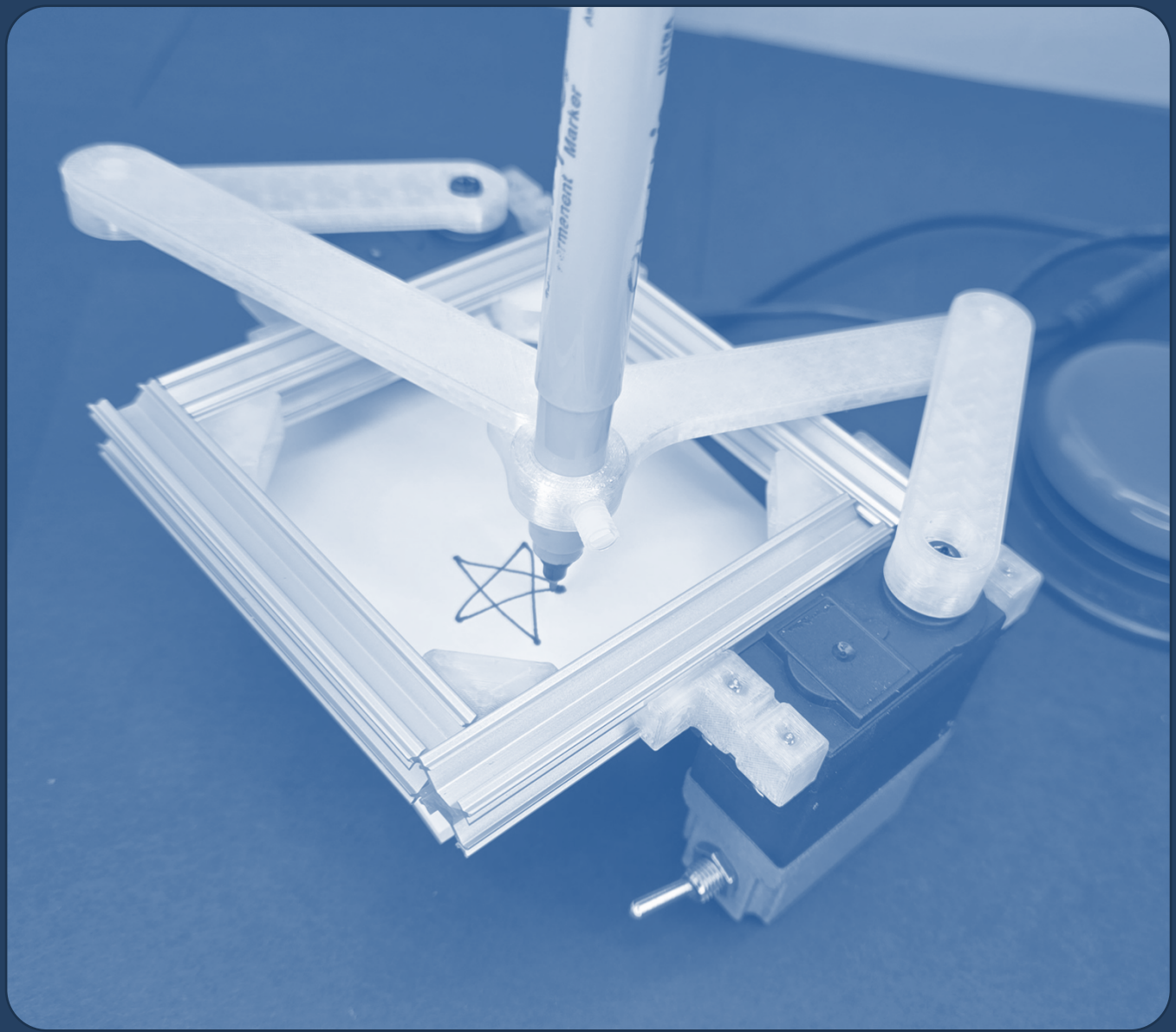
SMARTSERVO

AUTOMATED DRAWING: PATTERN CREATOR KIT



SMARTSERVO PROJECT

AUTOMATED DRAWING: PATTERN CREATOR KIT

Version 1.0 | Published: June 17,2025 | Author: Judson Wagner, Wagner Labs LLC

Commercial Use & Smart Servo Requirement

For commercial use, bulk educational licensing, or use outside of traditional educational settings, please contact Wagner Labs LLC.

Hardware Requirement: This guide requires **Smart Servo devices** to complete the projects and activities described. Smart Servos are available through the Smart Servo Store and authorized distributors.

About the Smart Servo Project

The Smart Servo Project empowers inclusive innovation by providing accessible tools for creating assistive technologies and engaging STEM education. Our mission is to bridge technology and compassion through community-driven maker education.

Support our mission by purchasing Smart Servos and sharing our resources with your educational community.

Contact Information:
Judson Wagner | Wagner Labs LLC
Email: **Judson@WagnerLabs.net**
Website: **WagnerLabs.net/SmartServo**
Smart Servo Store: **WagnerLabs..Store**

Client: Maya Patel, Age 13

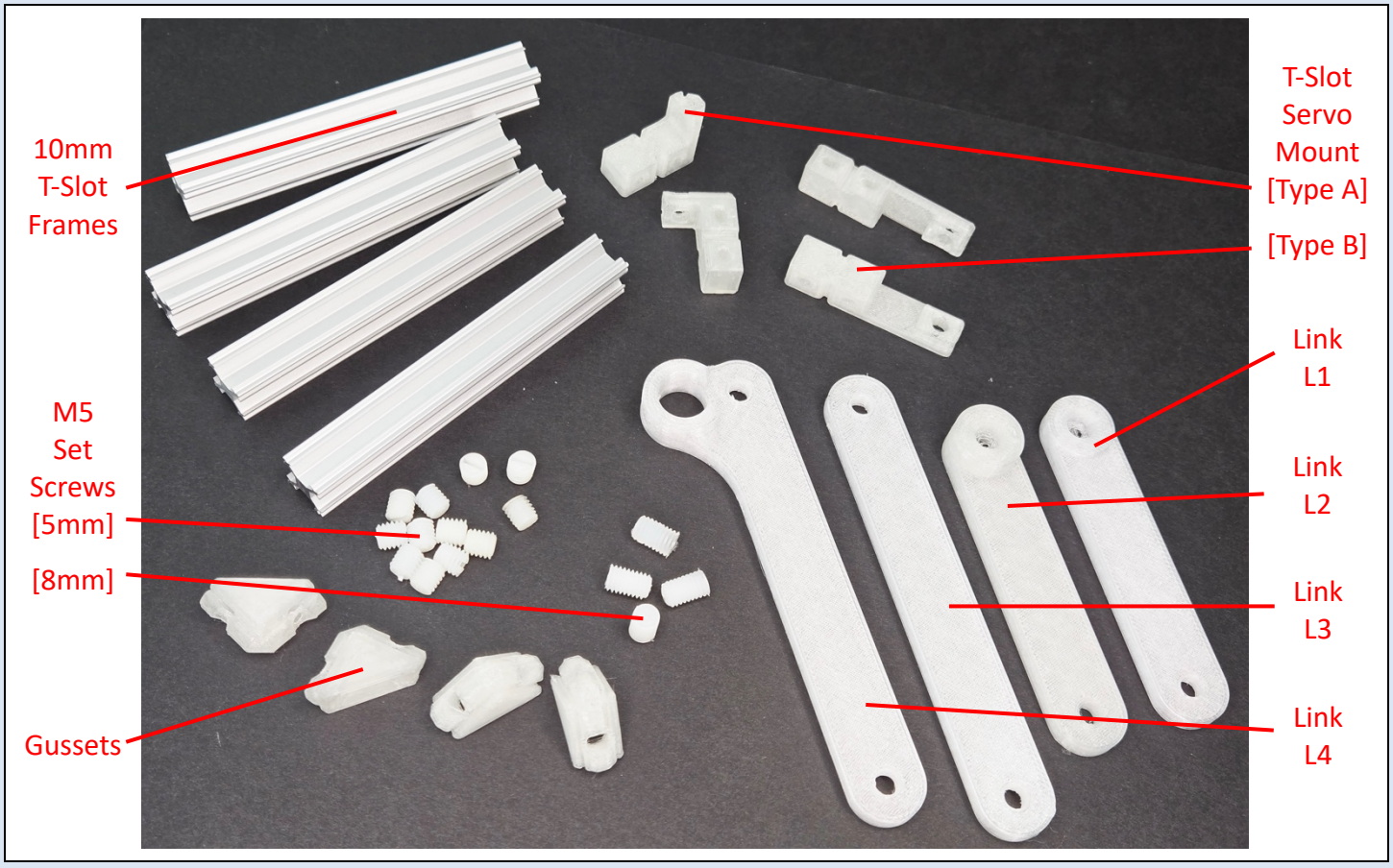
About Me: I'm a 7th grader who loves art and mathematics. I have limited hand strength and coordination due to juvenile arthritis, which makes drawing and writing very tiring and sometimes painful.

