Animatronic Raspberry Pi Project

Project Overview:

This is my first animatronic project, powered by a Raspberry Pi Zero 2 W paired with a Waveshare Servo Driver HAT capable of controlling up to 16 servos.

The system utilizes 7 micro servos to animate 3D-printed parts that simulate the movement of eyes, eyelids, a jaw, and a neck.

A 4x4 matrix keypad serves as the main input interface and each button controls a different component, while one button activates all components simultaneously.

Power is supplied by an external 7.4V Li-ion rechargeable battery. The system also includes two switches:

- Rocker switch: Acts as a hard power switch, connecting or disconnecting the main circuit.
- Momentary push button: Initiates a safe software shutdown for the Raspberry Pi.

⚠️ Important: Always press the momentary shutdown button before toggling the rocker switch to safely power off the system and avoid SD card corruption.

3D Printing the Parts:

- **Printer Used:** Bambu Lab P1S
- **Software:** Bambu Studio
- **Model Source:** Pre-existing model (modified in Bambu Studio)
- **Modifications Made:**
  - Added standoffs for the Raspberry Pi
  - Added a compartment for the Li-ion battery
  - Created space for the 4x4 matrix keypad
  - Designed an add-on mount for the two switches
  - Added base legs

Electronics Setup:

Components:

- Raspberry Pi Zero 2 W; flash your SD card with Raspberry Pi OS
- Waveshare 16-Channel PWM Servo Driver HAT (12-bit resolution, I²C interface)

- 7x SG90 Micro Servos
- 4x4 Matrix Keypad (16 buttons)
- 7.4V 900mAh Li-Po Rechargeable Battery (PH2.0 connector, USB charging)
- 1x Rocker Switch (hard power)
- 1x Momentary Push Button (soft shutdown)
- JST PH2.0 Connector Wires (male and female)

Setup Steps:

1. **Enable I²C on the Raspberry Pi**
   Go to raspi-config → Interface Options → I²C → Enable, then reboot the system.
2. **Connect the Servo Driver HAT**
   Attach the Waveshare Servo Driver HAT securely to the Raspberry Pi GPIO header.
3. **Connect the Servos**
   Plug each servo into its assigned channel on the HAT (see the channel mapping below).
4. **Prepare and Test the Scripts**
   Create a directory to store all Python scripts. Test each servo individually to confirm functionality before full assembly (details expanded below).
5. **Wire the Keypad**
   Connect the 4x4 matrix keypad to the designated GPIO pins (see the keypad pin mapping section).
6. **Assemble the 3D-Printed Parts**
   Once wiring and servo testing are complete, begin assembling the printed components.
7. **Install the Switches**
   - Rocker Switch: Wire between the battery and the Servo Driver HAT's VIN terminal (main power).
   - Momentary Button: Connect between GPIO11 and GND (for the shutdown trigger).

⚠️ Setup steps for assembling the 3D-printed parts will be included, just ignore the CyberBrick content and focus on the build itself.

Software Setup:

Programming:

- Language: Python 3.11
- Environment: Pipenv (for virtual environment management)

Python Libraries:

- RPi.GPIO
- adafruit-circuitpython-servokit
- time
- subprocess
- smbus
- gpiozero

Scripts:

- Test scripts – Used to test each movement individually. E.g. eyes, jaw, neck.
- Movement scripts – Execute randomized movement patterns.
- Calibration script – Important to run before attaching servo horns to the servos.
- keypad.py – Handles input from the 4x4 keypad, with each button corresponding to a specific movement.
- shutdown-press-simple.py – Runs the safe shutdown process when the momentary button is pressed

⚠️ The scripts will be provided.

Servo Driver HAT Channel Mapping:

Component	Servo Channel
Left Eyeball	1
Left Eyelids	0
Right Eyeball	4
Right Eyelids	5
Jaw	15
Neck (Horizontal)	10
Neck (Vertical)	11

⚠️ Calibration:

- Before attaching the servo horns, run the calibration.py script. Then attach the horns in their neutral positions as shown in the reference images.

