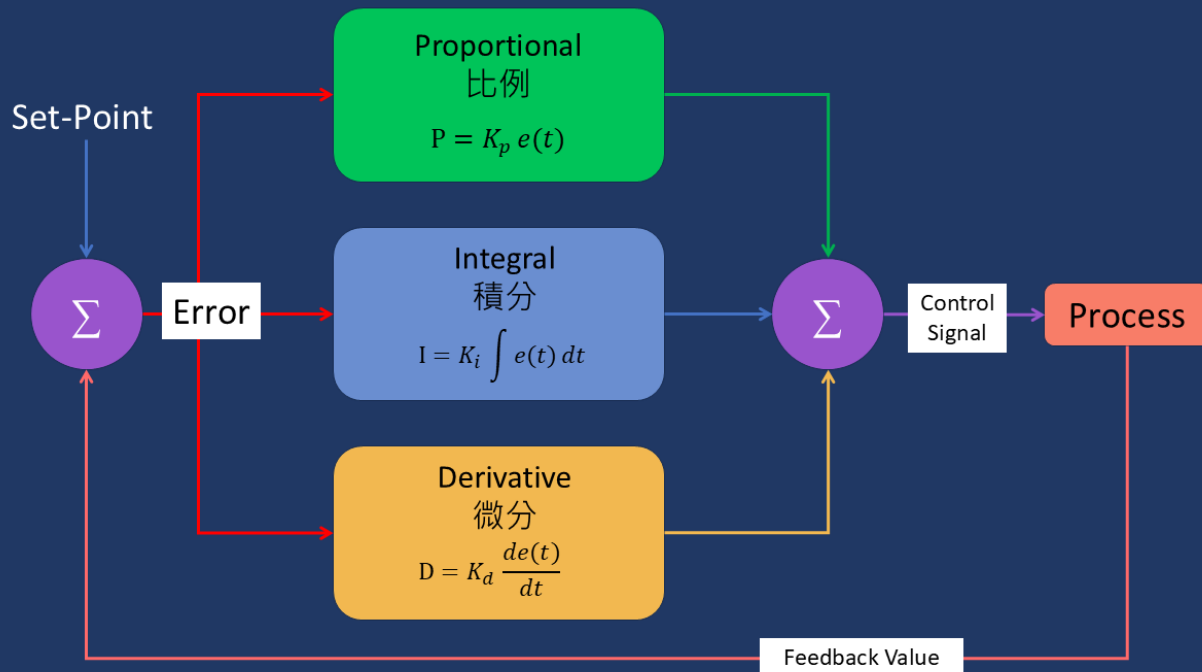


Workshop on Designing and Making of Self-Balancing Device

「設計及製作自平衡裝置」工作坊



Au-Yeung Fu
STEM Education Centre
29 May 2026

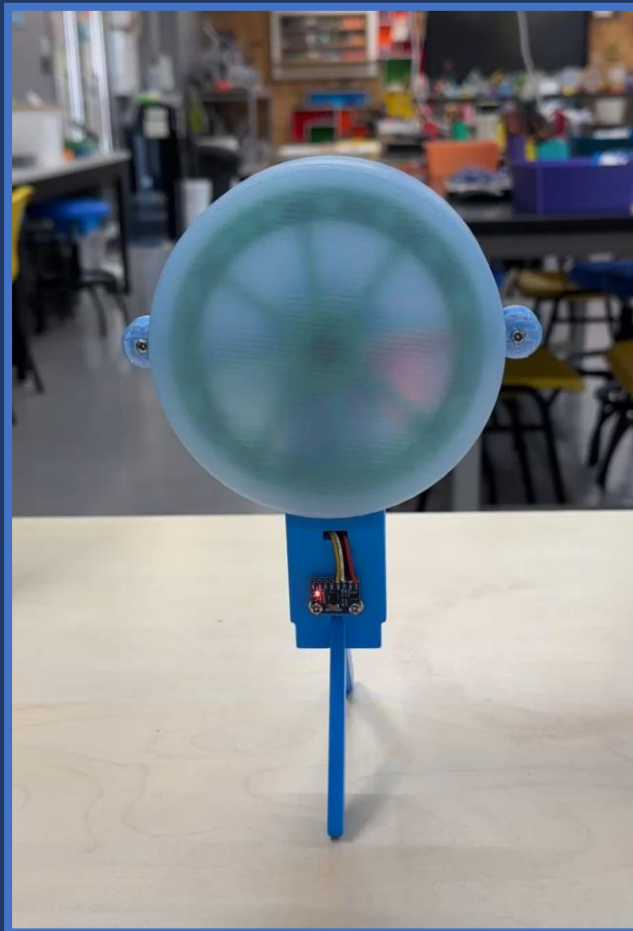
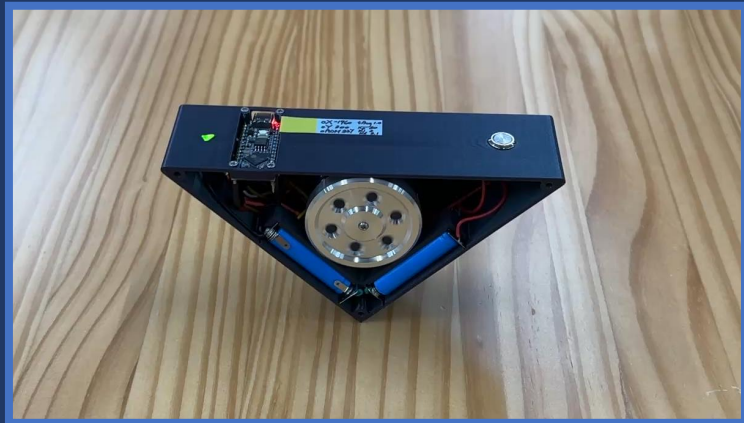
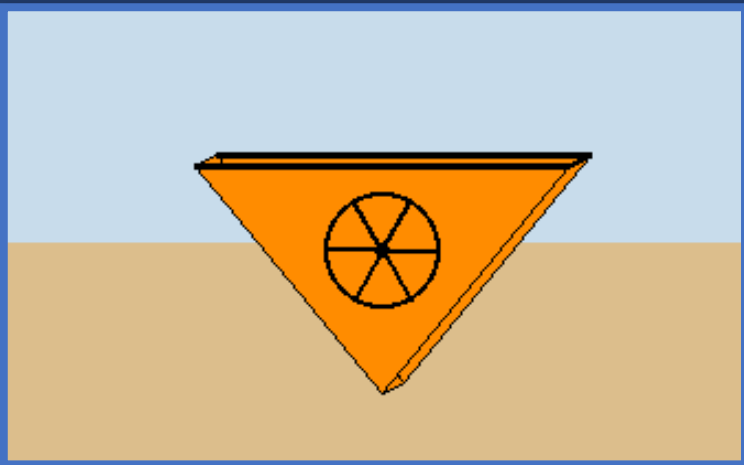
Workshop on Designing and Making of Self-Balancing Device

「設計及製作自平衡裝置」工作坊

Time 時間	Content/Activity 內容/活動
14:30 – 15:15	Introduction to the concepts of feedback control and optimisation technology, including the operating principles of Proportional-Integral-Derivative (PID) control, and how to incorporate learning tools such as self-balancing devices in facilitating STEAM projects in secondary levels. 簡介反饋控制與優化技術的概念，包括比例-積分-微分(PID)控制的運作原理，及如何應用自平衡裝置等的學習工具於中學促進STEAM專題研習
15:15 – 16:35	Hands-on activities: • Assembling the mechanical parts of the device • Wiring the electronic components of the device • Programming the device • Testing the functionality of the device and optimise its performance 動手做活動： • 組裝裝置的機械部分 • 裝置電子元件的接線 • 為裝置進行編程 • 測試裝置的運作效能並作出優化
16:35 – 16:50	Discussion on the application of artificial intelligence technology in optimising the performance of self-balancing devices 討論如何應用人工智能技術優化自平衡裝置的性能
16:50 – 17:00	Q&A 問與答

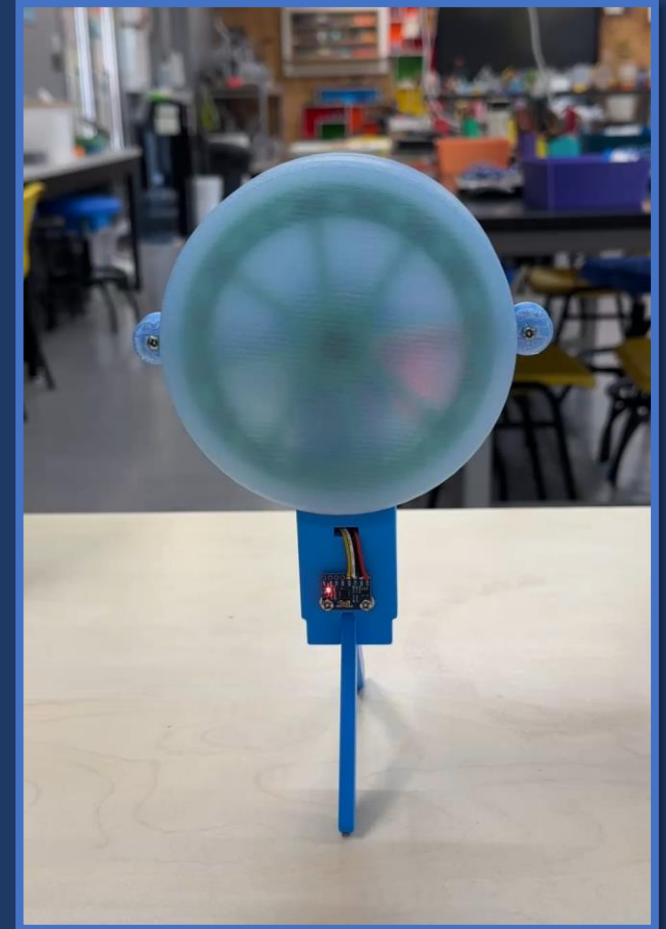
What are self-balancing devices:

A self-balancing device can be an electromechanical system that aims to automatically maintain stability and restore itself to a stable equilibrium position, even when subjected to external forces or intentional disturbances (like being pushed).



What are self-balancing devices:

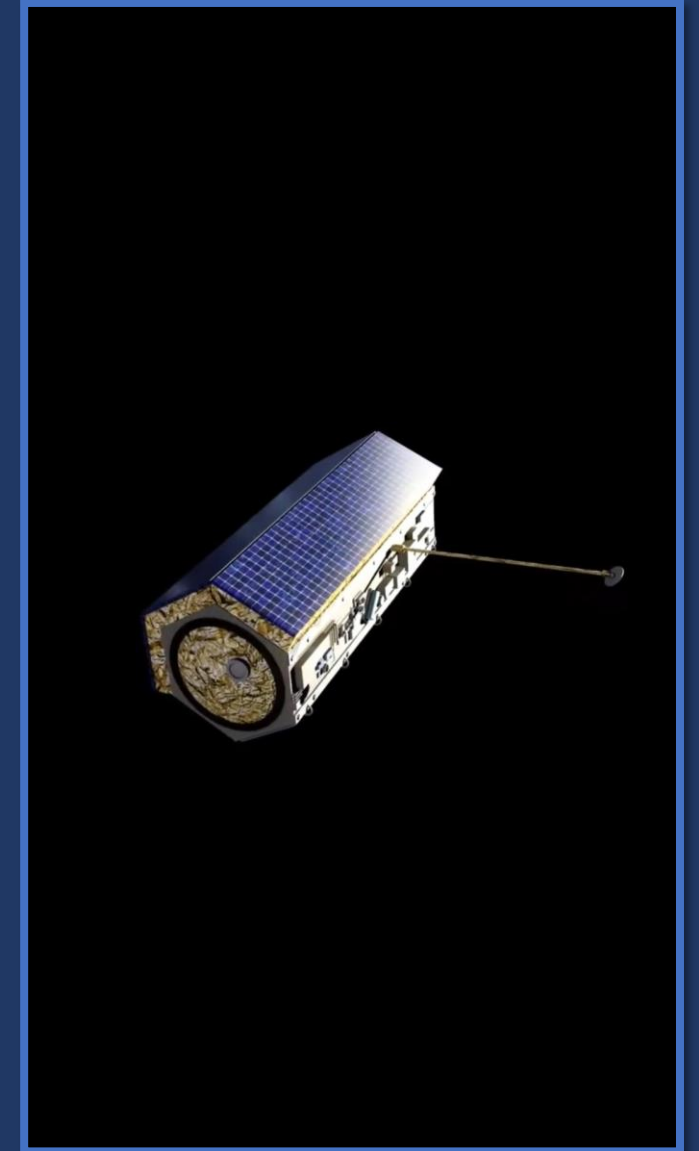
- The core principle behind these devices is **feedback control**.
- They continuously **measure their current orientation**.
- They **calculate the precise forces** required to counteract any deviation from the desired (stable) state.
- Actuators are the components responsible for applying physical force or torque to the device.
- The essential components include:
 - **Accelerometer and Gyroscope Module (Sensor)**: detects tilt and rotational rate of the main body
 - **MCU (Processor)**: executes the control algorithm
 - **Reaction Wheels (Actuator)**: the MCU computes the necessary corrective torque, which the reaction wheels produce internally to stabilize the device's main structure.



