Engineering Journal of Automated Soprano Recorder

Team member: Ash Wang, Alice Liu, Grace Luo
Project focus: Soprano recorder

Our goal was to design an automated musical instrument capable of playing a soprano recorder. The system needed to control three main functions: supplying a stable airflow, covering and uncovering the recorder's finger holes, and producing clear starts, stops, and rests between notes.

Instead of using the same actuator for every task, we selected different mechanical and electrical components based on the physical requirements of each part of the recorder. The final system combines a centrifugal blower, a 3D-printed air nozzle, servo motors, solenoids, lightweight paper, rubberbands, and foam earplugs.

We divided the whole project into several small tasks:

July 23rd: air flow, card board prototype, control one servo at a time, control one solenoid at a time

July 24th: port expander, control multiple servos at the same time, control multiple solenoids at the same time, laser cutting the platform, discuss potential control methods

July 25th: assembling platform, assemble servos and solenoids, control servos and solenoids together, label wires

July 26th: adjust parameters, tuning, experiment different material, try different ESP32

July 27th: play notes according to MIDI file, adjust position and mounting method

July 28th: experiment with different tunnels, experiment different mute mechanisms

July 29th: 3D print tunnel, tuning with new tunnel, connecting to WiFi

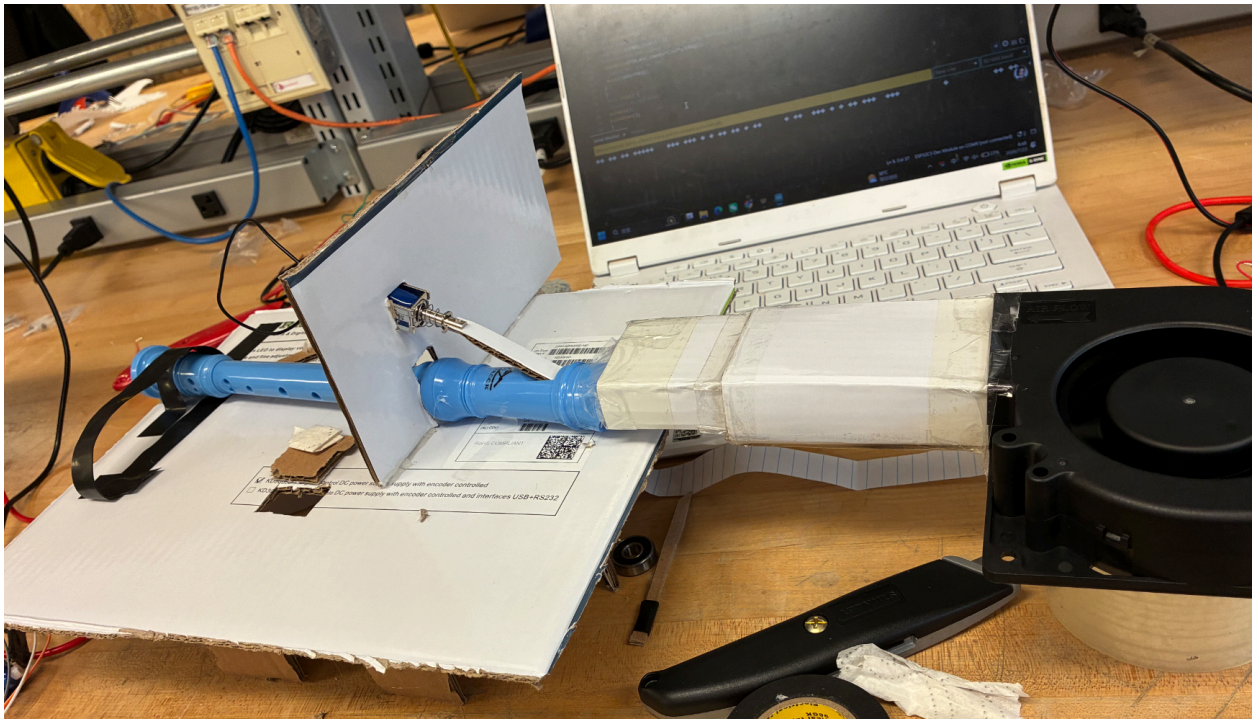
July 30th: final mute mechanism, play notes after receiving WiFi signals