Keypad Pin Layout:

The 4x4 matrix keypad is connected to the Raspberry Pi using eight GPIO pins—four for rows and four for columns. These can be reassigned as needed, but the following configuration is used in this build:

Function	GPIO Pin
Row 1 (L1)	25
Row 2 (L2)	8
Row 3 (L3)	7
Row 4 (L4)	1
Column 1 (C1)	12
Column 2 (C2)	16
Column 3 (C3)	20
Column 4 (C4)	21

Note: If you modify the pin assignments, make sure to update the corresponding variables in your keypad.py script to match your wiring configuration. The same can be said for the channel mapping on the Servo Driver HAT.

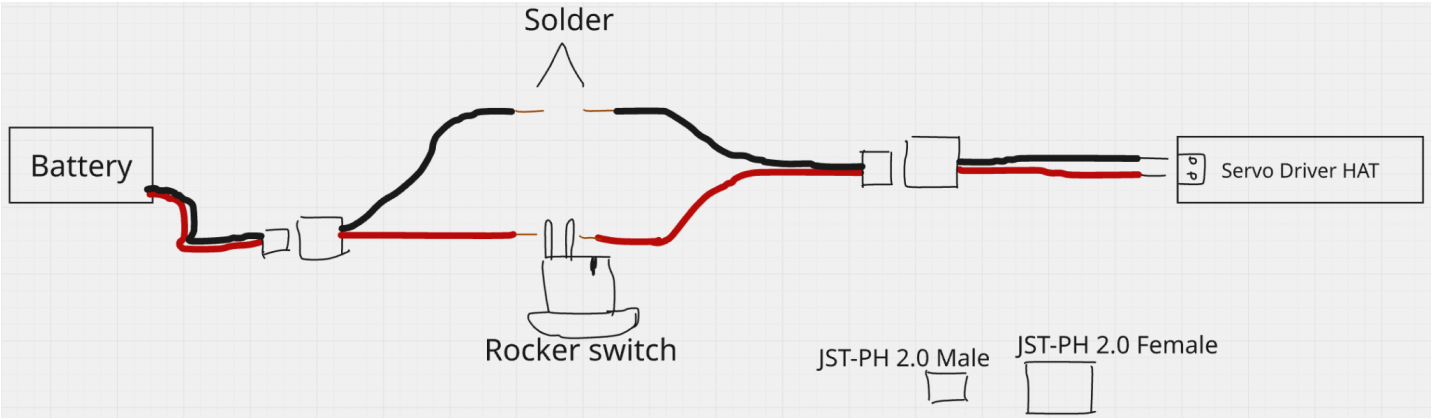
Power Management & Switching:

Soft Power (Software Shutdown):

- Momentary push button connected to GPIO11 and GND; can use jumper wires
- Raspberry Pi shutdown triggered by the button
- The button runs shutdown-press-simple.py to safely shut down the Pi

Hard Power (Main Switch):