What Is a Reaction Wheel?

A reaction wheel is a type of flywheel used primarily by spacecraft and satellites for precise attitude control and stabilization. Because they use electricity (usually from solar panels) rather than propellant, they significantly reduce the weight needed for fuel, allowing more room for payloads.

By **accelerating** or **decelerating** an electric motor-driven wheel, the spacecraft rotates in the **opposite direction** to maintain its orientation.

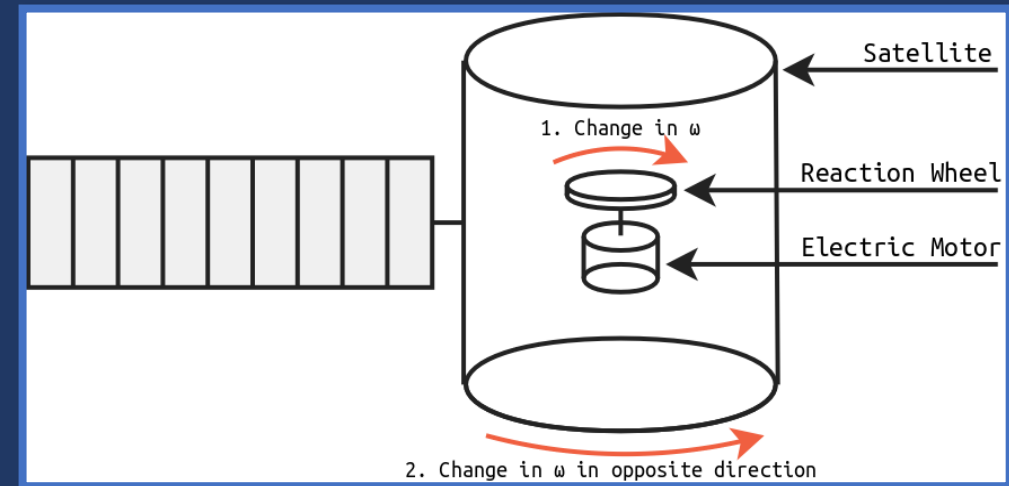


What Is a Reaction Wheel?

By **accelerating** or **decelerating** an electric motor-driven wheel, the spacecraft rotates in the **opposite direction** to maintain its orientation.



https://diqn32j8nouaz.cloudfront.net...r_momentum.gif



<https://charleslabs.fr/en/project-Reaction+Wheel+Attitude+Control>

Key Aspects of Reaction Wheels

Operational principle:

- The construction for the device is simply a disc mounted on an axis, commonly powered by some kind of electric motor.
- They operate based on Newton's third law—applying torque to the wheel produces an **equal and opposite** torque on the connected system.
- Reaction wheels are good in the use case where you would want stable and precise rotation in an axis.

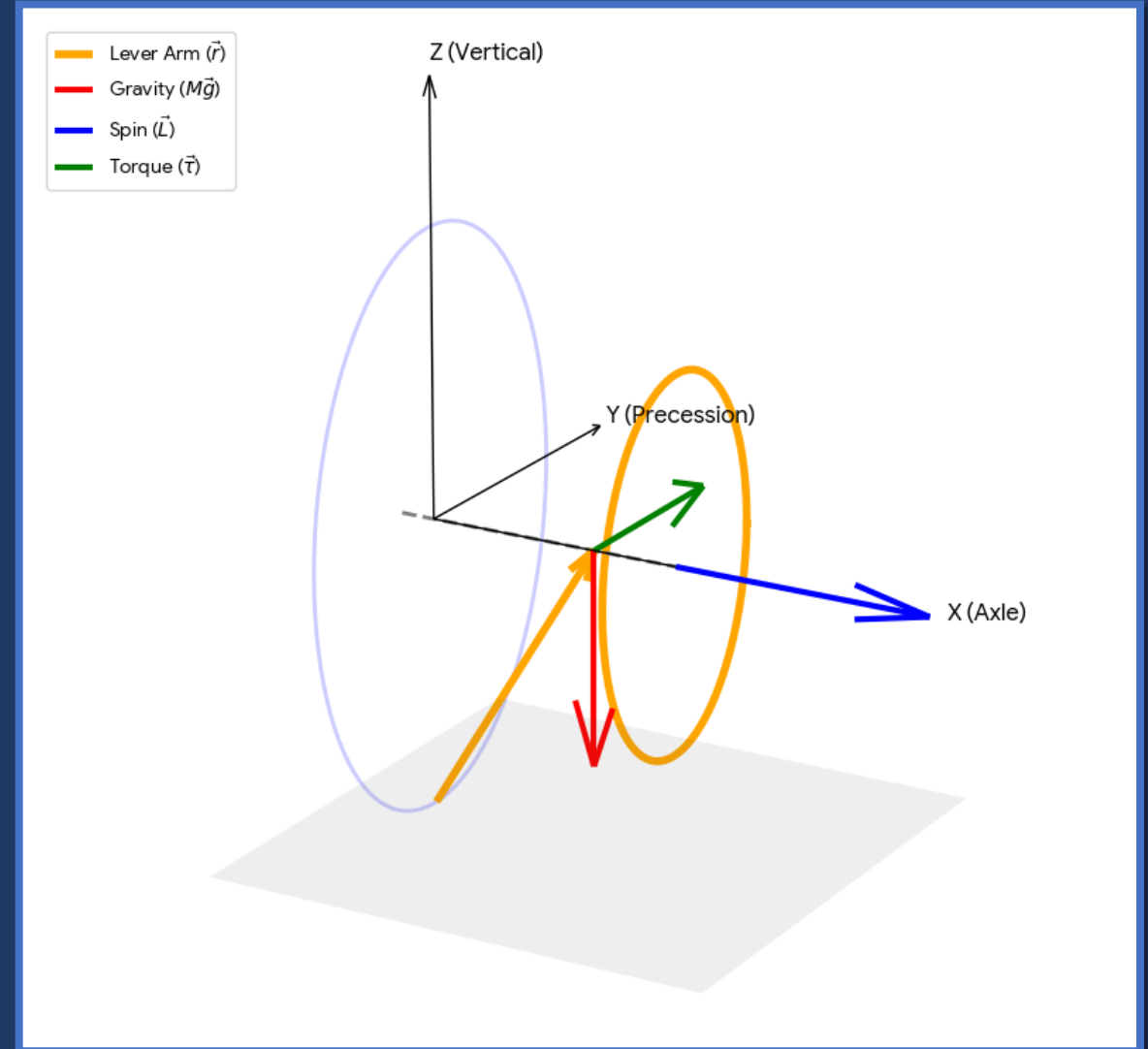
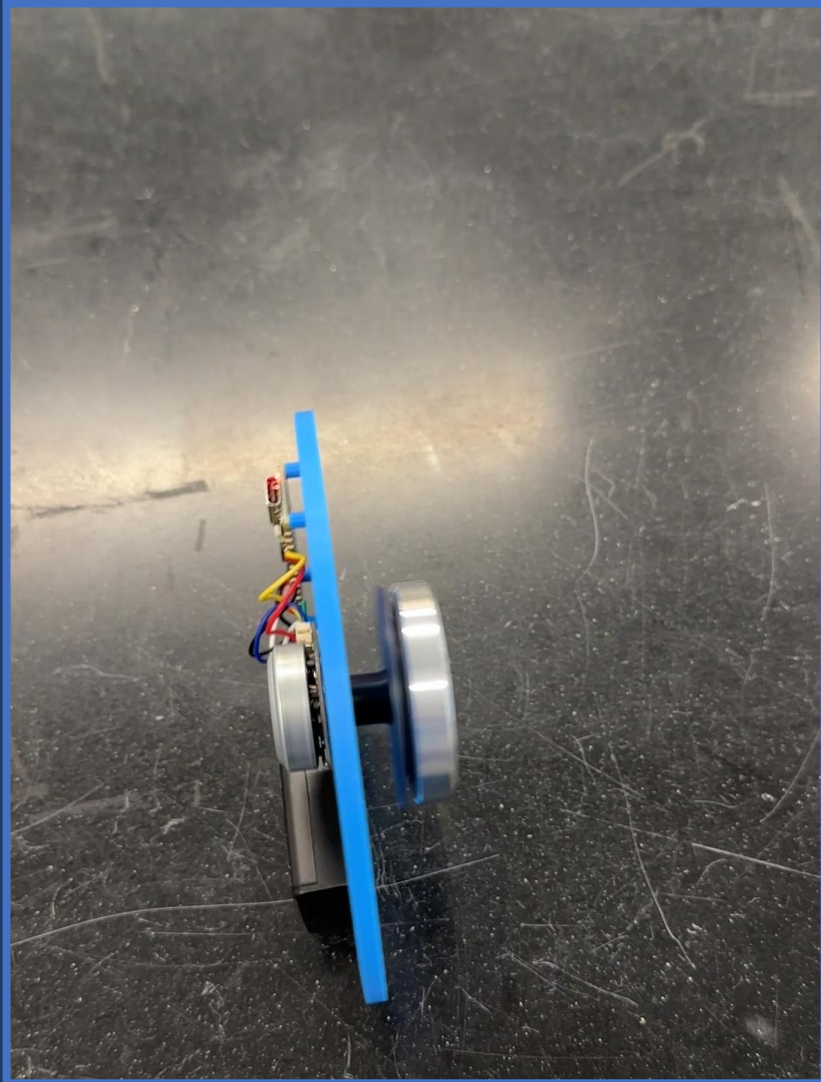
Saturation:

Reaction wheels have a maximum rotational speed. Once this limit is reached, the reaction torque they can provide **drops to zero**.