July 23rd

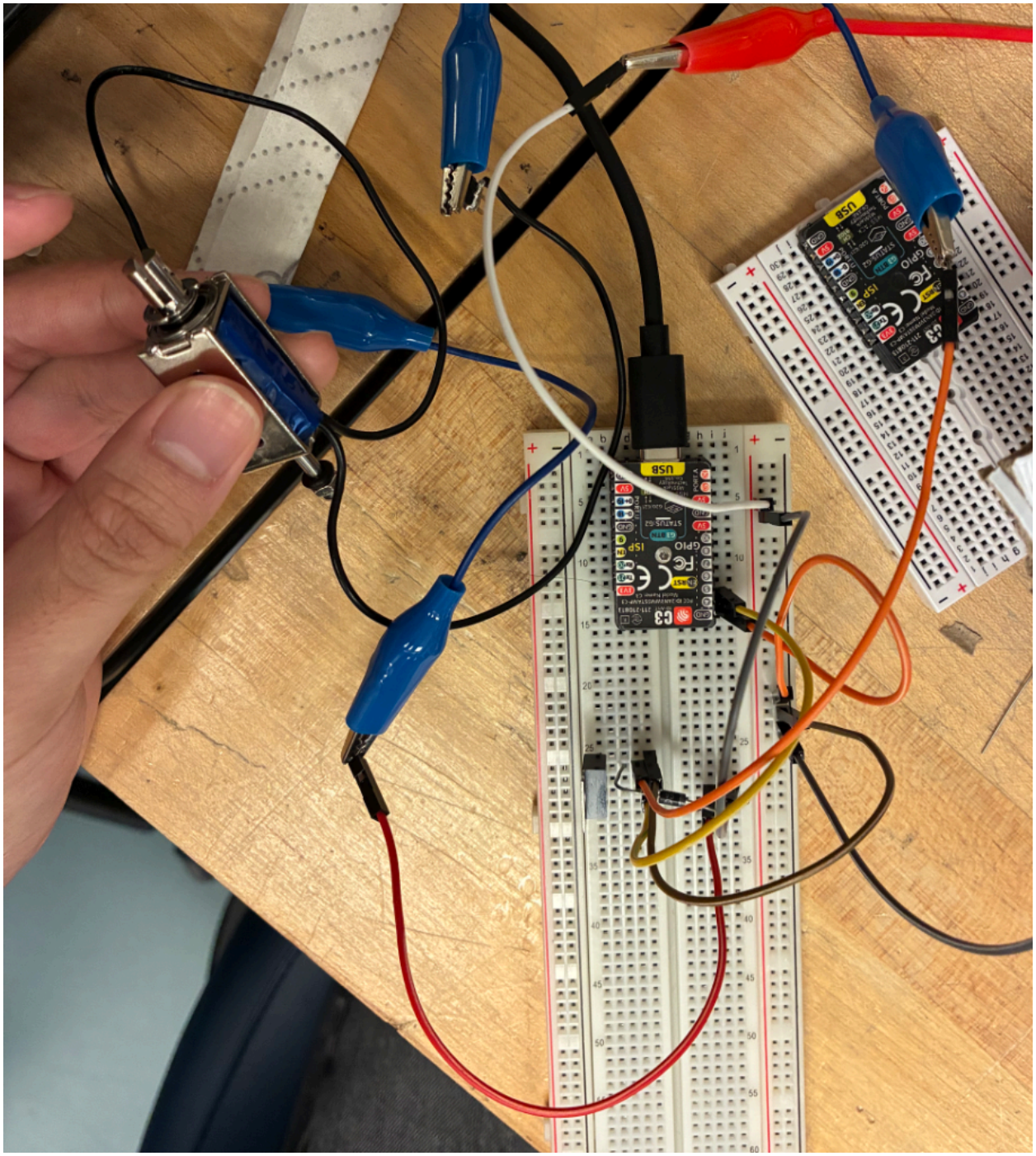
One of the first challenges was creating enough controlled airflow to make the recorder produce a clear sound. The centrifugal blower could move a large amount of air, but its original outlet was too wide to direct the airflow effectively into the narrow recorder windway. We made a simple air tunnel out of a card board and connected the blower outlet to the recorder mouthpiece. The nozzle reduced the cross-sectional area of the air path and concentrated the airflow into a smaller opening.

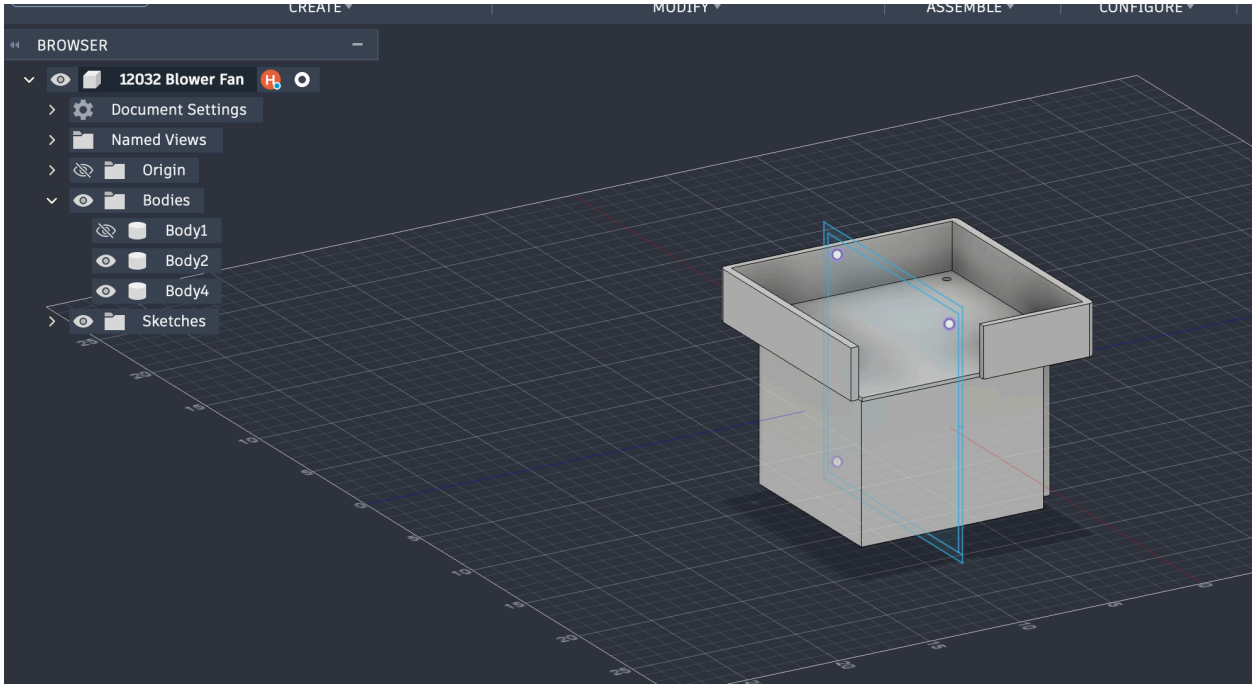
According to the continuity principle, reducing the cross-sectional area of a flow path increases the air velocity, assuming that a similar volume of air continues to move through the system. The nozzle therefore did not simply “create pressure.” Instead, it increased the speed and concentration of the air jet and improved the transfer of airflow into the recorder.

We also used the card board to make a platform to hold the recorder. By blocking the hole on the top, the recorder can be muted. We designed our first mute mechanism by placing a solenoid above the recorder and fixing a piece of card board at the end.