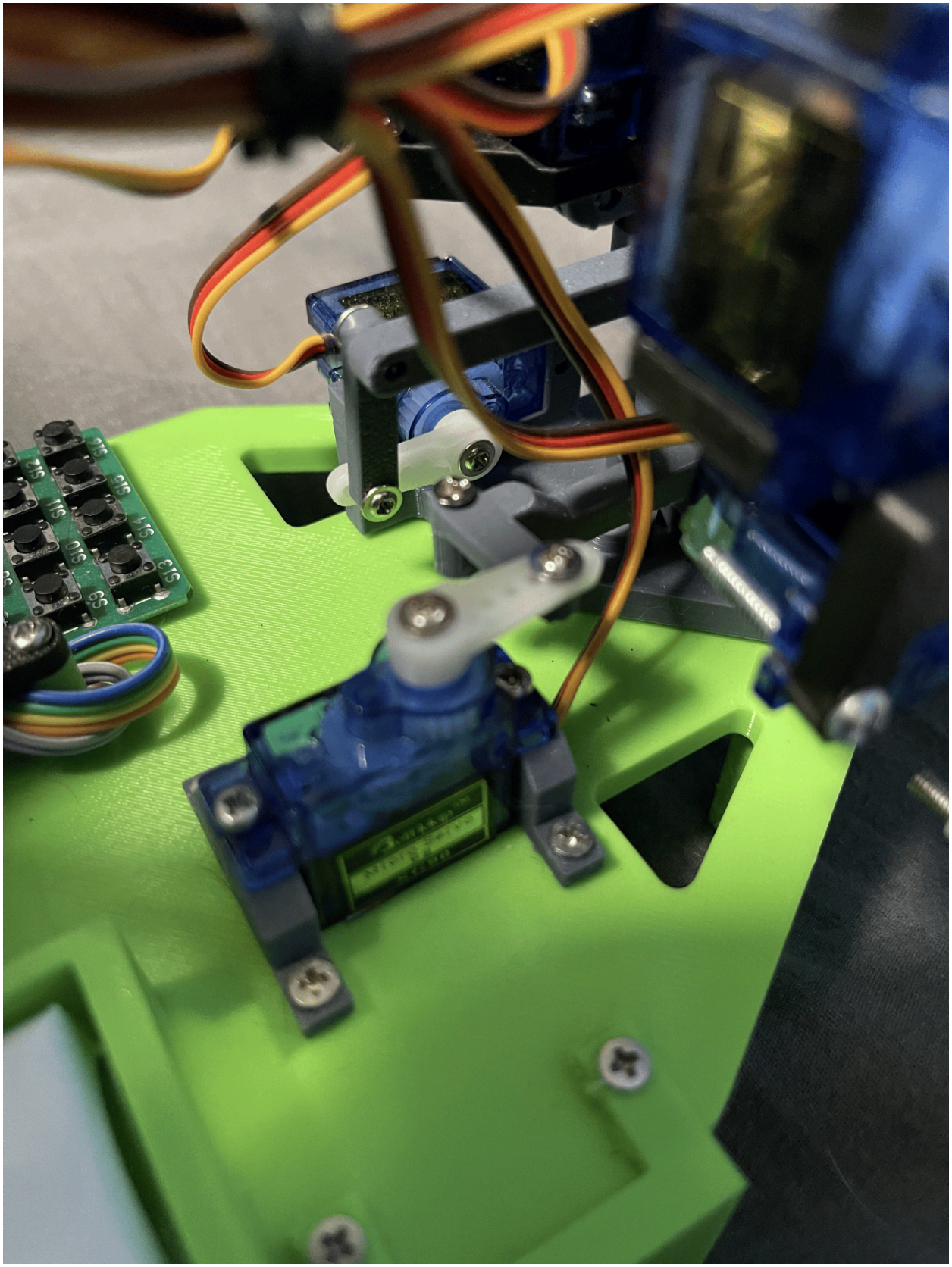
- Rocker switch wired between the battery and HAT VIN
- Use JST PH2.0 connectors for modular connections without cutting or soldering battery leads.
- Use a **female JST-PH 2.0 connector** for the HAT's VIN input. Next, prepare a **male JST-PH 2.0 connector** to mate with the HAT's **female connector**, and a **female JST-PH 2.0 connector** to mate with the battery's **male connector**. You should now have two pairs of wires—one red and one black in each pair. Connect the red wires through the rocker switch: either solder or crimp them onto the switch's two prongs—one prong connects to the load device (the Pi HAT), and the other connects to the battery. **Visual representation:**