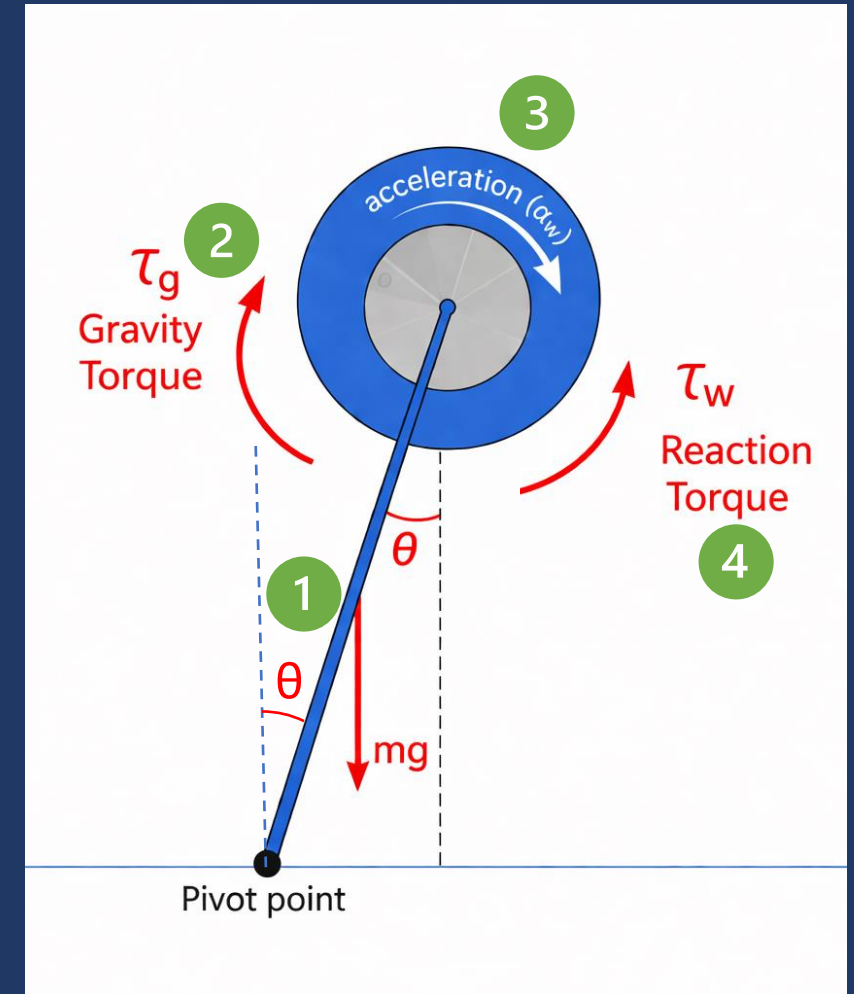
Demonstration

Demonstration of Angular Momentum Conservation at work in maintaining single-wheel balance



Key physical principle applied to the self-balancing device

1. Device tilts to the right ($\theta > 0$)
2. Gravity torque τ_g
 - Will **increase the tilt**
 - Direction: **clockwise** about the pivot
 - This is what makes the system **unstable**
3. Reaction wheel start to rotate and accelerate
 - It generates torque on the body opposite to its angular acceleration.
4. Reaction wheel torque τ_w
 - To **counteract the tilt**
 - Direction: **anticlockwise**
 - This torque helps **restore stability** to the device



Key physical principle applied to the self-balancing device

Center of Mass Consideration:

Although the wheel spins about its own axis, the exchange of angular momentum results in the **reorientation of the entire system around its center of mass**.

Momentum Conservation:

As an internal mechanism, the reaction wheel preserves the system's total angular momentum; it influences only rotational motion, without affecting the system's linear position.

Common confusion (important)

- Wheel **speed** $\neq$ torque
- Wheel **acceleration** $\rightarrow$ generates torque

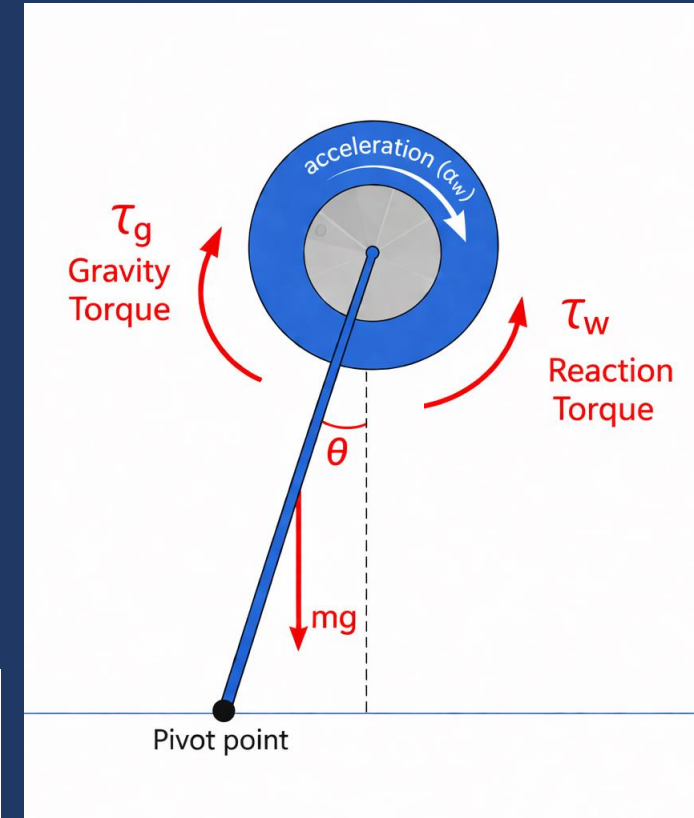
Torque Formula

$$\tau = I\alpha$$

τ is the torque (N·m)

I is the moment of inertia (kg·m²)

α is the angular acceleration (radians/s²)



Comparing Open-Loop and Closed-Loop Approaches to High-Accuracy Motion Control

Open-loop systems run motion commands **without positional feedback** or **verification**.

Examples:

Toaster – set timer

- Sometimes burnt, sometimes undercooked

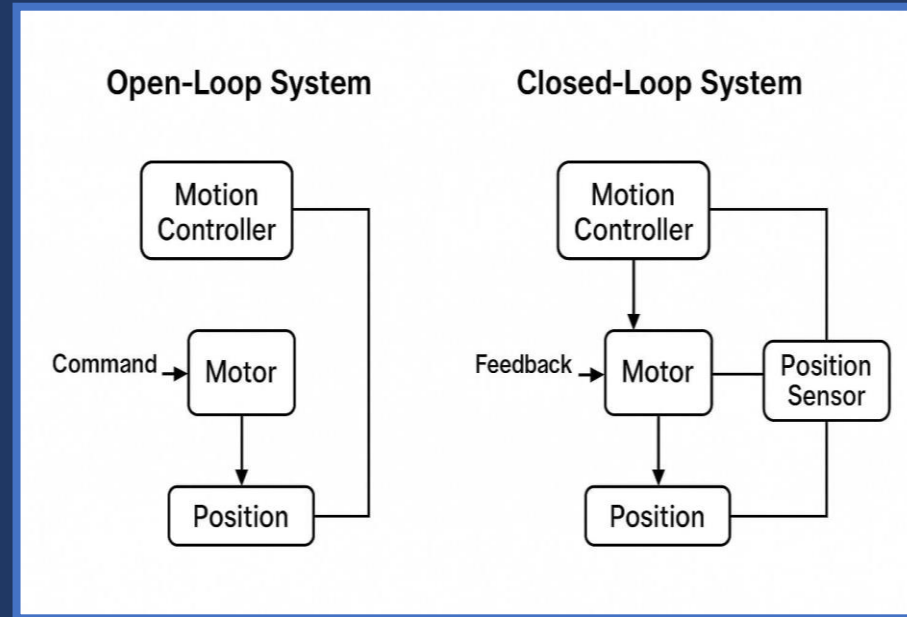
Washing machine - set time to wash

- Doesn't check if clothes are actually clean

Non-adaptive Traffic

light – fixed timing

- Same timing whether busy or empty



Closed-loop systems use real-time sensor **feedback** to continuously track and correct position, ensuring **high positional certainty**.

Examples:

Air Conditioner – set target 23 °C

- Sensor measure actual temperature
- Maintains exactly 23°C

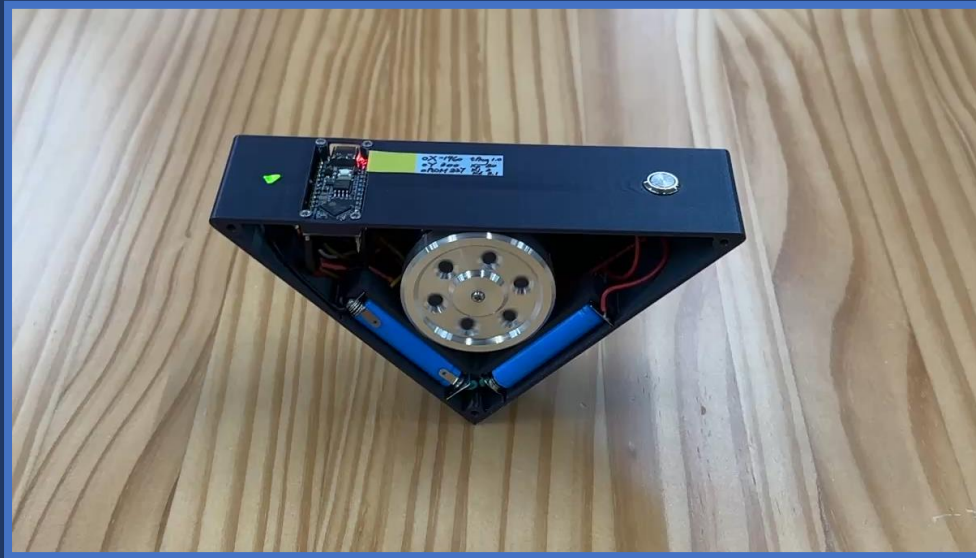
Adaptive Traffic Light

Servo Motor – move to 80°

- Encoder measures actual position
- Accurate positioning

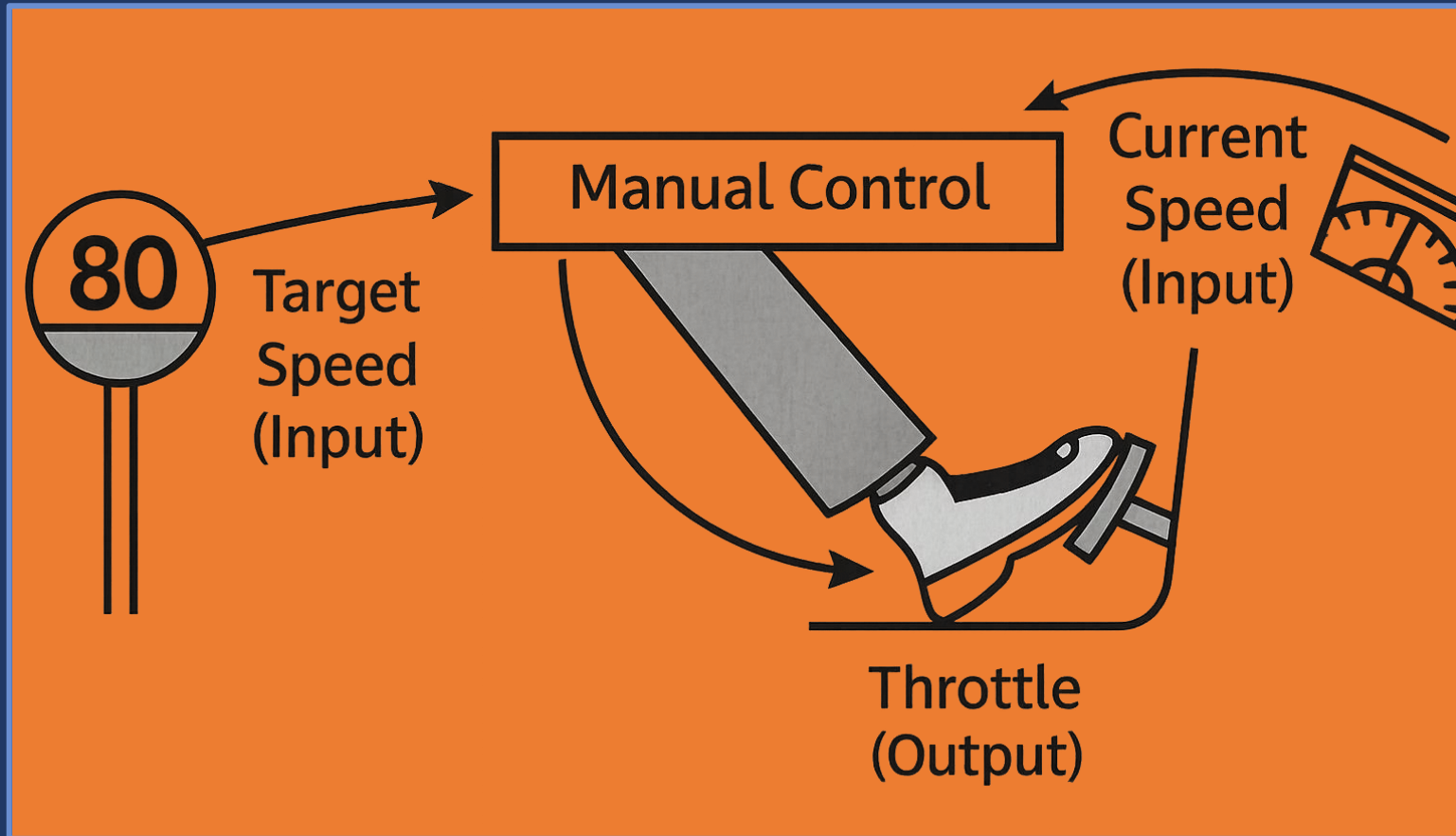
Which control system is best suited to prevent the device from toppling?