Ash is in charge of the solenoid, Alice is in charge of the servo, and Grace is in charge of 3D modelling.

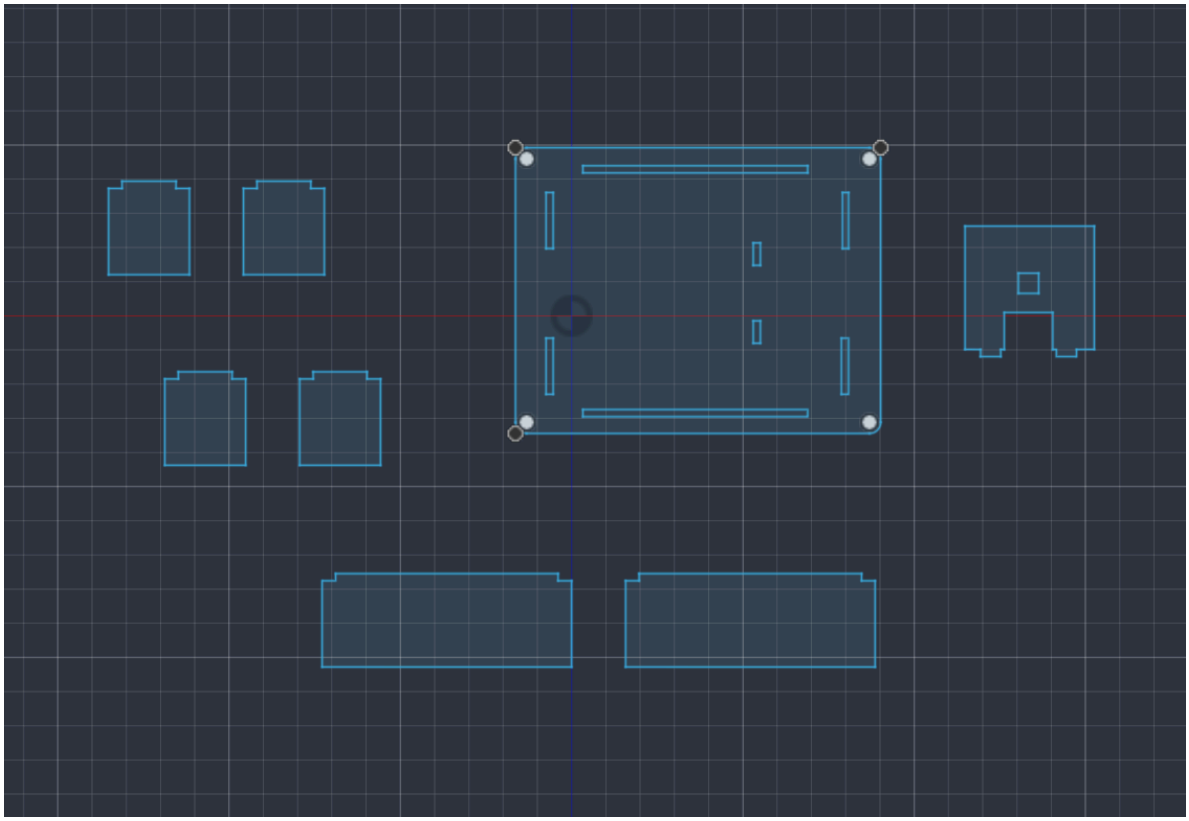

USB
GPIO
ISP
C3
USB
GPIO
ISP
C3


CREATE
MODIFY
ASSEMBLE
CONFIGURE
BROWSER
12032 Blower Fan
Document Settings
Named Views
Origin
Bodies
Body1
Body2
Body4
Sketches

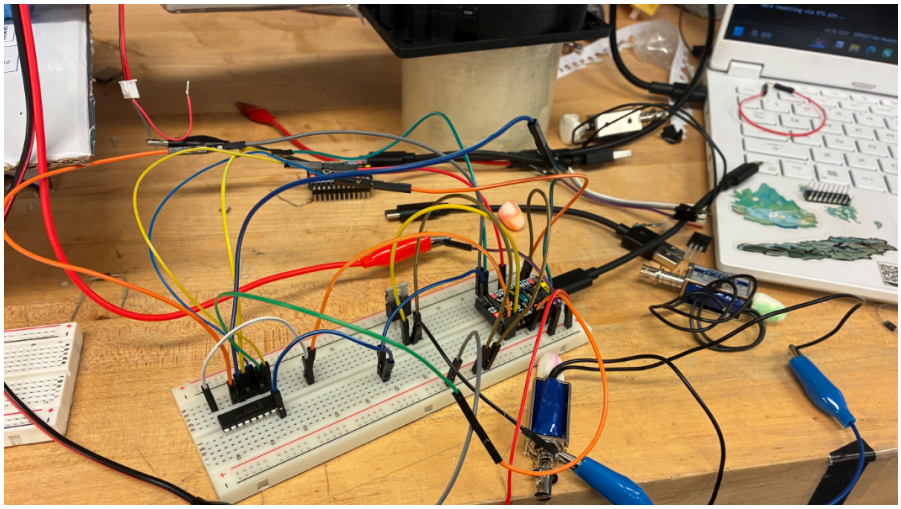
July 24th

port expander, control multiple servos at the same time, control multiple solenoids at the same time, laser cutting the platform, discuss potential control methods