My Challenge: I enjoy creating geometric art and exploring mathematical patterns through drawing, but extended drawing sessions cause significant hand fatigue and joint pain. I want to continue exploring the intersection of math and art without physical limitations.

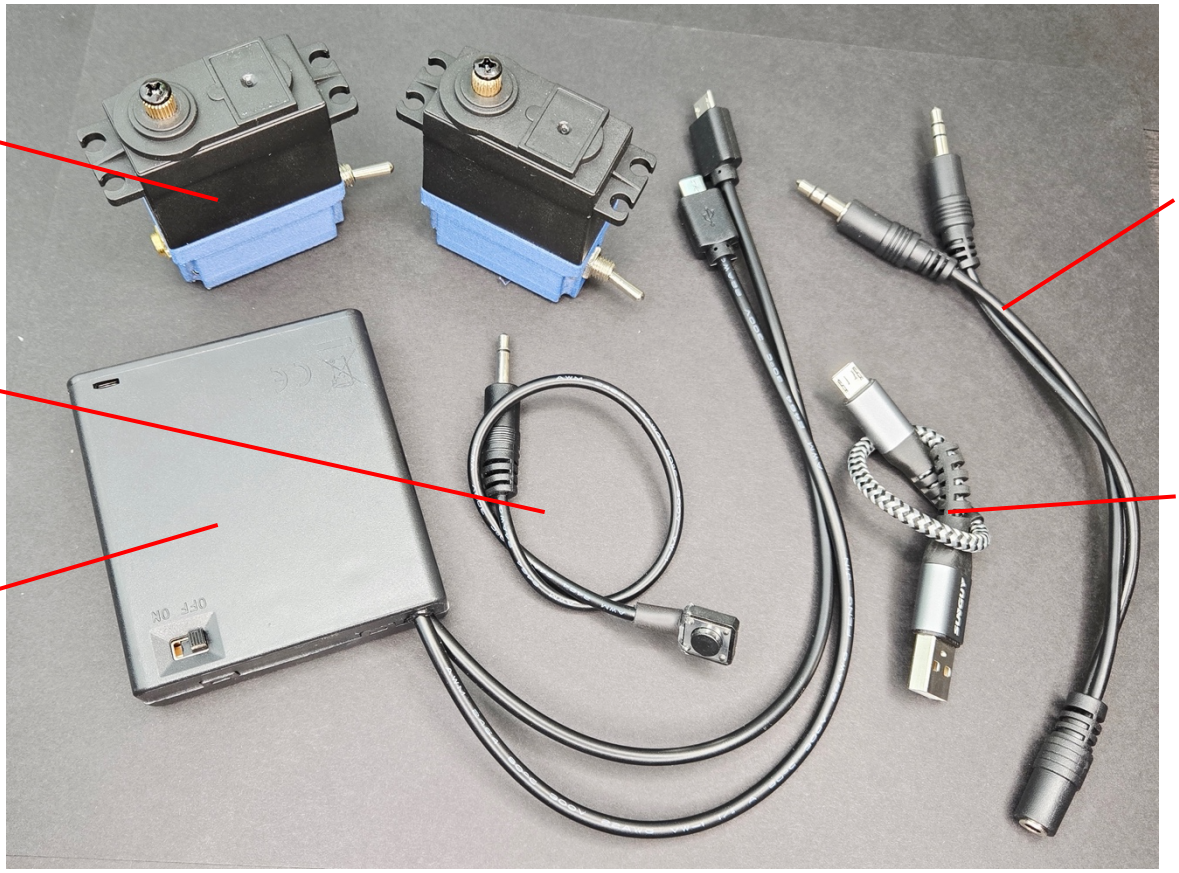
Technical Need: An automated drawing system that can create precise geometric patterns and shapes, allowing me to explore mathematical art concepts without the physical strain of manual drawing.

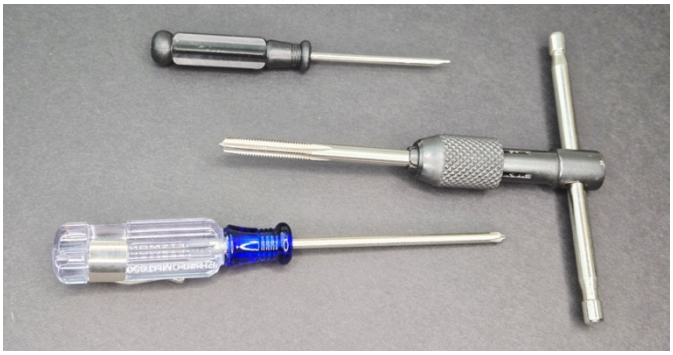
Let's now investigate our kit and see if we can get started on something that can assist Maya.

STEP 1: Lay out all the components that are new in this kit.



STEP 2: Make sure you have these items from your previous kits.