Open-loop / Closed-loop ?



An Example of Manual Control

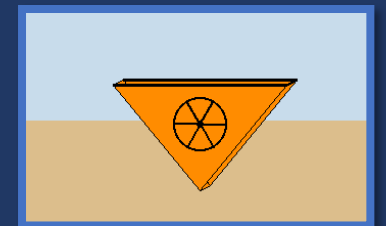
To keep an automobile moving at 80 km/h



- Would maintaining a speed of **exactly** 80 km/h be easy for the driver?
- In general, if the speed occasionally deviates **slightly from 80 km/h**, will it have a significant impact on the driver?

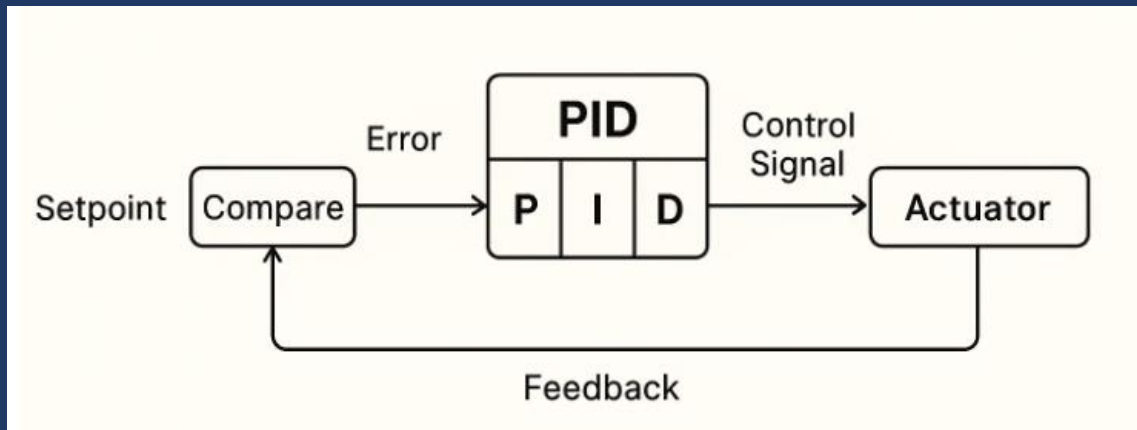
Think carefully:

- Can a device that needs self-balance be operated manually?

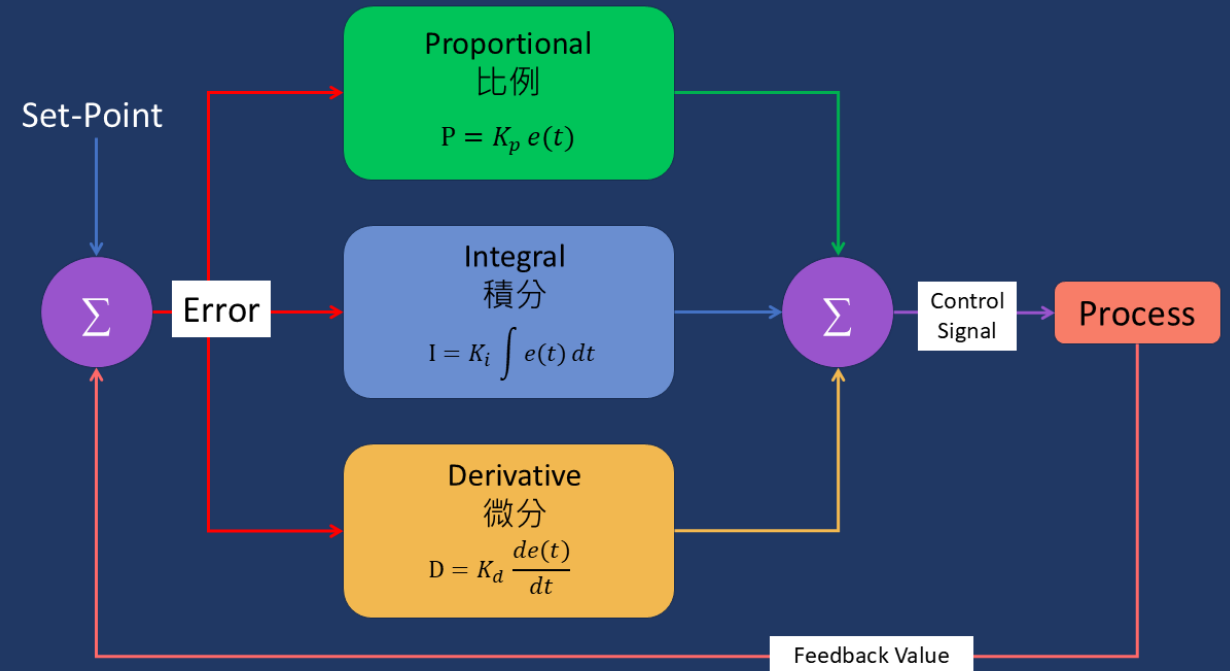


Automatic control with a PID Controller

A PID controller is a popular feedback control system that continuously calculates an error value between a desired setpoint and a measured process variable, and applies a correction based on **Proportional**, **Integral**, and **Derivative** terms. This control method is widely used in industrial automation, robotics and any application requiring stable **closed-loop control**.



PID's job: Find ways to make error zero



Understanding the Three Components: P, I, and D

Analogy: maintaining exactly 100 km/h using cruise control

With a **Setpoint** of 100 km/h, each PID component handles the task differently:

Proportional (P): "The Present"

Adjusts throttle proportionally to the gap between current and target speed.

Action: Ascending the hill drops speed to 70 km/h, creating a 30 km/h error. The controller increases throttle strongly, then eases off as the target is approached.

Problem: P-control alone cannot reach the exact target — the vehicle may settle near it or oscillate around it.

Integral (I): "The Past"

Tracks cumulative error over time to eliminate persistent offsets.

Action: If speed gets stuck at 98 km/h, the I-controller detects the ongoing deficit and gradually adds throttle until 100 km/h is reached exactly.

Problem: Excessive accumulation risks slow response and potential overshoot.

Derivative (D): "The Future"

Evaluates how quickly speed is changing to anticipate future behaviour.

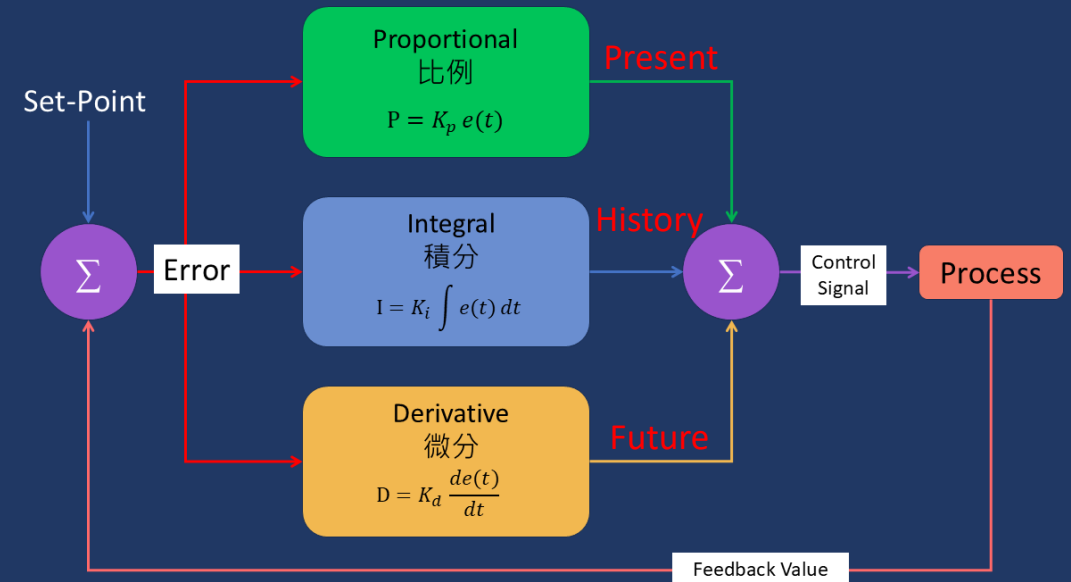
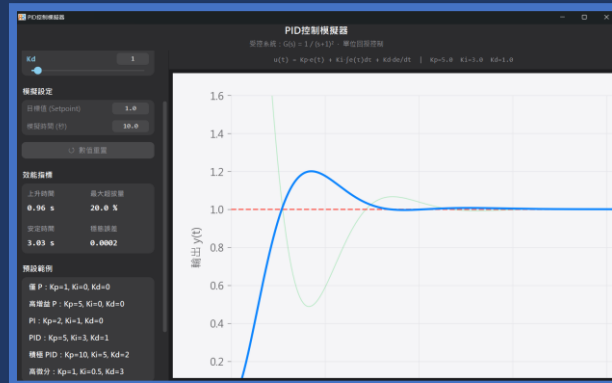
Action: Approaching 100 km/h too rapidly triggers a dampening effect, smoothing the response and preventing overshoot.