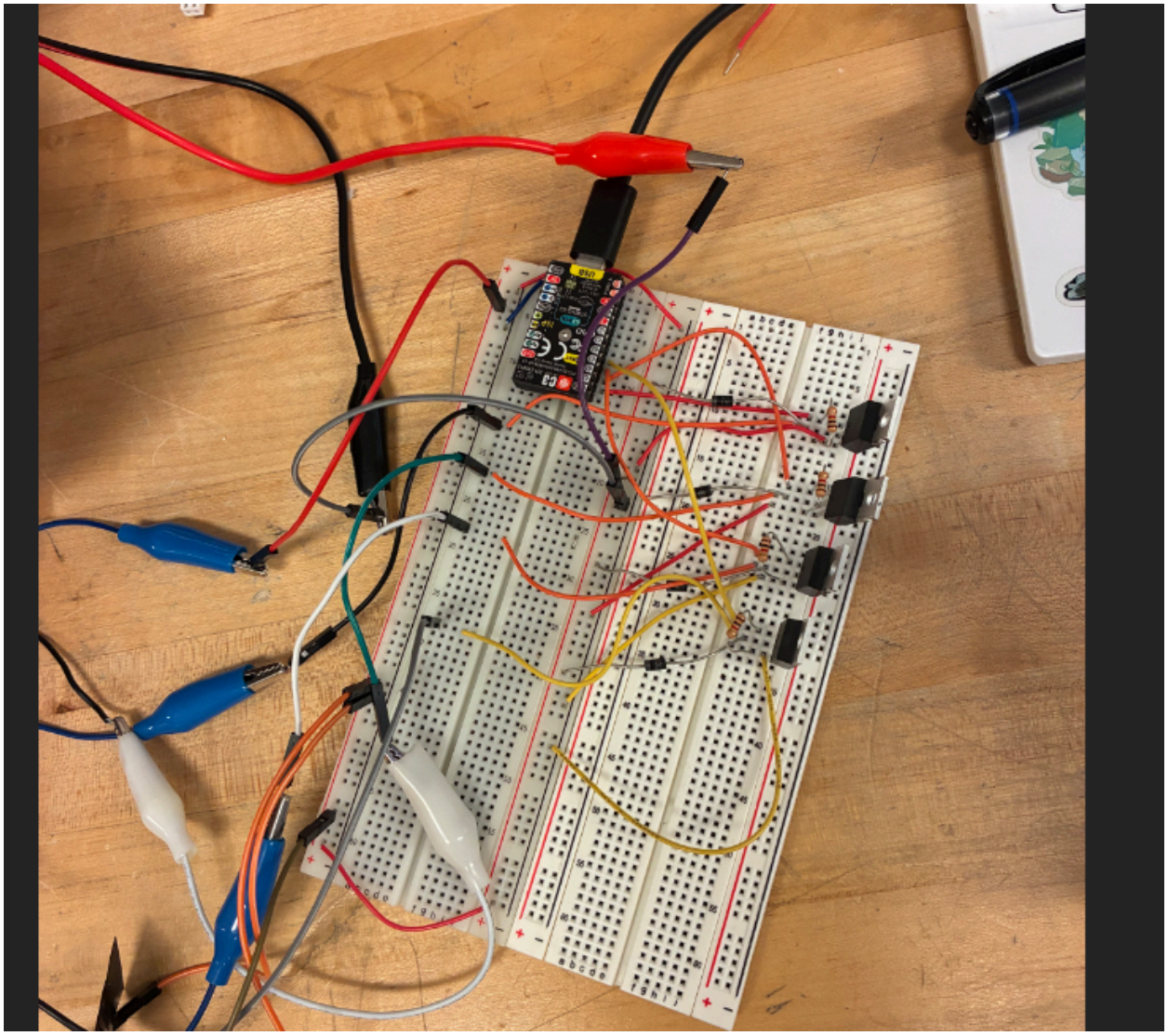
We finished the design of the platform and laser cutted the components.



As there are only 9 safe GPIOs in a single ESP32 C3, we decided to try using the port expander to put separate servo and solenoid together.



But there are some problems with the communication between ESP32 and the expander. Nick helped us out by checking the pattern of the signal. The signal looks fine, but we still cannot build up the connection between these two components. At last, we decided to use two ESP32s instead.



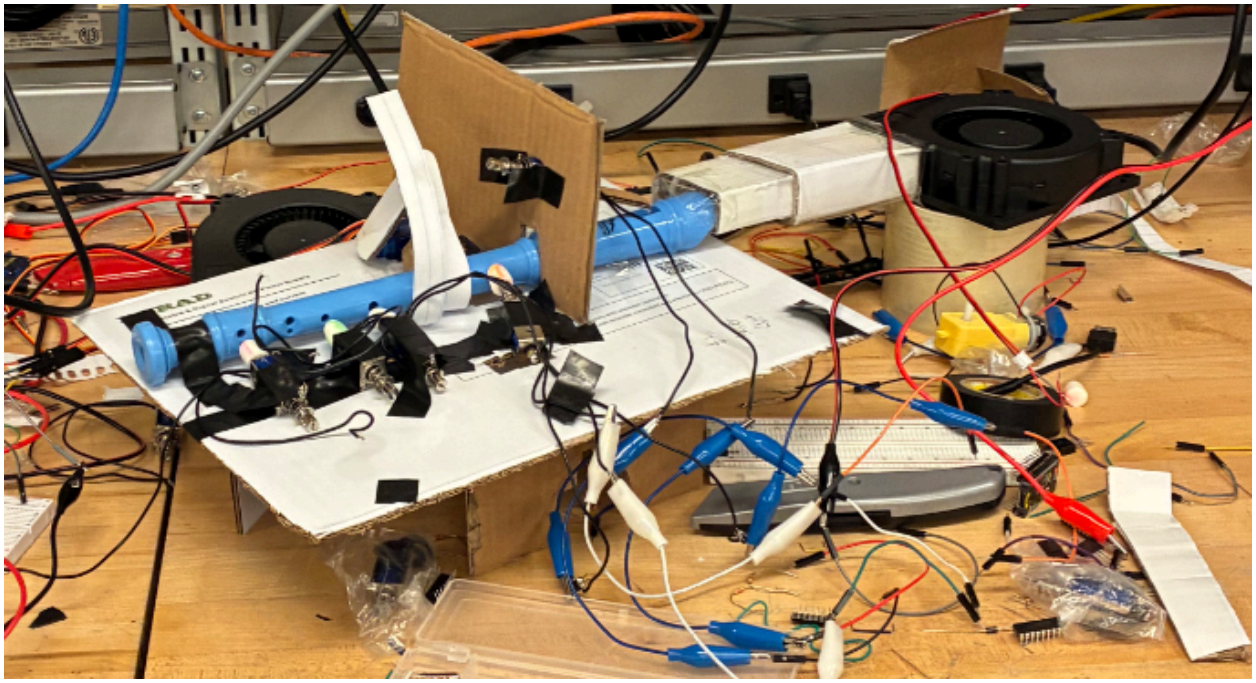
We also talked about using a conductor micro controller and making it send a signal to another executive micro controller. However, considering the complexity and delays, we agreed on synchronizing these two steps on one single ESP32.