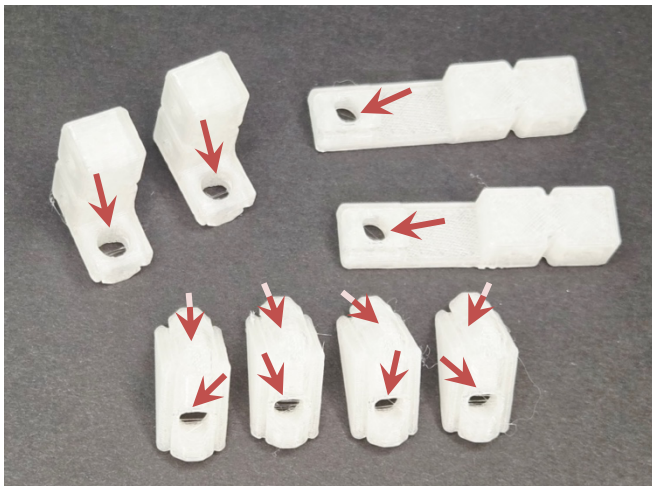


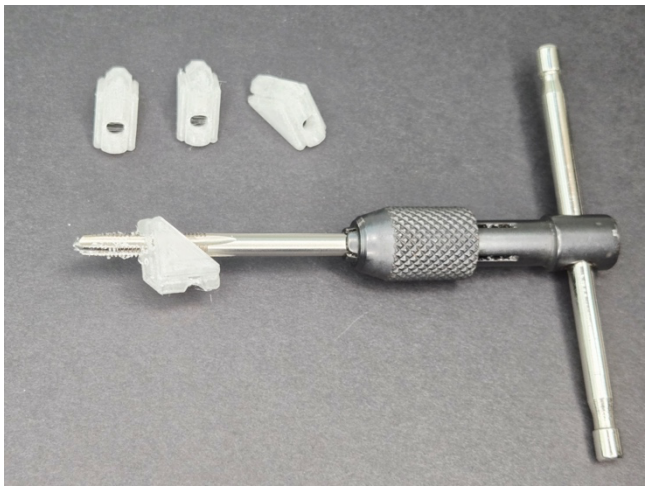


Tap, Tap Handle, Flat, and Phillips Screw Drivers

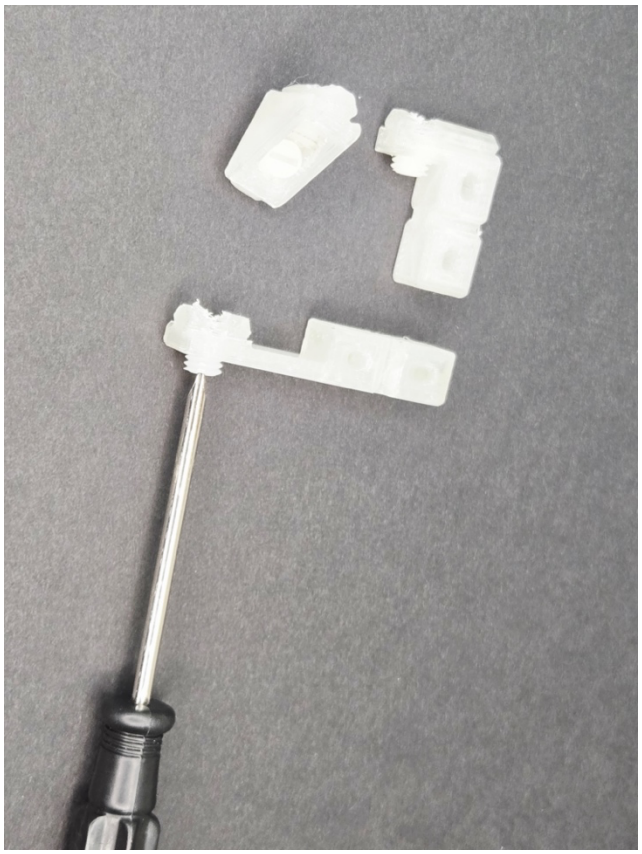
12x M5 [5mm], 4x M5 [8mm], & 8x Mounting Screws

STEP 3: Tap all of the parts that will use the 5mm M5 Set Screws and T-Slot framing. Look for the 4.3mm holes; there should be 12 total. (Note: It may be easier to tap the Gussets from the inside out as shown.)

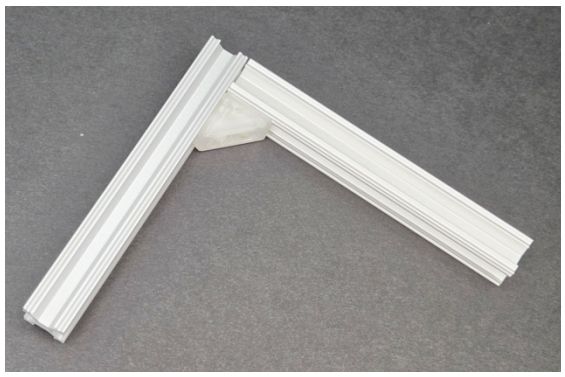
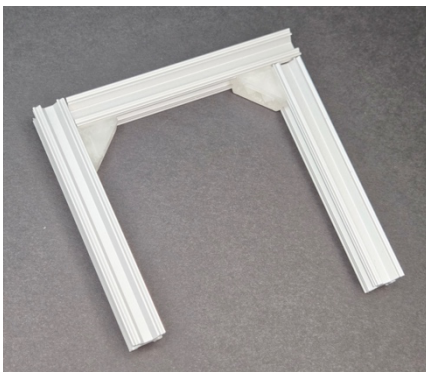




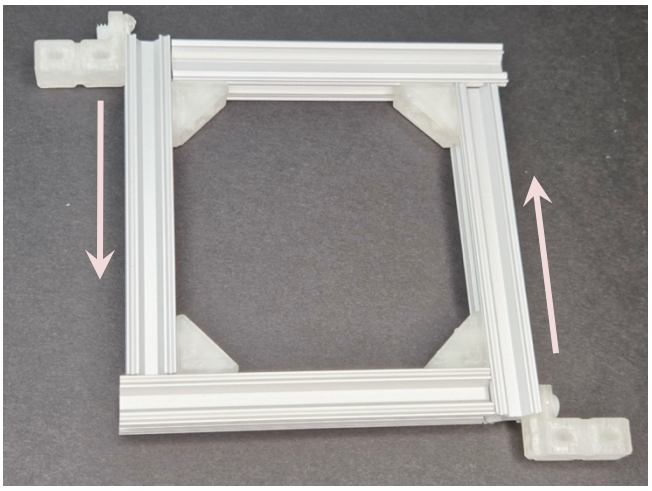
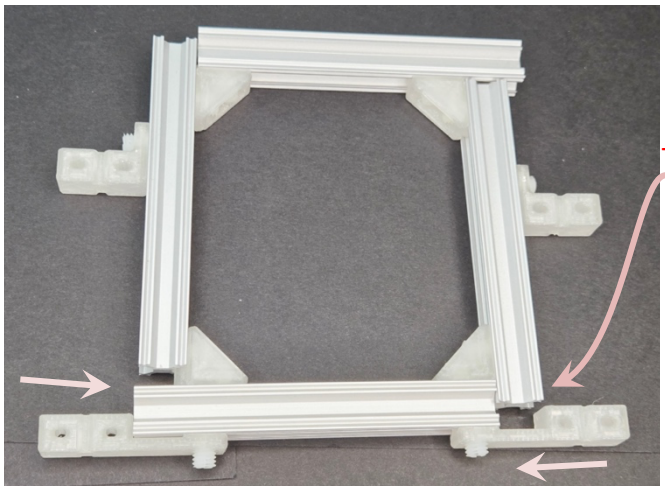
STEP 4: Use the Flat-end Screwdriver to screw in the 5mm M5 set screws. Be sure that they can pass through but back them off so that only a couple of millimeters are in.



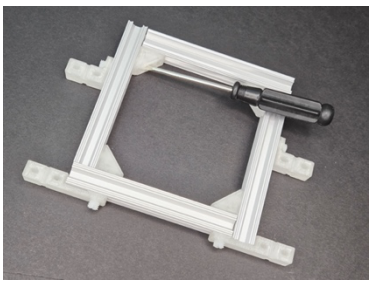
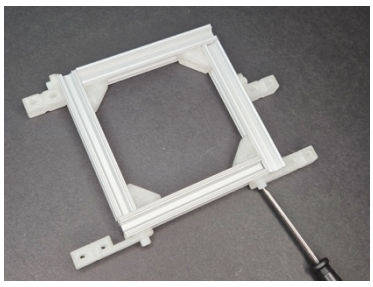
STEP 5: Start assembling the T-Slot Frames and the Gussets as shown below. The pieces should slide together. If not, try backing off the set screws more.

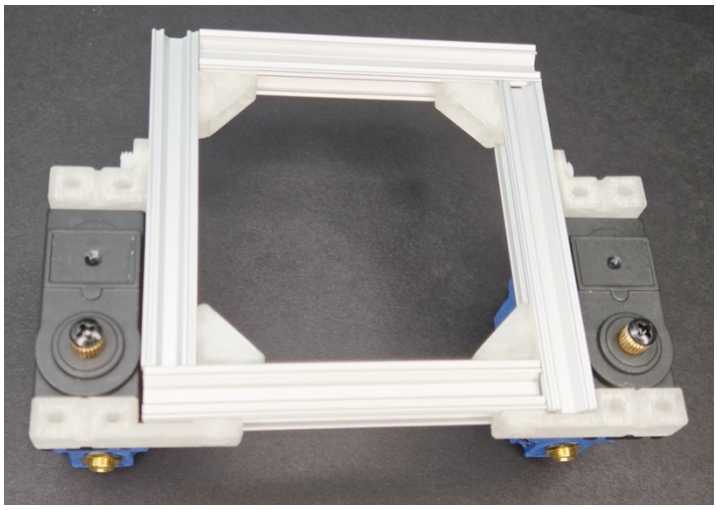

STEP 6: Slide on the Servo Mounts, starting with the Type A Servo Mounts and then the Type B. Note that this requires a little strategy due to the access to the T-slots and since the Gussets will need to move to allow one of the Type B Servo Mounts to fit into position.