Understanding the Three Components: P, I, and D

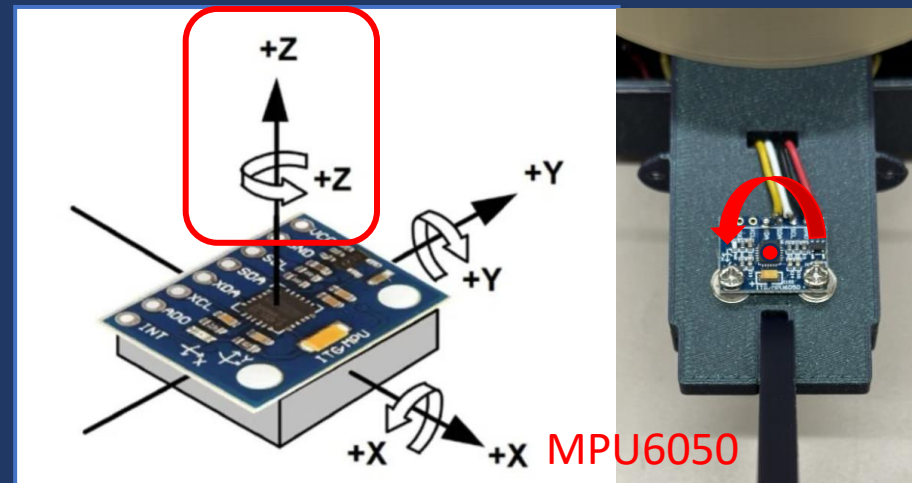
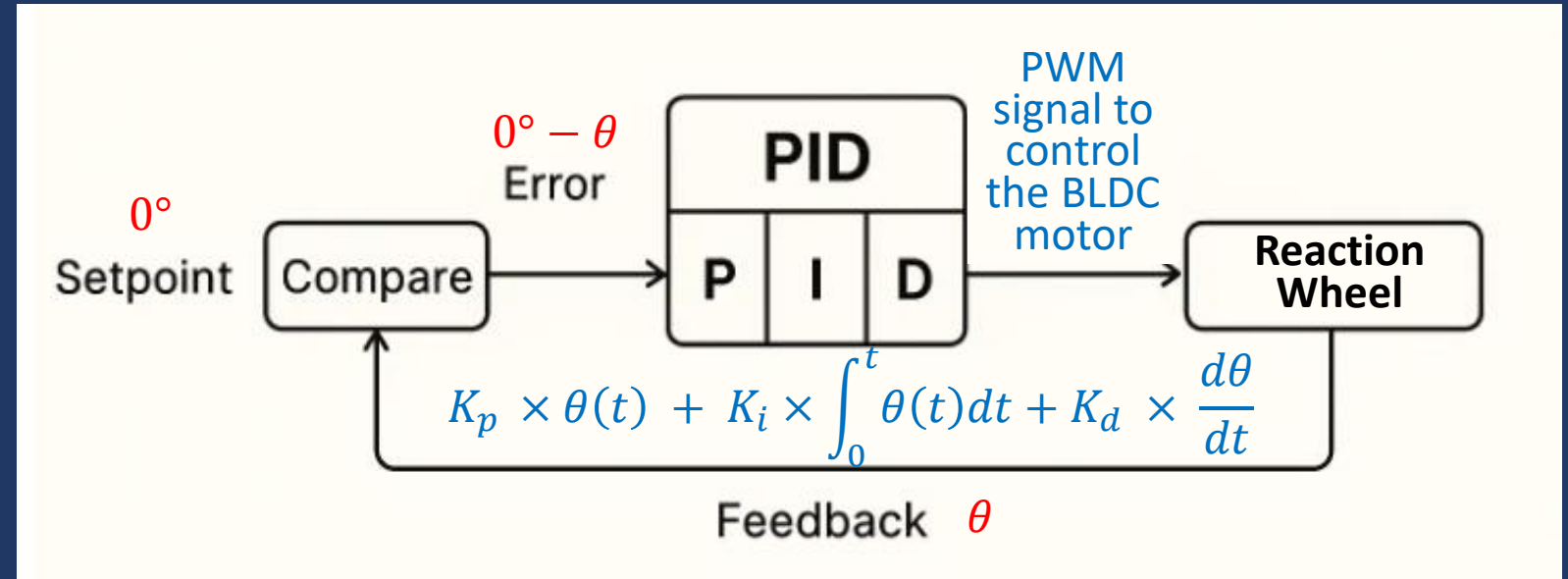
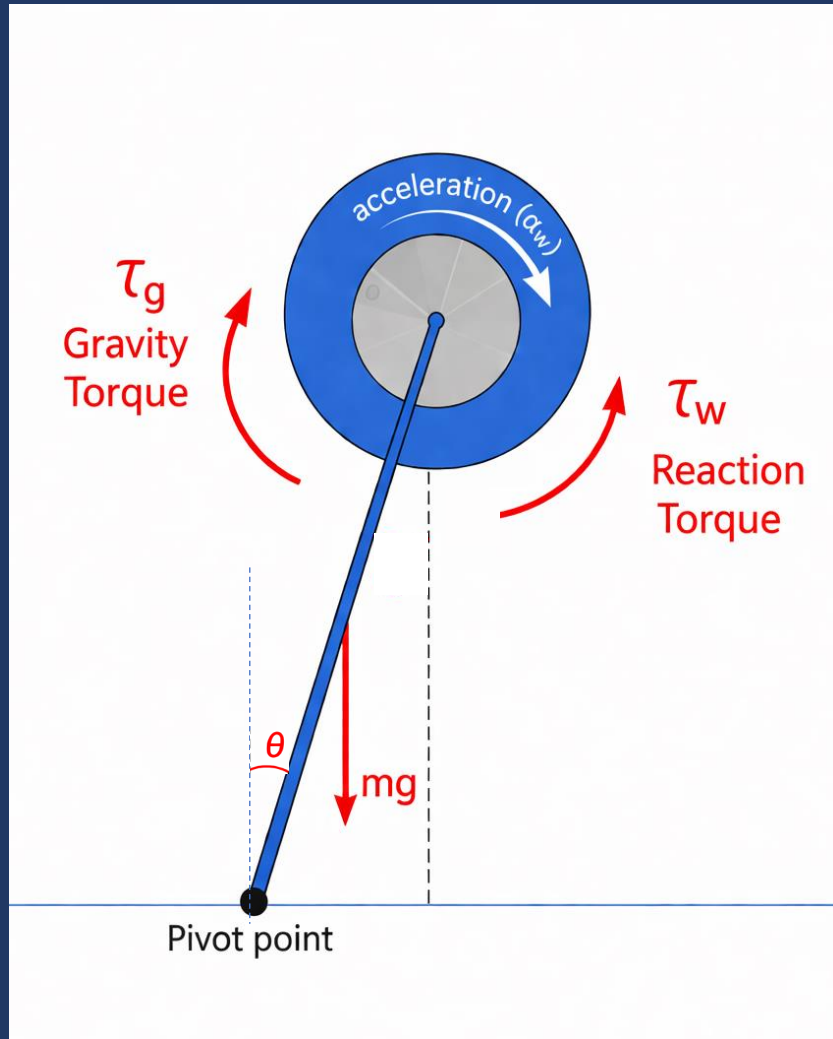
- **Proportional (P):**
Reacts **instantly** to the current error. The larger the error, the stronger the correction.
- **Integral (I):**
Accumulates **past** errors over time to address any persistent offset that the proportional term cannot fully correct.
- **Derivative (D):**
Predicts **future** error trends by evaluating the rate of change. This anticipatory action **helps dampen oscillations** and improve settling time.

These three terms form the core of a **PID control algorithm**, tuned with gain constants:

- K_p
- K_i
- K_d



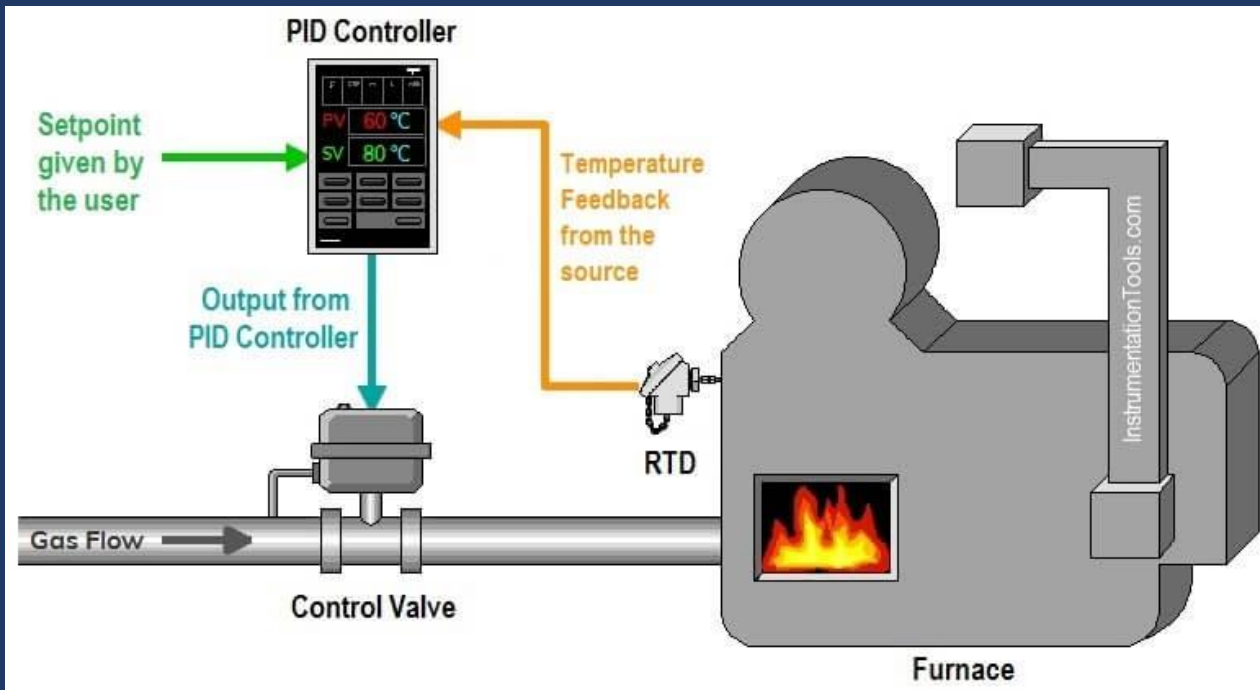
Automatic control with a PID Controller



$\theta(t)$, $\int_0^t \theta(t)dt$ and $\frac{d\theta}{dt}$ can be calculated or estimated using the information obtained from the MPU6050 (IMU sensor)

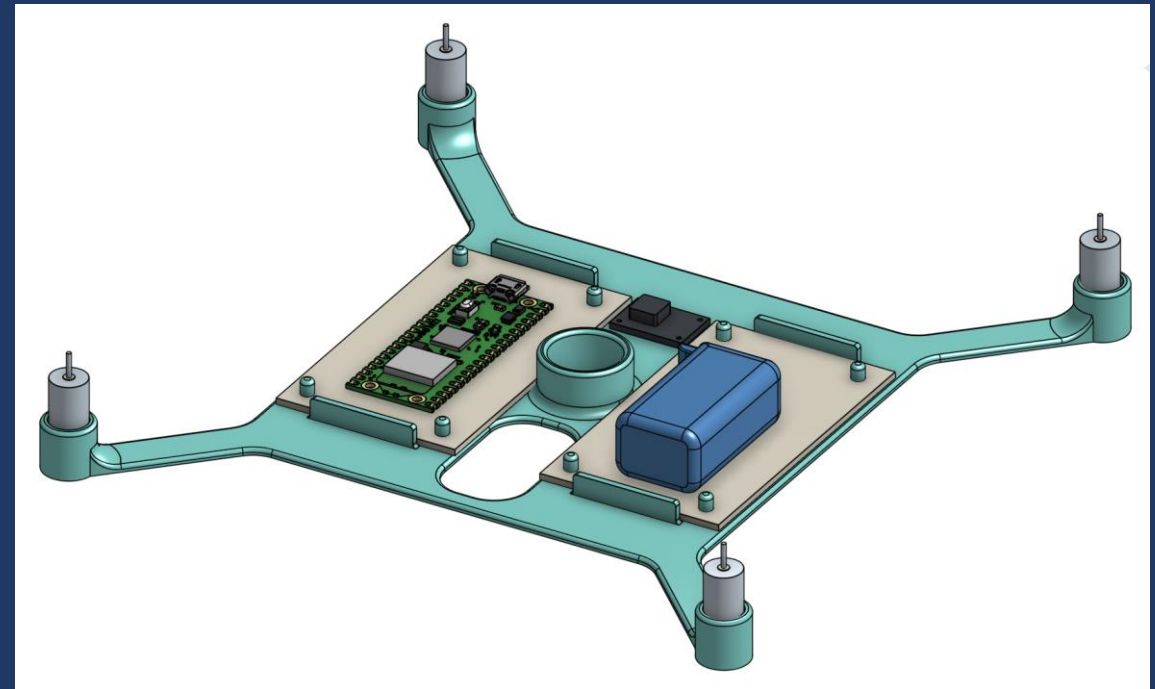
More examples of applications utilizing PID Controllers