https://drive.google.com/file/d/138Wr8Y-UWuRaY3aqTxZ6zoO89OxZqNsq/view?usp=sharing

https://drive.google.com/file/d/1vUotwe8qKST4JiUsCAYhD2rZ_G1571_u/view?usp=sharing

July 25th

After we finished separate functional checks for the servos and solenoids, we began hardware assembly on one ESP32 board. I measured each GPIO-pin voltage before wiring the actuators. Some surplus wire was cut off to fit our installation space. All lead-out wires of the solenoids got soldered, then I checked every solder spot for shorts and bad welding. I sorted out tangled cables and put name tags on every part, cable and usable pin. Every actuator-pin match was written down in our log. Once everything was wired, I turned on the power and carried out a basic hardware check. All of the above-mentioned work supports later-stage parameter modification and experimental testing.



https://drive.google.com/file/d/1lbEsUm7l3a3Sgy4F_U5DAiTVBBYY9AcM/view?usp=sharing

The recorder system uses **two ESP32 microcontrollers**. One ESP32 controls the note actuators that cover and uncover the recorder holes. It receives MIDI note data and activates the correct servos or solenoids. The second ESP32 controls the mute mechanism, including the mute solenoid and mute servo. It mainly responds to rest commands in the music. Because both ESP32 boards receive the same musical data, they do not need to communicate directly with each other.

The ESP32 uses **PWM, or Pulse Width Modulation**, to control servos. PWM rapidly switches a digital signal between ON and OFF. By changing the percentage of time that the signal remains ON, called the **duty cycle**, the controller can adjust the average power delivered to a device. The microcontroller sends pulses with different widths, and each pulse width represents a target servo angle. This allows the servos to move the finger mechanisms and mute mechanism to specific positions.

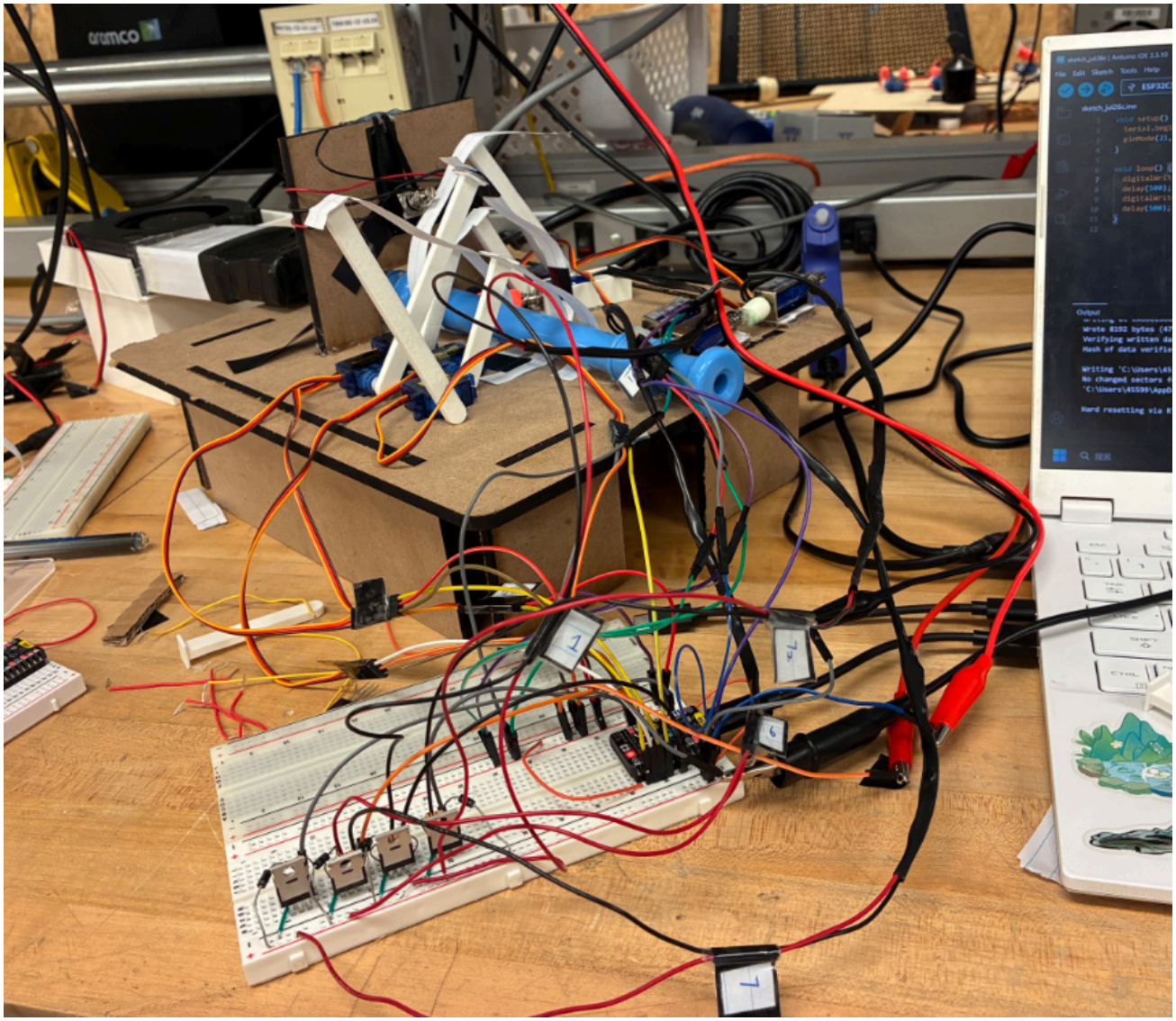
ESP32 cannot directly power the solenoids or the blower because its output pins provide only a small amount of current. Therefore, **MOSFETs** are used as electronic switches. It sends a low-current control signal to the MOSFET gate. The MOSFET then allows the larger current from an external power supply to flow through the solenoid or fan. This protects the ESP32 while allowing it to control high-power components. For PWM-controlled devices, the MOSFET switches on and off at high speed according to the PWM signal. This makes it possible to control the average power while keeping the ESP32 electrically separated from the main load current.