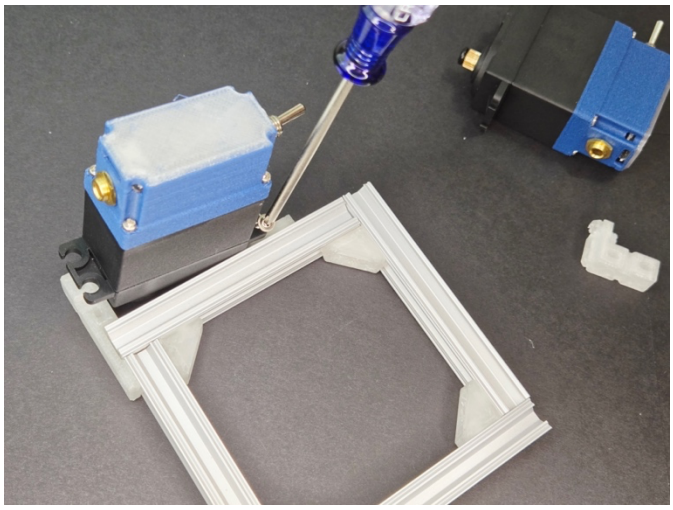



STEP 7: Once everything is in place and ends of each length of the T-Slot Frames are flush and square with each subsequent piece, use the Flat Screwdriver to tighten the Gussets first and then the Type B Servo Mounts. Take care to not over-tighten.

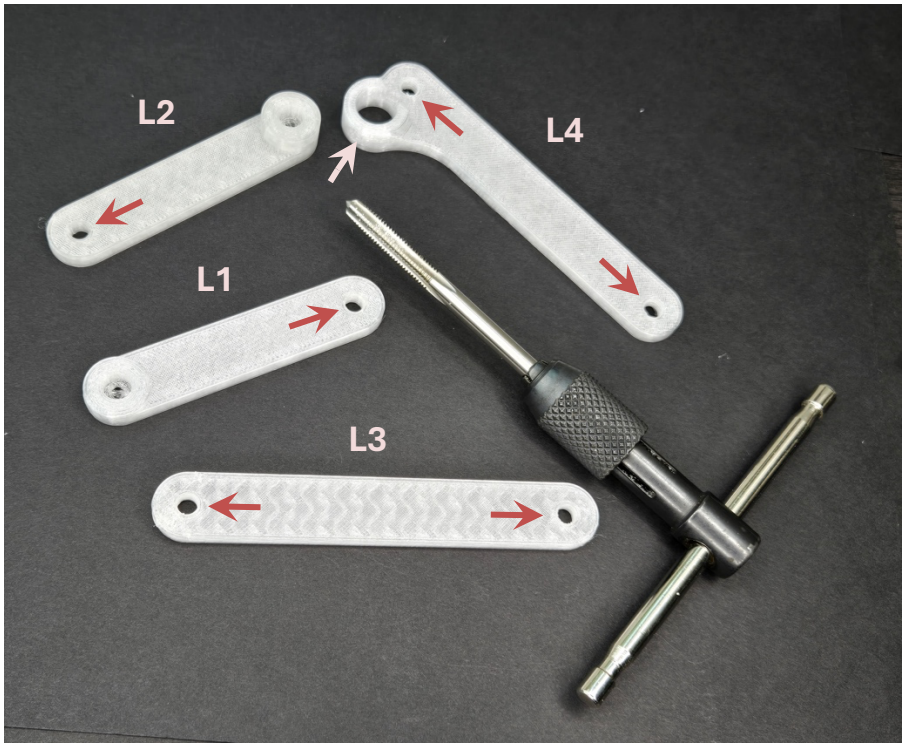



STEP 8: Place the T-Slot "square" over the two Smart Servos and slide the Type A Servo Mount pieces over the mounting holes on the servos. Tighten down the set screws on the Type A Servo Mount, flip over, and mount the servos using the Mounting Screws.

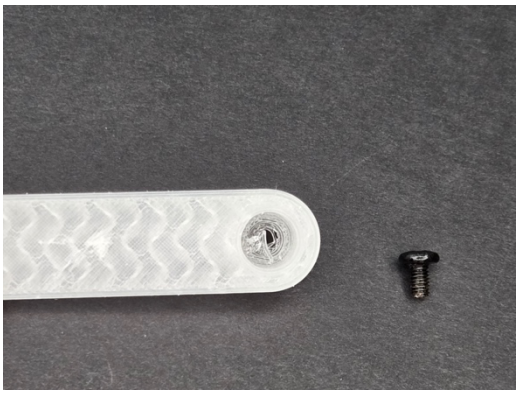


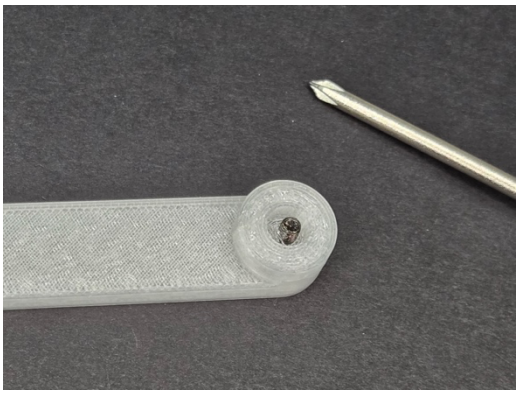


STEP 9: Tap each of the 4.3mm holes in the linkages. Note that L4 has three (3) holes that need to be tapped. There should be seven (7) holes to tap in the linkages.



STEP 10: During the 3D printing process, some filament in strands will cover the Spline holes on Links 1 and 2. The Spline shape is perfectly fine but there may be some interference for the M3 screw. Take a moment and make sure that the M3 Spline Screw can fit through the hole for Link 1 and Link 2.

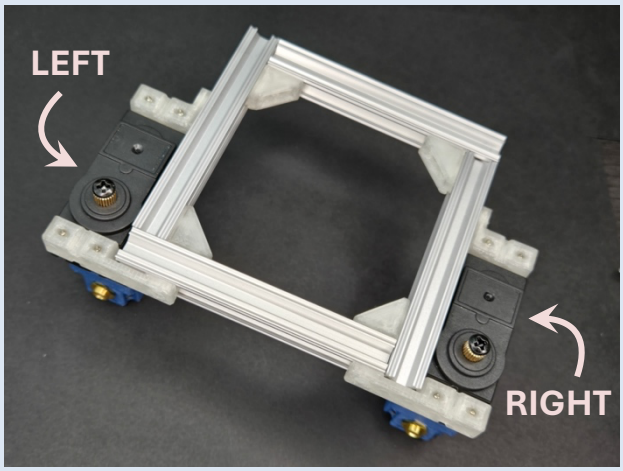




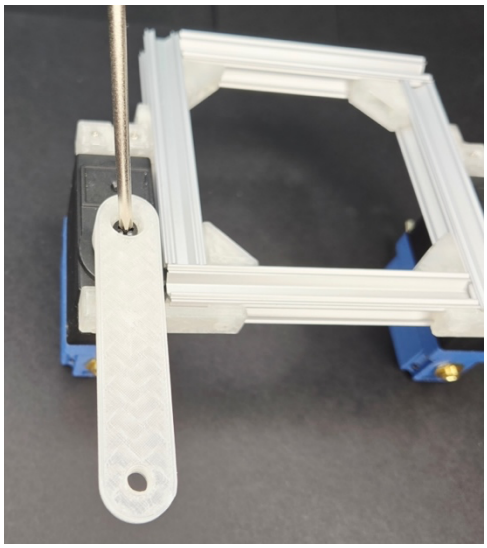
STEP 11: At this point, we'll need to make sure that each servo is set to 0-degrees. Go into the Circuit Python code.py file and make this change. Here's a simple snip to assist you with this:

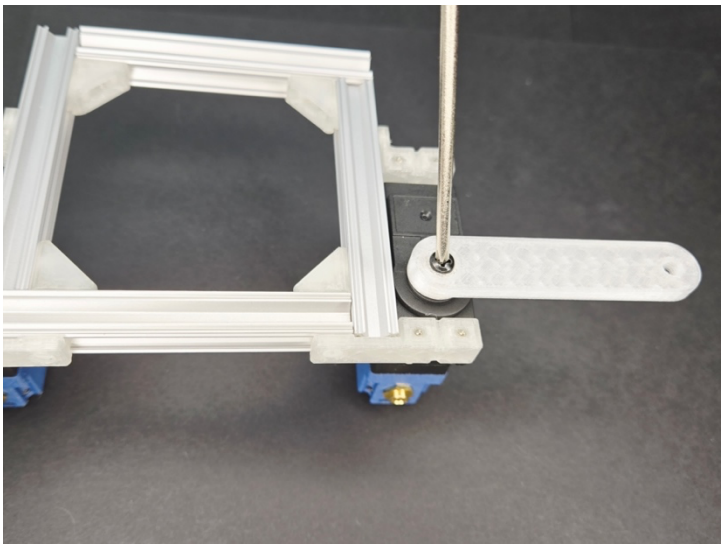
```
import board
import pwmio
import servo
pwm = pwmio.PWMOut(board.A2, duty_cycle=2 ** 15, frequency=50)
servo1 = servo.Servo(pwm)
while True:
      servo1.angle = 0
```

STEP 12: We also want to make sure that we can clearly differentiate the two Smart Servos. Use the designations below to identify "LEFT" and "RIGHT".

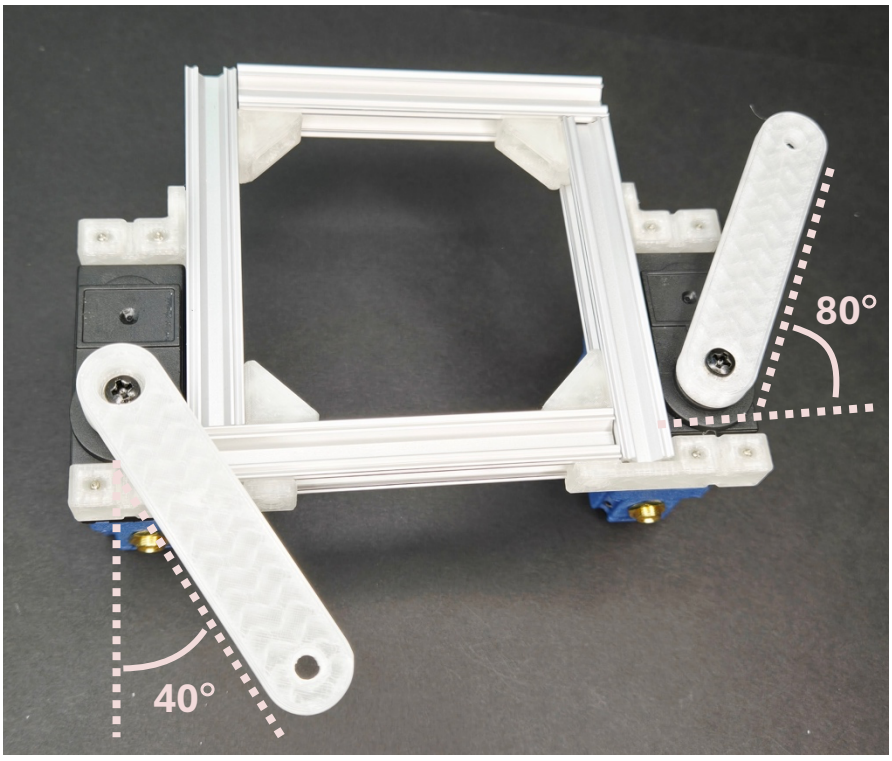


STEP 13: With the LEFT Smart Servo and the RIGHT Smart Servo set to 0-degrees. Attach Link L2 to the LEFT servo spline directed straight down (or south) and attach Link L1 to the RIGHT servo spline directed to the right (or east).

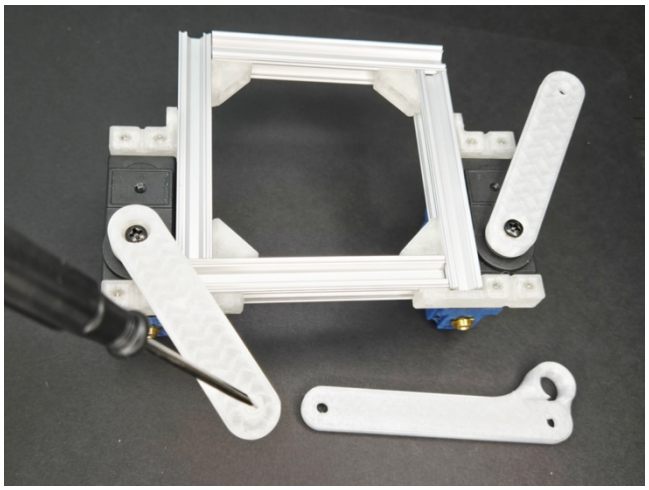


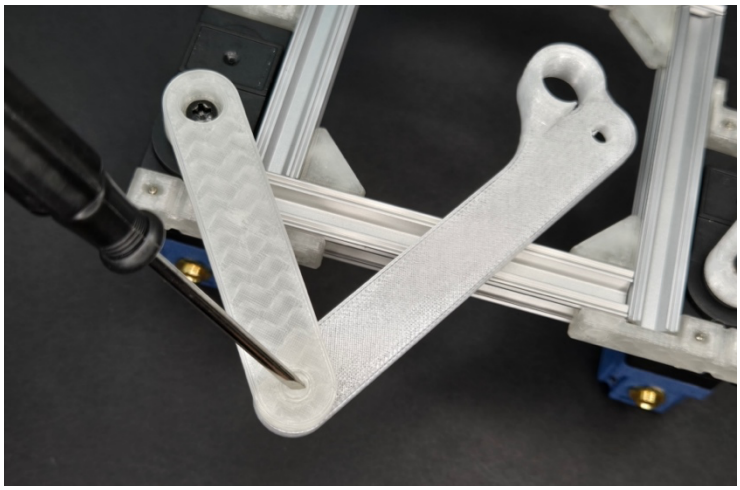


STEP 14: Going back into the Circuit Python code.py, rotate L2 on the LEFT servo to 40-degrees and rotate L1 on the RIGHT servo to 80-degrees.