Temperature control within a furnace



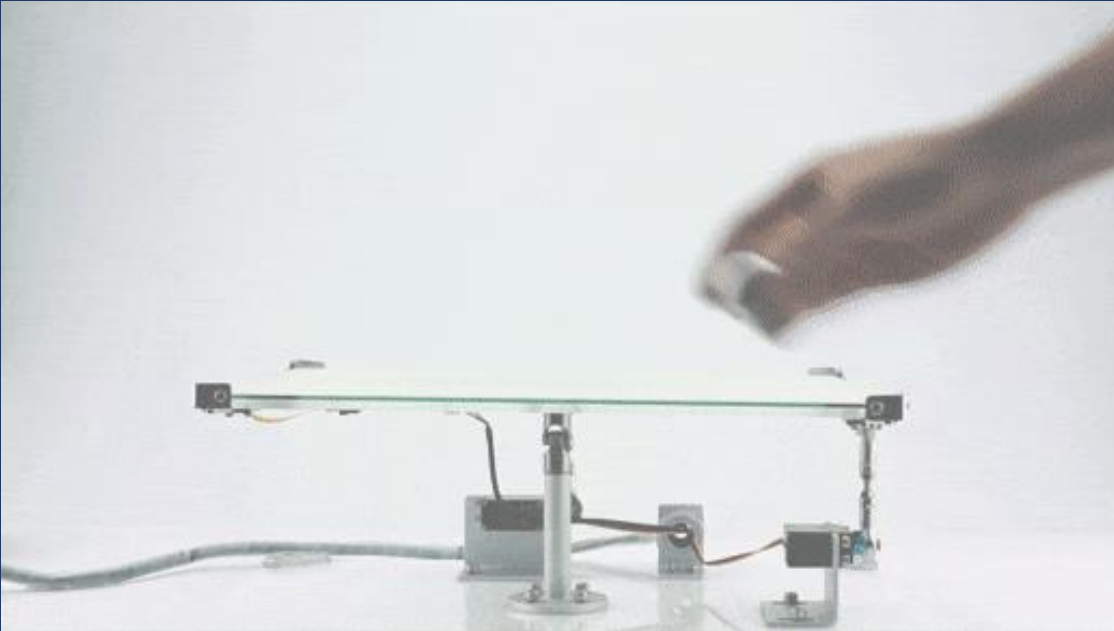
<https://instrumentationtools.com/what-is-pid-controller>

Keep the drone flying at a consistent altitude



https://ece4760.github.io/Projects/Fall2023/gb438_dy297_zr86/ece5730_quadcopter.html

More examples of applications utilizing PID Controllers



<https://acrome.net/post/understanding-pid-control-using-2-dof-ball-balancer-experiments>



<https://medium.com/xodlang/xod-and-pid-powered-self-balancing-mbot-6c420600271e>

Other examples of Self-Balancing Applications



Concept Bike from Yamaha at the Osaka World Expo (2025 Apr)



Self-Balancing Motorcycle Concept
Yamaha Motoroid 2.0

www.instagram.com/reel/DP3t1wkE4o5

Selecting the Appropriate PI, PD or PID Control Approach

PI Controllers

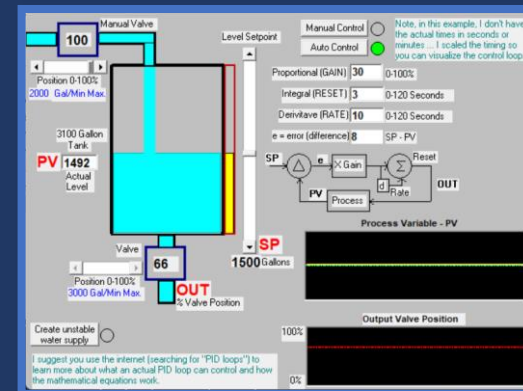
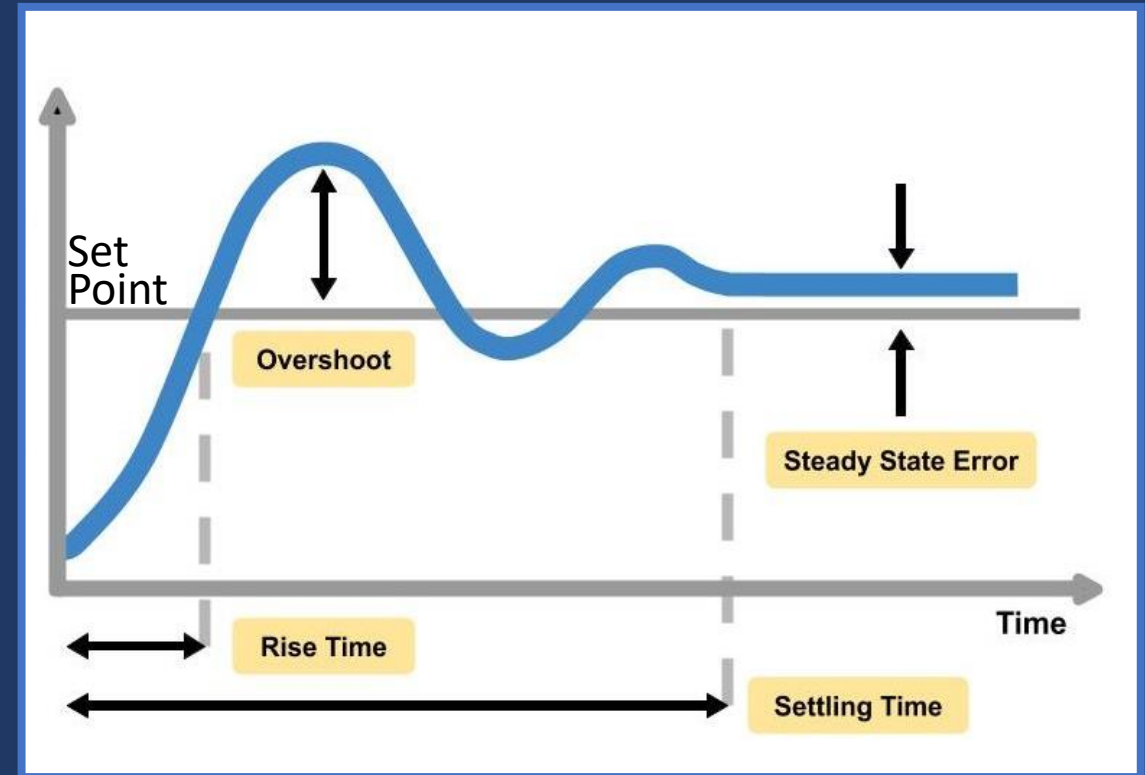
- serve to **eliminate steady-state error**
- use where **rise time is not a concern**
- well-suited to flow and level applications

PD controllers

- enhances both response speed and stability
- excel at **minimizing overshoot** while managing rapid transient responses

PID controllers

- deliver superior speed and accuracy
- their **tuning complexity** and noise sensitivity demand greater care.



Considerations in designing the device

- Minimize the use of equipment to the essential components only.
- Select a Brushless DC Motor that:
 - delivers the necessary torque.
 - allows precise control over rotation speed and direction.
- Design the body to be as compact as possible.
- Ensure compatibility with rechargeable AAA batteries.
- Route electrical wiring neatly through designated channels.
- The project should be feasible for most students to complete.

Electronic components of the Device



Battery Box
with Switch



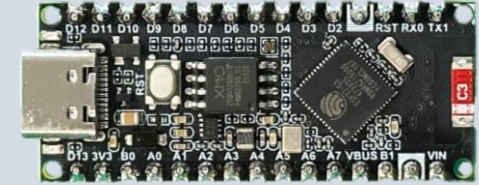
10440 3.7V
Re-chargeable
battery x 3



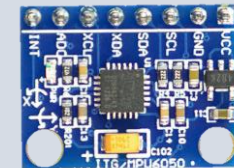
DC Voltage Step-
Down Converter



Nidec-24H
Brushless DC Motor



ESP32-S3-Nano
Development
Board



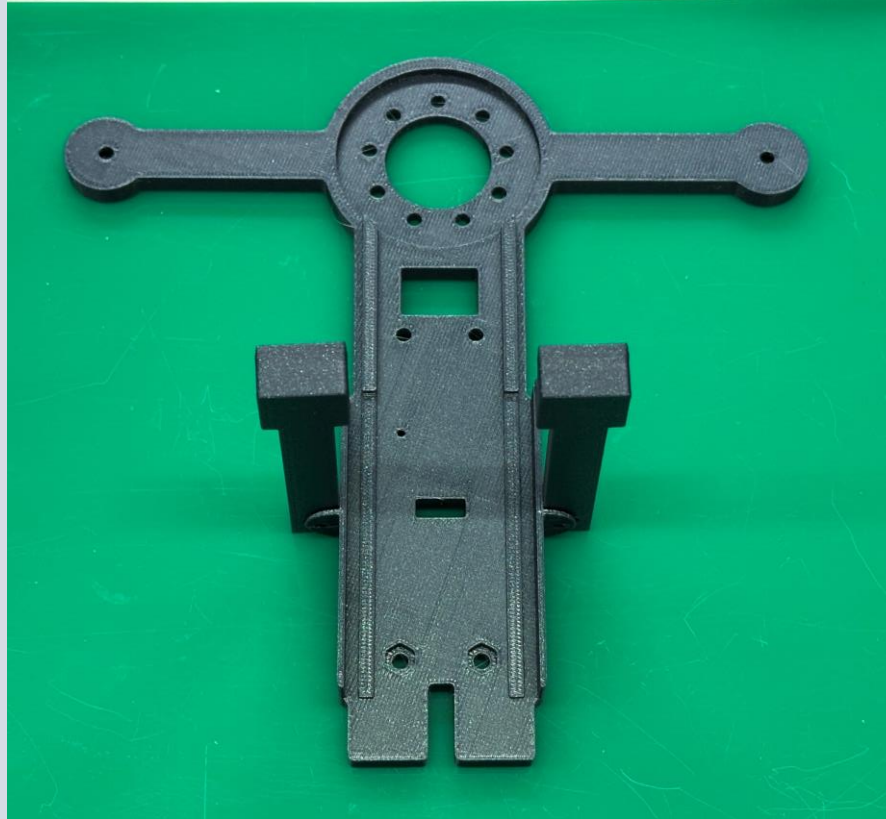
Accelerometer &
Gyroscope Module
(MPU6050)

3D-printed components of the Device



Reaction wheel

(Using metal screws to adjust its
Moment of Inertia 轉動慣量)