Resistors are used around the MOSFET gate to keep the control signal stable and prevent the MOSFET from turning on accidentally. A small gate resistor can also reduce sudden current spikes from the ESP32 pin. Because solenoids are inductive loads, a **flyback diode** should be connected across each solenoid. When the solenoid turns off, it can generate a high reverse-voltage spike. The diode safely redirects this energy and protects the MOSFET and ESP32.

The circuit uses three external power sources:

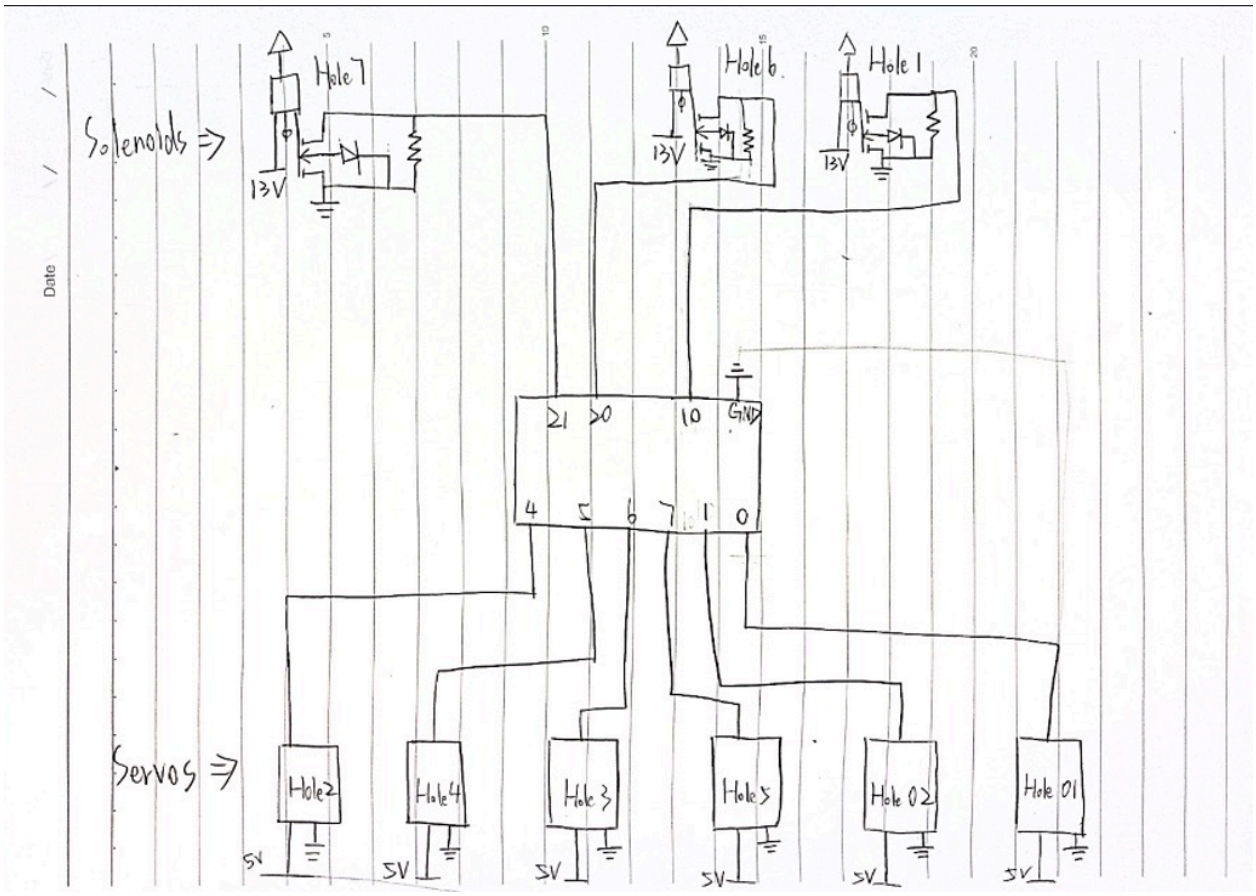
- one for the blower fan(22V);
- one for the solenoid actuators and mute solenoid(14V);
- one for the actuator servos and mute servo(5V).