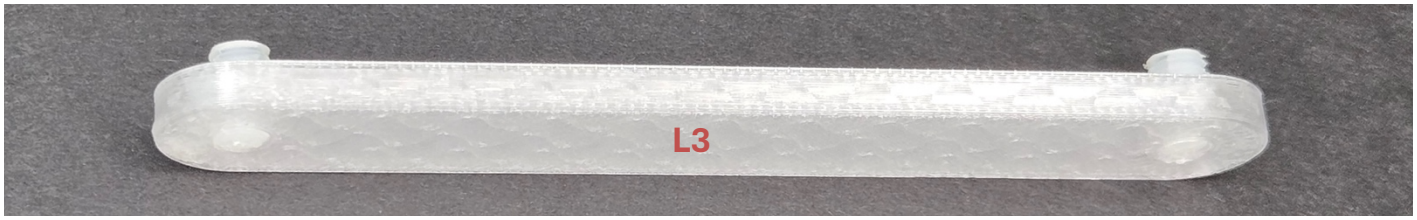


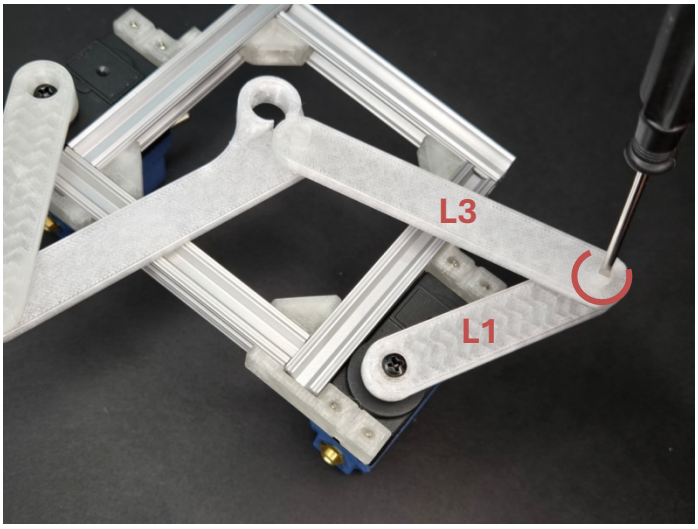
STEP 15: Connect linkages L2 to L4. Start by putting an 8mm M5 Set-Screw through the other end of L2. Just as it emerges on the bottom side, place a matching hole from L4 as shown. Double check the orientation of L4. Note that L2 and L4 are 4mm thick at the sites of the tapped holes so you should be able to screw in the 8mm set screw to connect the two.

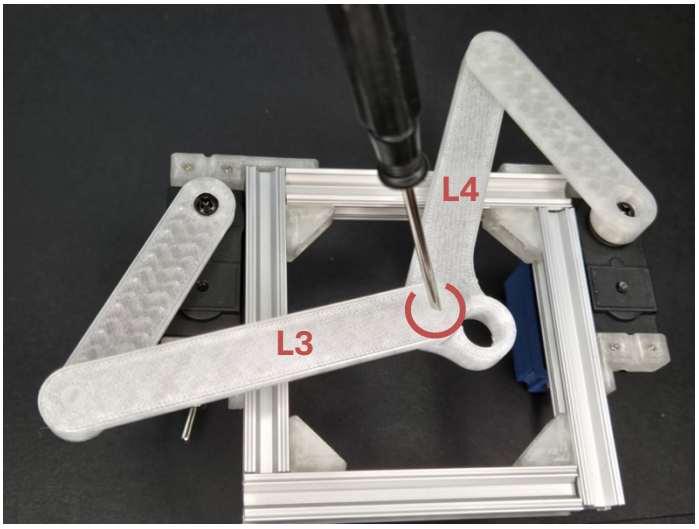




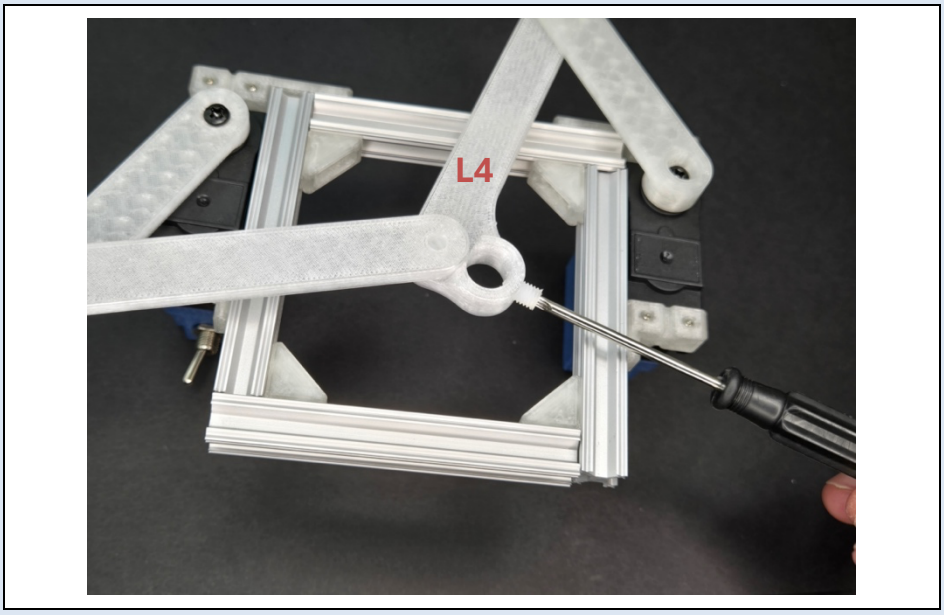
STEP 16: Insert setscrews into each end of L3 until they start to emerge at the bottom. Connect each end, screwing downwards into L4 and L1.







STEP 17: Add the final 8mm long M5 set screw to the end of the L4. We'll use this to hold down a writing instrument later.



STEP 18: Let's pause and update the code.py for each of the Smart Servos. The example code below will move each servo to five different angles before returning to the original position. Due to the linkage, each combination of angles will put the writing instrument at a specific location, or point. Each of the five angle combinations are for the positions at five points on a star. As the writing instrument moves from point to point, it draws out a pentagram star.

POINT	LEFT SERVO ANGLE	RIGHT SERVO ANGLE
Point 1 (Starting Point)	37 Degrees	80 Degrees
Point 2	68 Degrees	68 Degrees
Point 3	40 Degrees	60 Degrees
Point 4	55 Degrees	85 Degrees
Point 5	60 Degrees	50 Degrees
Point 6 (Point 1 again)	37 Degrees	80 Degrees

Note that the code below has two parts. The top part is not commented out and the bottom part is commented out. This should remain this way on the LEFT servo. When adding the code to the right servo, make sure to comment out the top part, and un-comment the bottom part.

Where the numbers came from will be covered later in this guide.

EXAMPLE CODE [STAR Pattern]

The modified code below can be copied to each Smart Servo. For the RIGHT Servo, be sure to comment out the section for LEFT and un-comment the section for RIGHT.

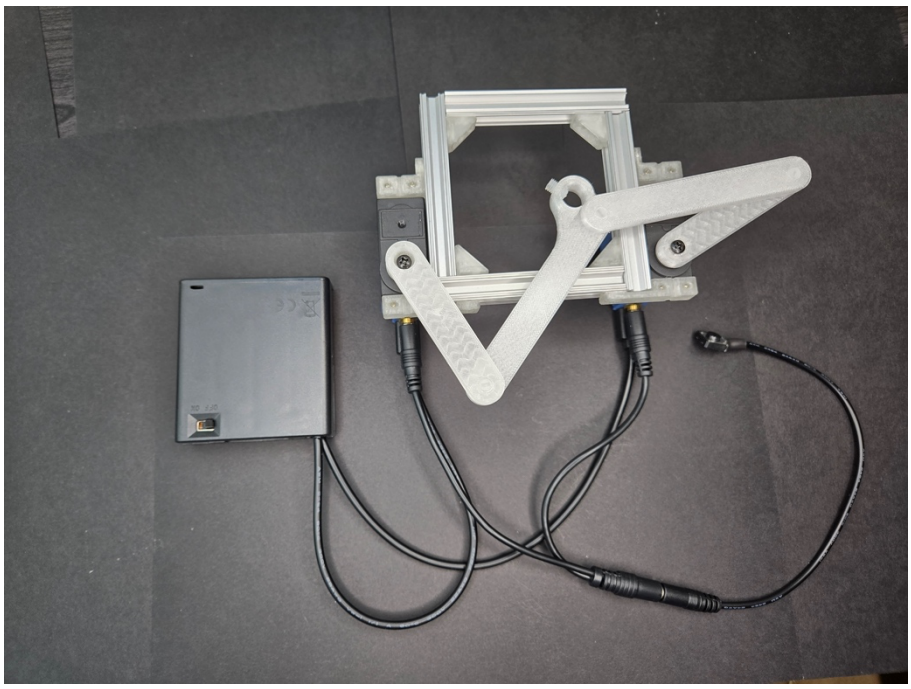
COPY & PASTE this code from: tinyurl.com/ss-code-draw