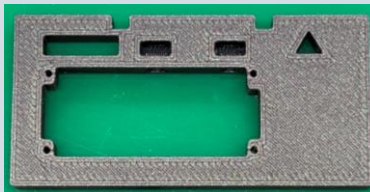
Device main body



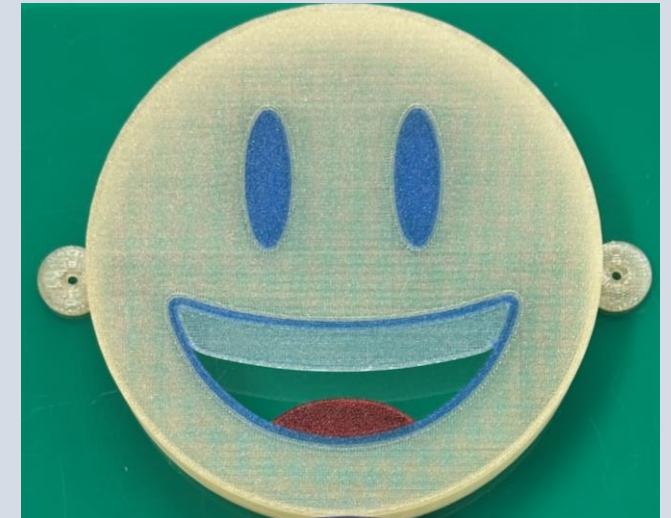
Back protective cover



Device pivot support

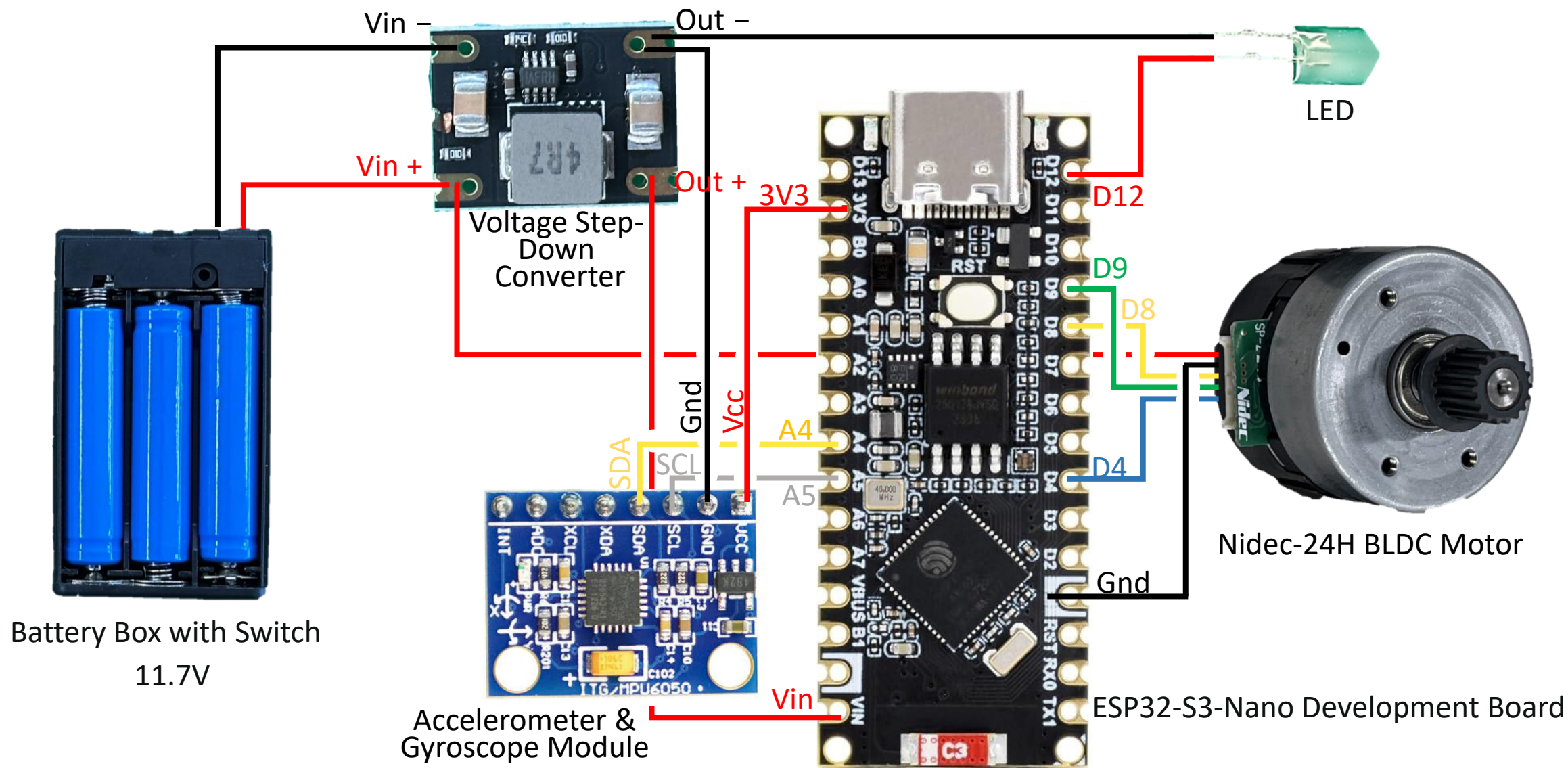


MCU bracket



Front protective cover

Simple Wiring Diagram

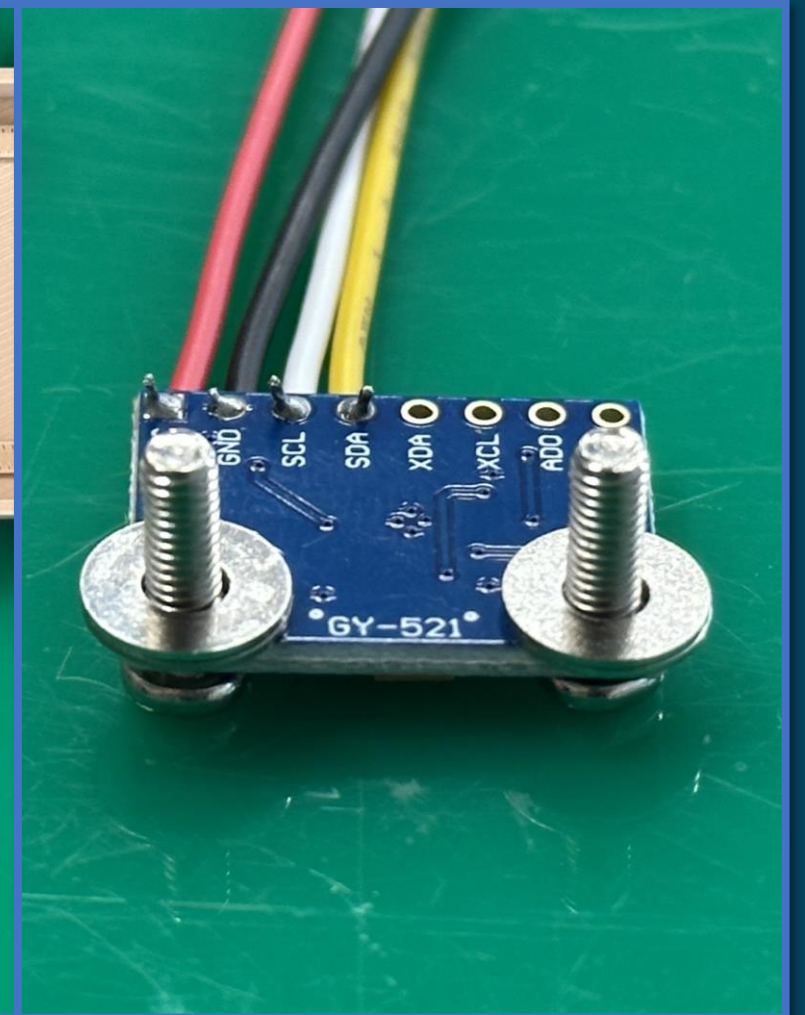
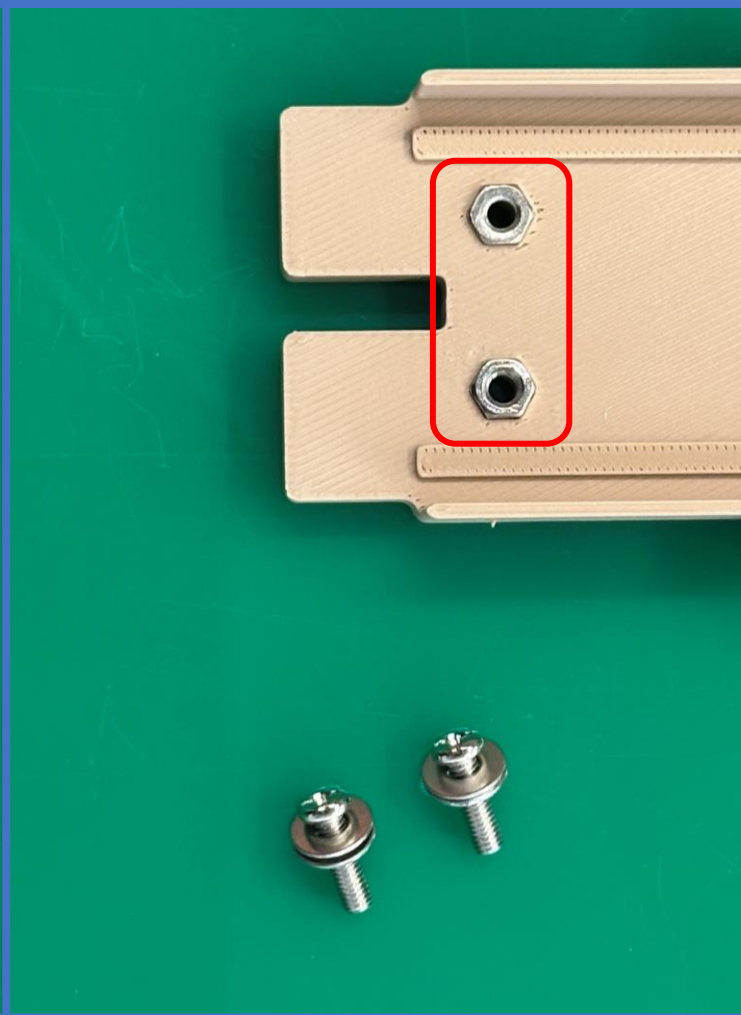
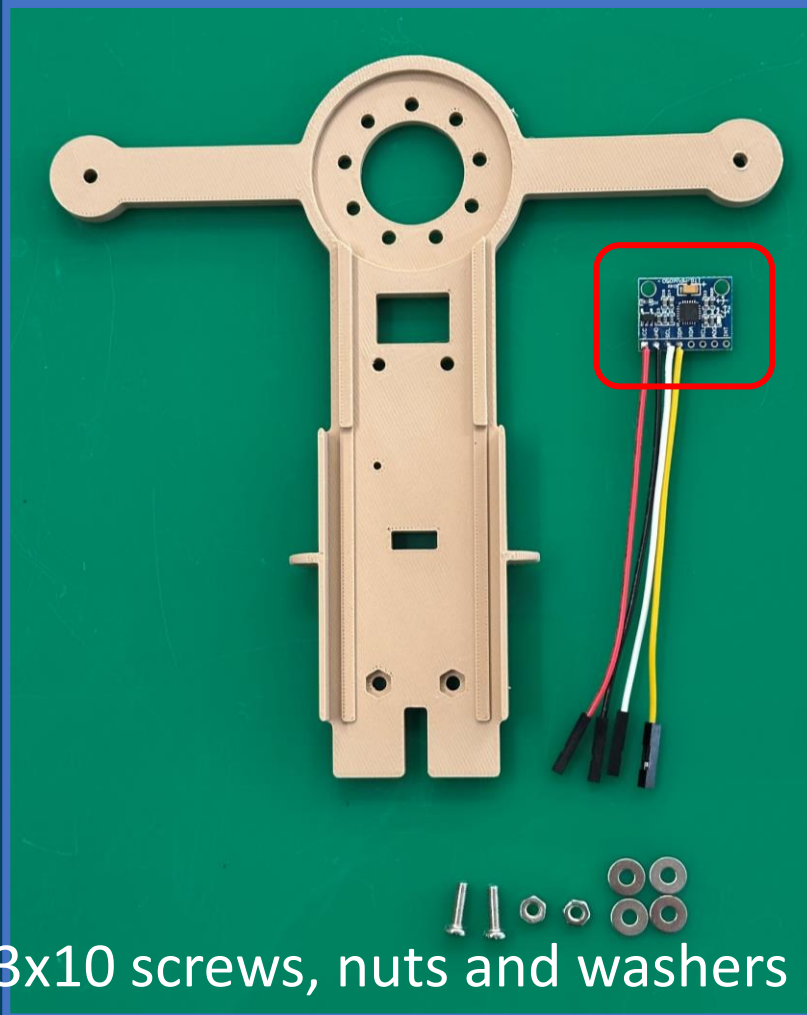