Separating the power supplies reduces voltage drops and electrical noise when several actuators operate at the same time. However, the ESP32 boards and external power supplies must normally share a **common ground** so that all control signals have the same voltage reference.


drumco

July 26th

Once the hardware assembly was completed, we moved on to subsequent debugging work. We adjusted relevant operating parameters for servos and solenoids again and again, carried-out repeated actuator tuning to stabilize movement response. We ran comparison tests and experimented with different materials to block the holes effectively. We used paper stripes, clay, plastic foam, ear plugs, and sides of ear plugs. Different holes are suitable for different materials. We chose paper stripes for hole 0,2,3,4,5, and ear plugs for hole 1, and sides of ear plugs for hole 6 and 7. Paper stripes enable us to perform notes that require half hole covering. Since hardware compatibility varied, we also tried the ESP32-S2 board during debugging. After comparing their signal output stability and USB-port performance, we picked ESP32 C3 for our automatic-playing prototype at last.



https://drive.google.com/file/d/1_ueXrZnBtLAQM2_dkki1-c1GnF8rnuZl/view?usp=sharing

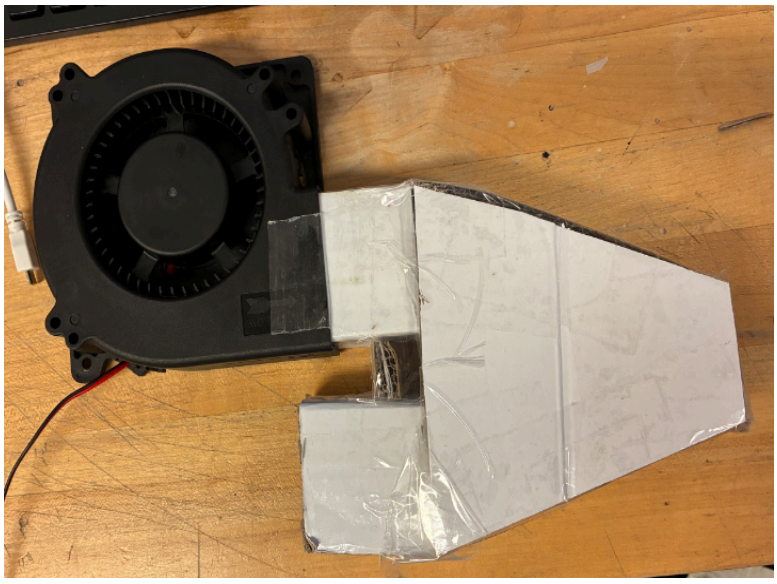
July 27th

We loaded and parsed data from the MIDI file so the hardware could play individual musical notes in sequence. During the test, we spent time altering the placement of servos and solenoids and testing multiple-way mounting schemes on the bracket. Every rearrangement received a quick playback test to locate mechanical jams and obstructions, with practical installation adjustments carried out until the mechanical playback prototype functioned properly.

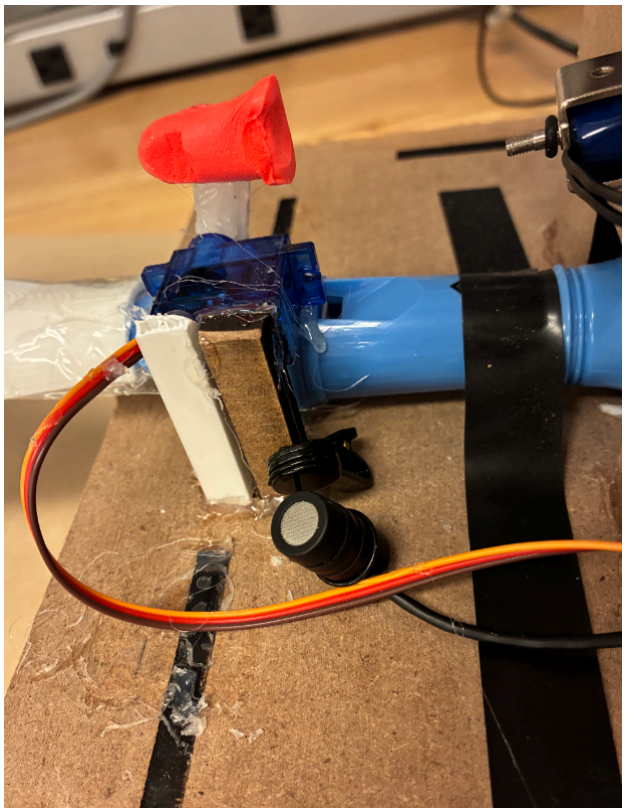
https://drive.google.com/file/d/1W1pVPpJUgKgMG1F5V-zzImbaVaSwM_tk/view?usp=sharing

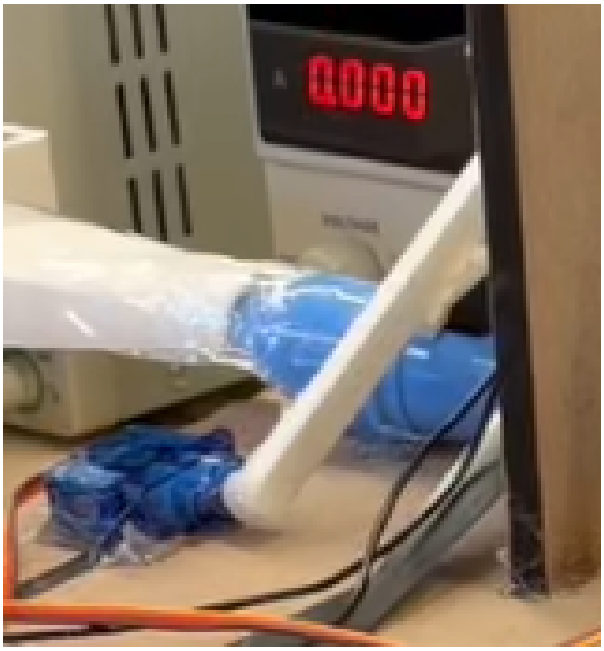
July 28th

We swapped out and trialled multiple-sized tunnel structures for airflow adjustment. We tried out tunnels of different shapes and also made one that can fit two fans, which provides a larger volume of air flow. The two-fans mechanism allows us to play lower notes, but as a soprano group, we decided to stick to high notes that the tenor recorder group can't play. We also tested different mute methods.