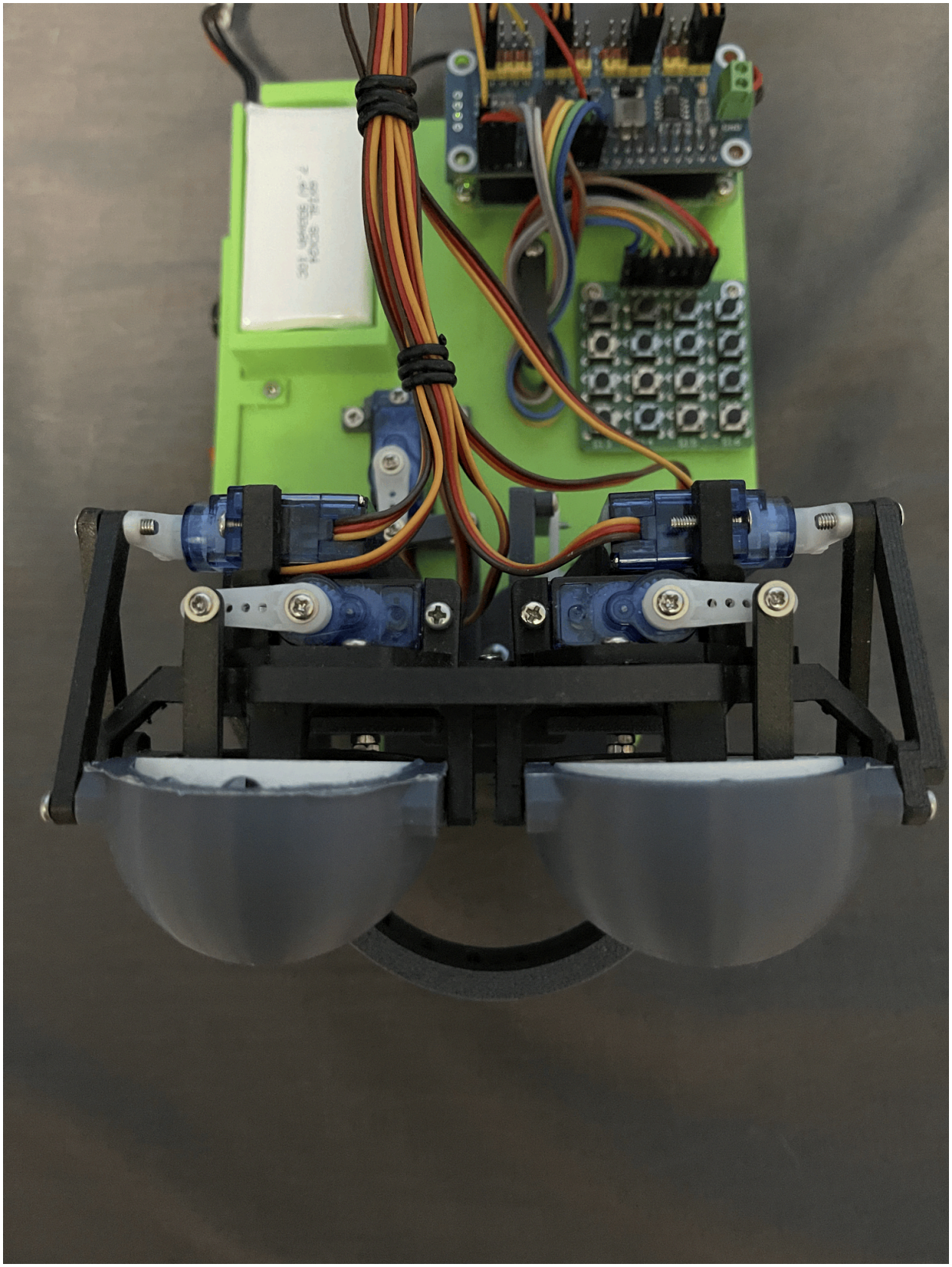


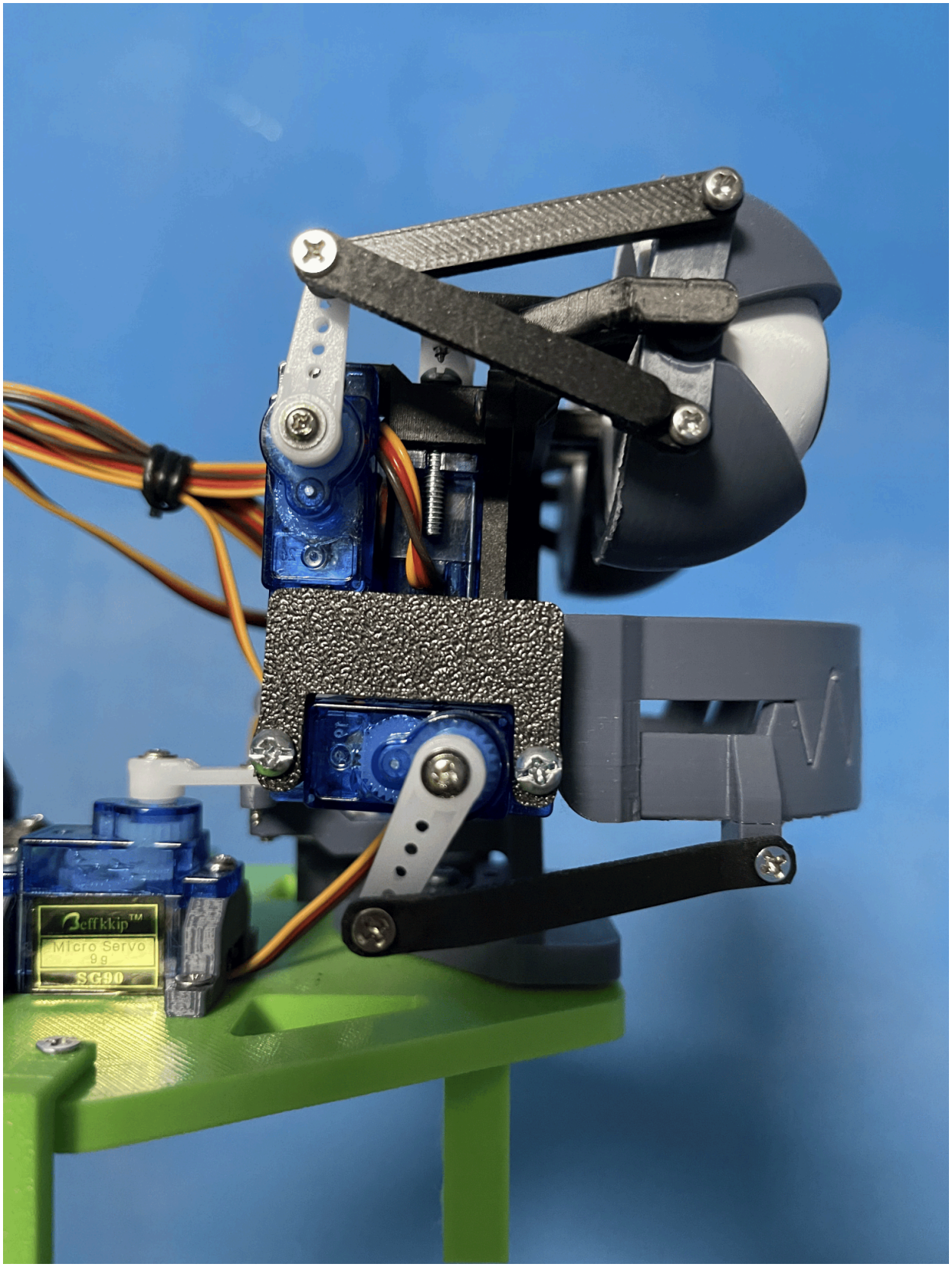
Note: If you don’t want to include the rocker switch, you can simply skip that step and directly connect the battery’s male JST-PH 2.0 connector to the HAT’s female JST-PH 2.0 connector.

⚠️⚠️⚠️ Please do this with the Raspberry Pi turned off and the battery unplugged. Also, make sure that no metal parts are touching each other.

Servo horns alignment:


S12
S13
S14
S15
S16
S9
S10
S11




Micro Servo
9g
SG90

- You can make any necessary angle adjustments, especially if you notice unhealthy servo noises such as grinding or whirring.

Addendum: Script Locking with flock

- To prevent multiple scripts from running simultaneously and interfering with each other, I implemented **flock**. This ensures that only **one instance of a script can run at a time**, blocking the execution of the same or other movement scripts until the current one finishes.
- Sources:
  - https://linux.die.net/man/1/flock
  - https://www.tutorialspoint.com/unix_commands/flock.htm

Addendum: Shared Servo Control Package

- All scripts utilize a **shared Python package** that centralizes the servo control logic. This package initializes the Adafruit ServoKit with 16 channels and defines all servo assignments and movement functions for the eyes, neck, and jaw.
- By keeping these functions—such as lid control, gaze direction, neck movement, and jaw positioning—in one place, any **adjustments to servo angles** or behavior can be made quickly and consistently across all scripts.