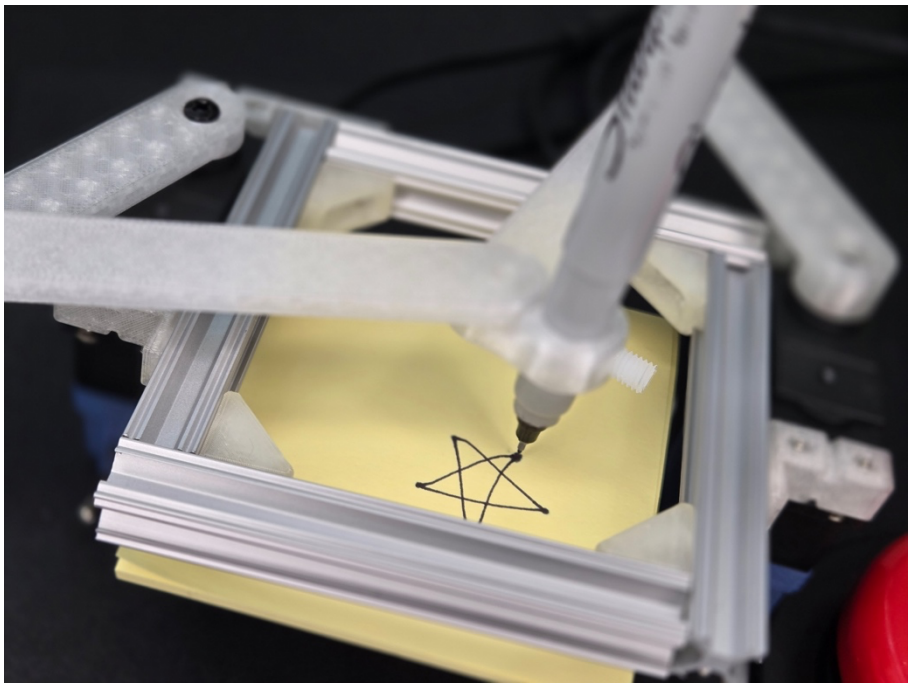
```
''
#DrawBot v8 - LEFT
import time
import board
from digitalio import DigitalInOut, Direction, Pull
import pwmio
import servo

import adafruit_dotstar
pix = adafruit_dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1)

button = DigitalInOut(board.D2)
button.direction = Direction.INPUT
button.pull = Pull.UP

switch = DigitalInOut(board.D1)
switch.direction = Direction.INPUT
switch.pull = Pull.DOWN

pwm = pwmio.PWMOut(board.A2, duty_cycle=2 ** 15, frequency=50)
servo1 = servo.Servo(pwm)

state = 0
while True:
    pix[0] = (0,100,0)
    if switch.value == 1 and button.value == 0 and state == 0:
        servo1.angle = 37 # Point 1
        pix[0] = (80,0,80)
        time.sleep(0.5)
        servo1.angle = 68 # Point 2
        pix[0] = (150,0,0)
        time.sleep(0.5)
        servo1.angle = 40 # Point 3
        pix[0] = (0,150,0)
        time.sleep(0.5)
        servo1.angle = 55 # Point 4
        pix[0] = (0,0,150)
        time.sleep(0.5)
        servo1.angle = 60 # Point 5
        pix[0] = (100,20,50)
        time.sleep(0.5)
        servo1.angle = 37 # Back to Point 1
```

```
        pix[0] = (80,0,80)
        time.sleep(0.5)
        state=1
        pix[0] = (0,100,0)
    elif switch.value == 1 and button.value == 1 and state == 1:
        servo1.angle = 37
        pix[0] = (0,100,0)
        state=0
        time.sleep(0.0)
    elif switch.value  ==0 and button.value == 0:
        servo1.angle = 37
        pix[0] = (125, 115, 3)
        time.sleep(0.5)
    elif switch.value == 0 and button.value == 1:
        servo1.angle = 37
        for i in range (10,255,1):
            pix[0] = (i,0,0)
        for i in range (255,10,-1):
            pix[0] = (i,0,0)
        time.sleep(1.0)
    time.sleep(0.0)
''
'''
#DrawBot v8 - RIGHT
import time
import board
from digitalio import DigitalInOut, Direction, Pull
import pwmio
import servo

import adafruit_dotstar
pix = adafruit_dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1)

button = DigitalInOut(board.D2)
button.direction = Direction.INPUT
button.pull = Pull.UP

switch = DigitalInOut(board.D1)
switch.direction = Direction.INPUT
switch.pull = Pull.DOWN

pwm = pwmio.PWMOut(board.A2, duty_cycle=2 ** 15, frequency=50)
servo1 = servo.Servo(pwm)

state = 0
while True:
    pix[0] = (0,100,0)
```

```
    if switch.value == 1 and button.value == 0 and state == 0:
        servo1.angle = 80 # Point 1
        pix[0] = (80,0,80)
        time.sleep(0.5)
        servo1.angle = 68 # Point 2
        pix[0] = (150,0,0)
        time.sleep(0.5)
        servo1.angle = 60 # Point 3
        pix[0] = (0,150,0)
        time.sleep(0.5)
        servo1.angle = 85 # Point 4
        pix[0] = (0,0,150)
        time.sleep(0.5)
        servo1.angle = 50 # Point 5
        pix[0] = (100,20,50)
        time.sleep(0.5)
        servo1.angle = 80 # Back to Point 1
        pix[0] = (80,0,80)
        time.sleep(0.5)
        state=1
        pix[0] = (0,100,0)
    elif switch.value == 1 and button.value == 1 and state == 1:
        servo1.angle = 80
        pix[0] = (0,100,0)
        state=0
        time.sleep(0.0)
    elif switch.value  ==0 and button.value == 0:
        servo1.angle = 80
        pix[0] = (125, 115, 3)
        time.sleep(0.5)
    elif switch.value == 0 and button.value == 1:
        servo1.angle = 80
        for i in range (10,255,1):
            pix[0] = (i,0,0)
        for i in range (255,10,-1):
            pix[0] = (i,0,0)
        time.sleep(1.0)
    time.sleep(0.0)
''
```

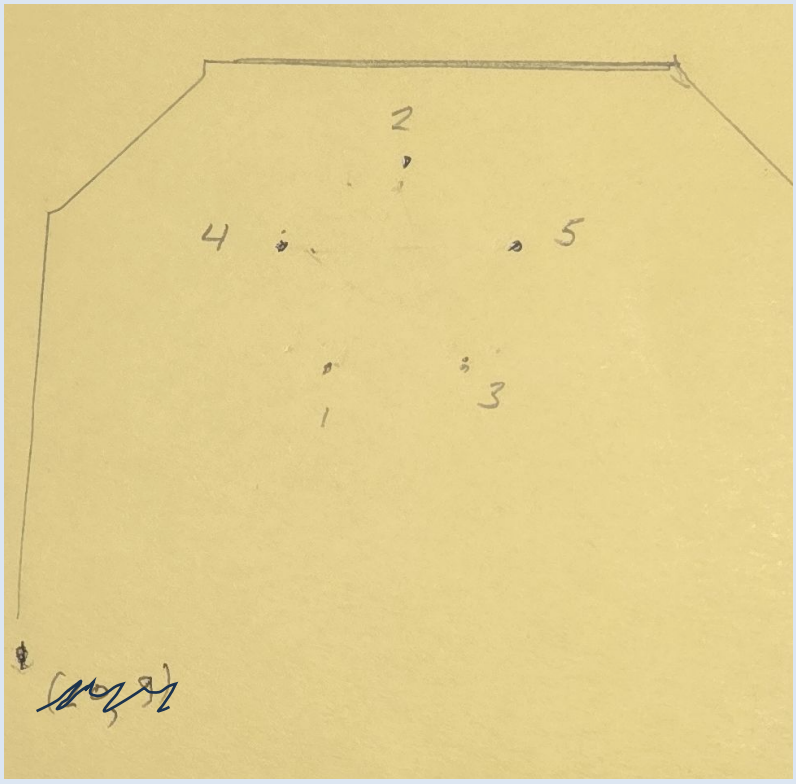
STEP 19: Now try it out. You'll want to simultaneously connect both Smart Servos using the Dual 3.5mm Jack to connect just one Test Button or Assistive Button and the Dual Power Pack. Do a test run to see if the end of L4 draws out the familiar pentagram pattern and the lights of both Smart Servos is the same color at the same time. If ready to proceed, get a piece of paper or a sticky notepad (perfect size really) and a writing instrument. Use the set screw to hold onto the writing instrument. Hold the stick note to the bottom of the T-Slot Square and press the button.