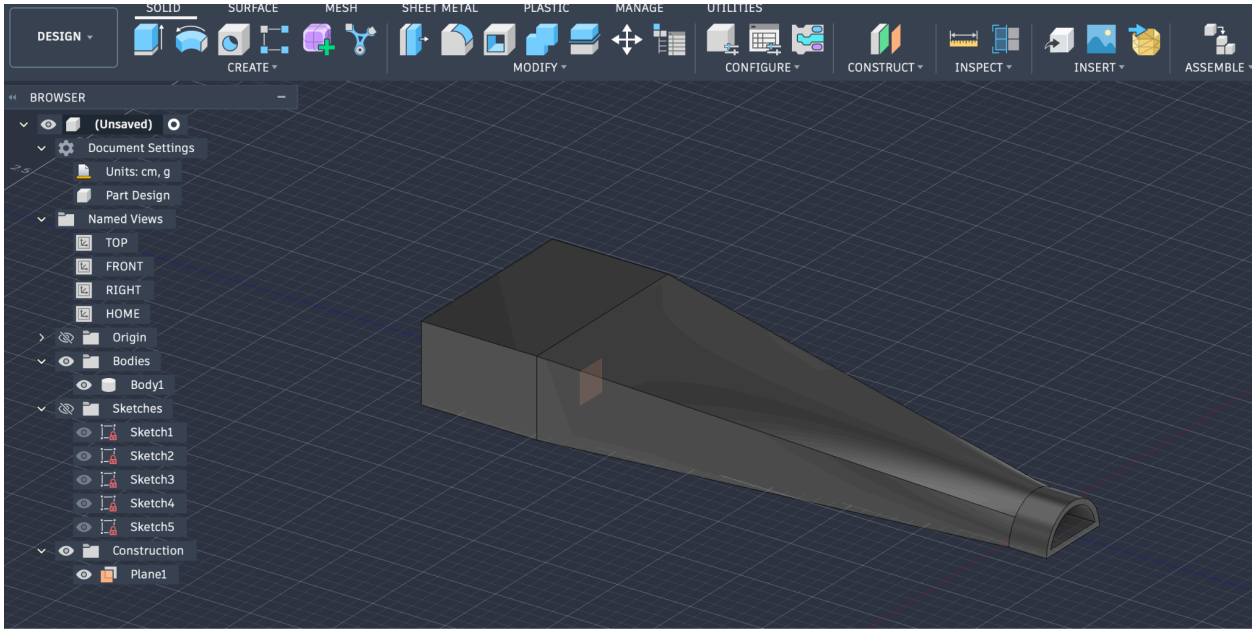
https://drive.google.com/file/d/1t2jbgWokuc83N-hM6dzT-4NMkh0dmU6x/view?usp=sharing

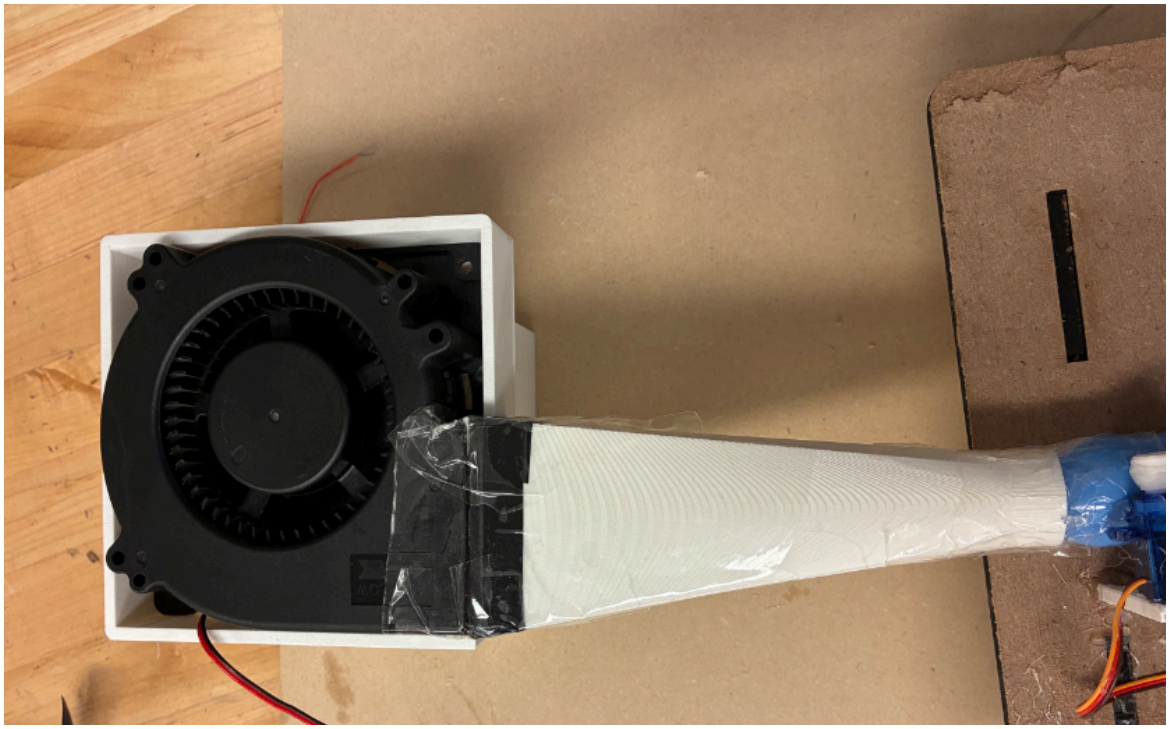




July 29th

We manufactured air-flow tunnels via 3D-printing with adjusted structural dimensions. Afterwards, we did the tuning again like the previous time. Today we spent more time on coding to connect the whole setup to WiFi.





July 30th

https://drive.google.com/file/d/1eRcobdYKR_cZj1cfx7wA3waoR2RLhIyp/view?usp=sharing

This is our final project performance. After receiving the command, it started to play Moondance.