CREATE A SHAPE FOR MAYA

To create a shape that Maya would want, we can follow the steps used for this pentagram and modify it as needed. For this guide, we picked a star and thought about how most people would draw a star which is actually the common pentagram pattern. We then need to space out and number points that would act like a dot-to-dot for someone to follow along. In this case, we want out Smart Servo DrawBot to follow along.



We used a sticky note and secured it below the t-slot square. We also traced part of the t-slot and gussets so that we could go back to check later it the paper fell off. Using a code like the one from STEP 11 above, we changed the angles of the LEFT and RIGHT Smart Servos until the center of the hole in Link 4 was just above the dot. We then recorded the angles of each Smart Servo for each point. Once we completed our table like the one in STEP 18, we could enter the information into our cody.py.

Note: This process was easier when working with a partner and we could put a computer on each Smart Servo as we identified the best angles

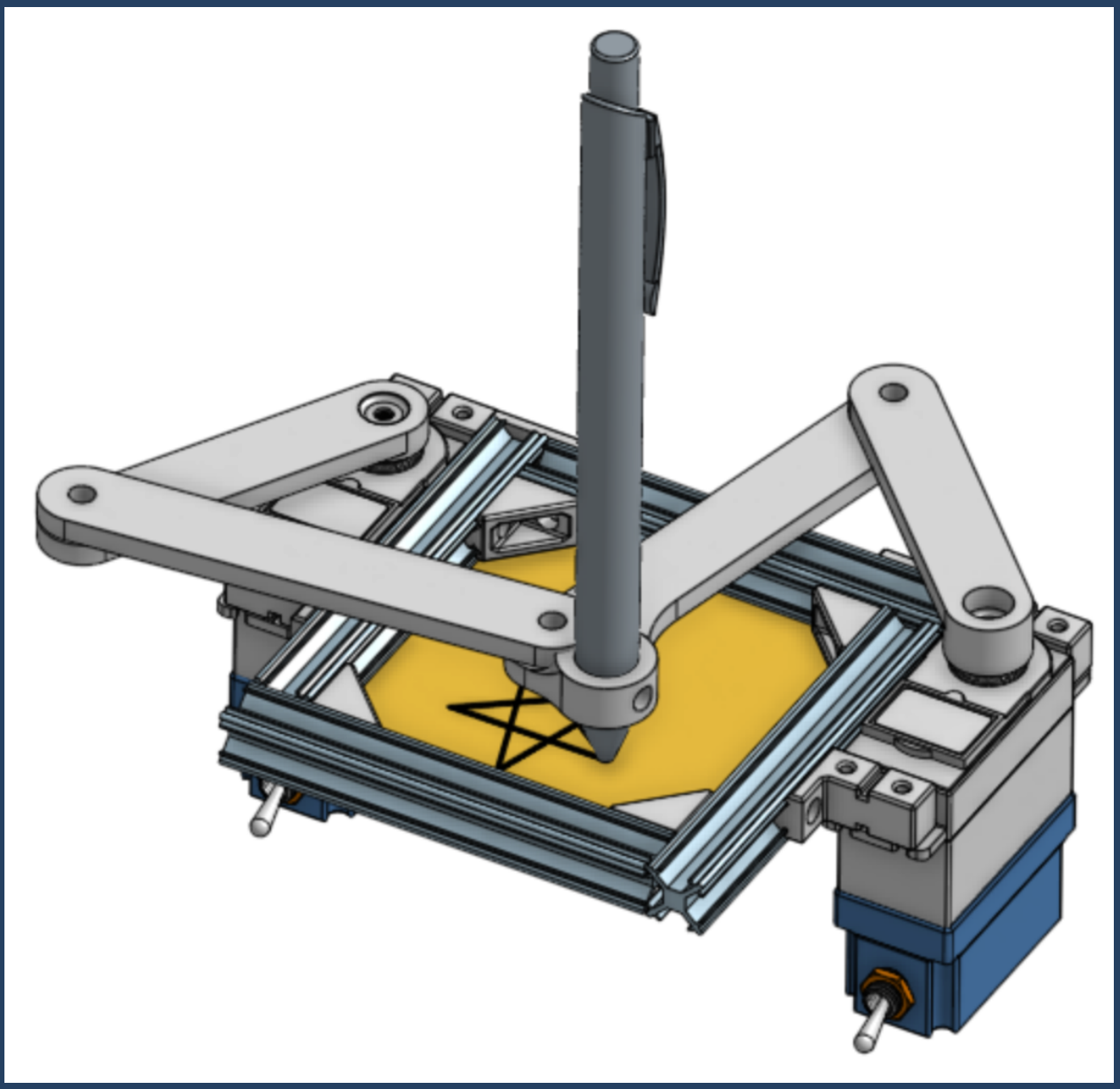
Do you think you could follow a similar process for Maya's shape?

REMINDER ABOUT CODING SNIPS

If you want to return your code to the original “factory setting”, just copy and paste from here: tinyurl.com/SmartServoSnips

3D PRINTING FILES

If you’re able to 3D Print, download the 3D parts used in this project here: tinyurl.com/SS-STL-DRAW



THE BIGGER PICTURE

UNDERSTANDING THE ROBOTICS BEHIND YOUR DRAWBOT

Five Bar Linkages in Real Robotics

What you've just built is called a five bar linkage system - a fundamental mechanism used in many real-world robots! Count the "bars" in your DrawBot: the two T-slot frame pieces act as fixed ground links, plus Links L1, L2, L3, and L4 make five total. This type of linkage gives robots precise control over where their "end effector" (in your case, the drawing tool) can reach within a specific workspace.

Five bar linkages are everywhere in robotics because they're incredibly versatile. Industrial robots use them for pick-and-place operations in factories, surgical robots use similar systems for precise movements during operations, and even some prosthetic arms use five bar mechanisms to give amputees natural-feeling control. The key advantage is that two motors can control the position of the end point anywhere within a curved workspace - just like how your two servos can move your pen to draw any shape within reach.

The Challenge of Inverse Kinematics

When you were figuring out which servo angles would place your pen at each point of the star, you were actually solving what roboticists call an inverse kinematics problem. Here's the challenge: if you know where you want the robot's end effector to be (like "I want the pen at this specific dot"), how do you figure out what angles each joint needs to move to get there?

This is the opposite of forward kinematics, which asks "if I set the servos to these specific angles, where will the pen end up?" Inverse kinematics is much harder to solve mathematically, which is why you had to use trial and error to find the right angles. Professional robots use complex computer algorithms and lots of math (trigonometry, calculus, and linear algebra) to solve these problems instantly, but the basic challenge is exactly what you experienced - working backwards from where you want to be to figure out how to get there.

From Straight Lines to Smooth Curves

Your DrawBot currently draws straight lines between points, creating the sharp angles of a pentagram. But real robots need to draw smooth curves and follow complex paths. Advanced DrawBots and CNC machines use techniques like trajectory planning and interpolation to create smooth motion between points. Instead of jumping from angle to angle, they gradually change the servo positions over time, creating fluid curves.

Engineers also use mathematical functions called splines and Bézier curves to plan smooth paths. These are the same mathematical concepts used in computer graphics software like Photoshop or in designing car bodies. Your DrawBot has given you a foundation to understand how robots translate mathematical descriptions of shapes into precise mechanical movements - whether they're drawing on paper, welding car frames, or performing delicate surgery.