Screws and nuts used in securing different components of the device

Step	Item	Type	Length	Number	
1	M3 screw	Pan Head	10	2	
	M3 washer			4	
	M3 nut	Hex		2	
2	M2 screw	Flat head	8	1	
3	M1.4	Pan Head	6	4	
5	M3 screw	Pan Head	15	2	
	M3 nut	Hex		2	
7	M3 screw	Pan Head	6	3	
9	M4 screw	Pan Head	10	14	
	M4 nut	Hex		14	
	M4 nut	Cap		14	
10	M3 screw	Flat head	14	2	
	M3 nut	Hex		2	
11	M3 screw	Pan Head	15	2	
	M3 nut	Hex		2	
--	M2 screw	Flat head	10	2	Already fixed on device body
--	M2 nut	Hex		2	

Assembly of the device:

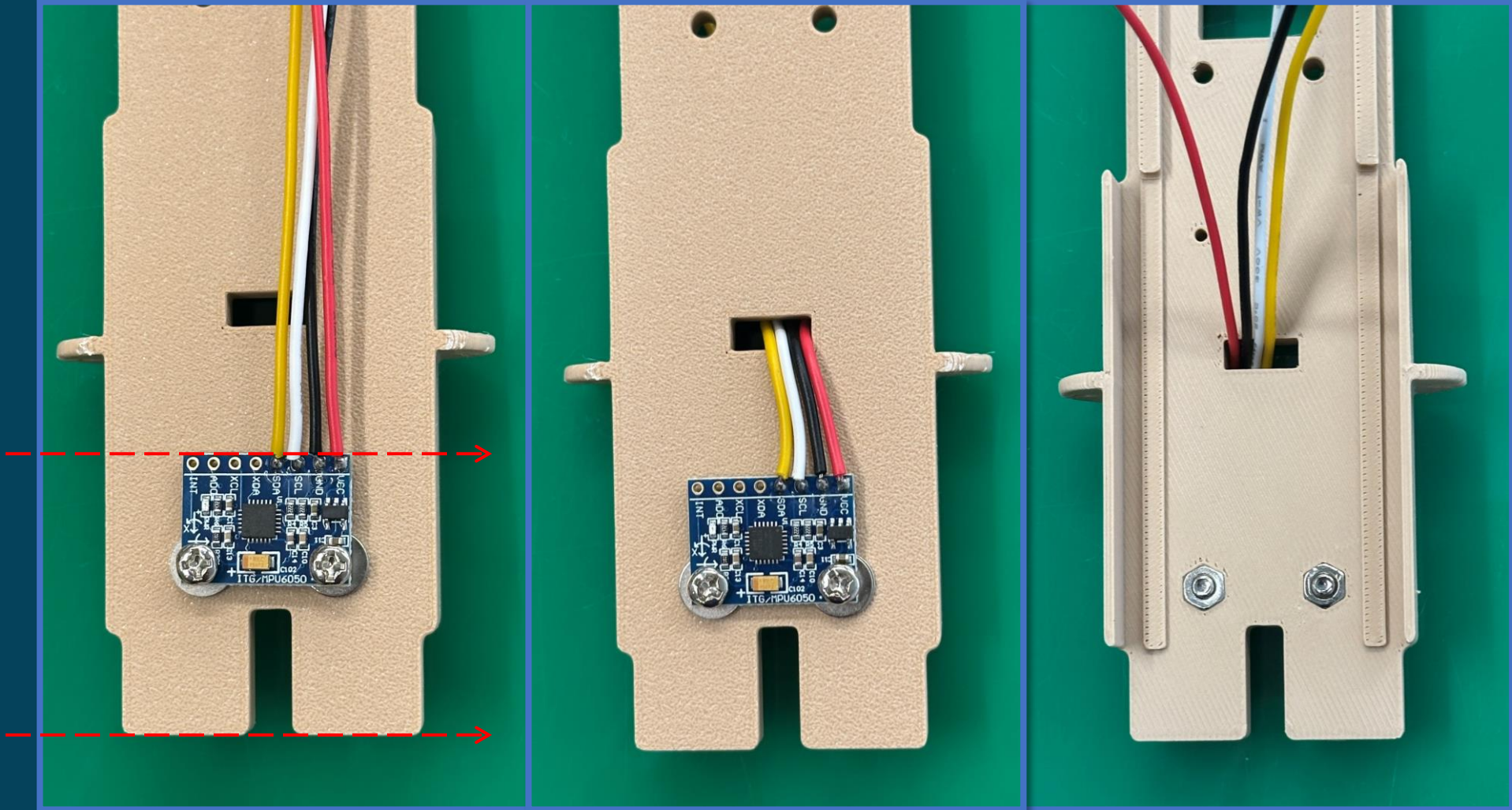
- 1 Secure the **Accelerometer & Gyroscope Module**



M3x10 screws, nuts and washers

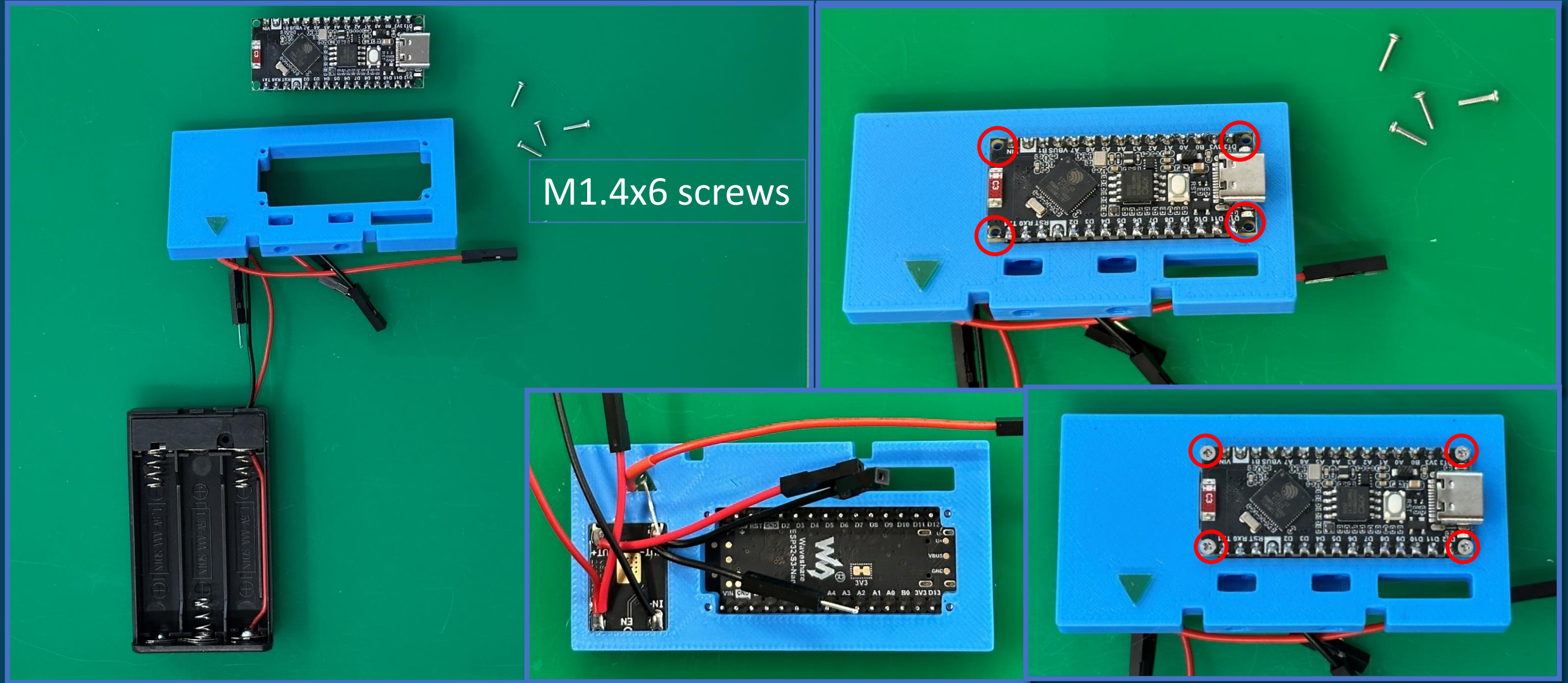
Assembly of the device

1 Secure the Accelerometer & Gyroscope Module

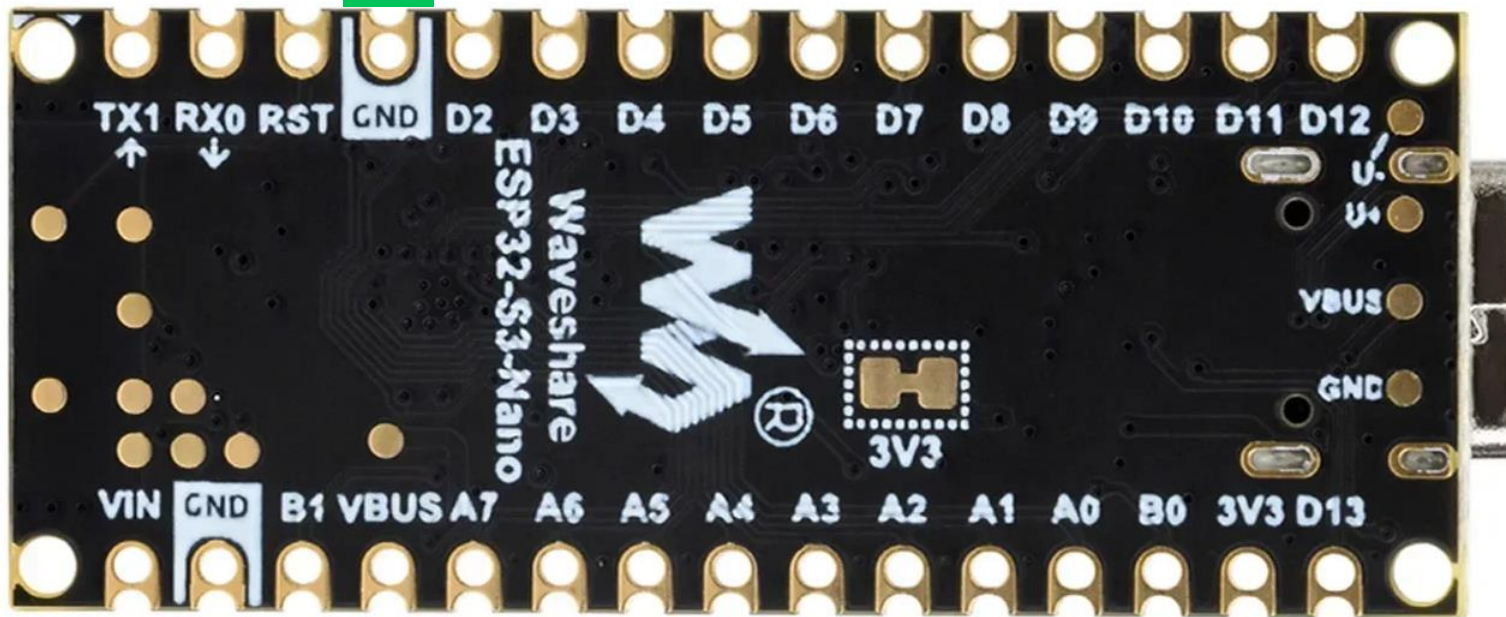


Assembly of the device

2 Place the ESP32-S3-Nano Development Board

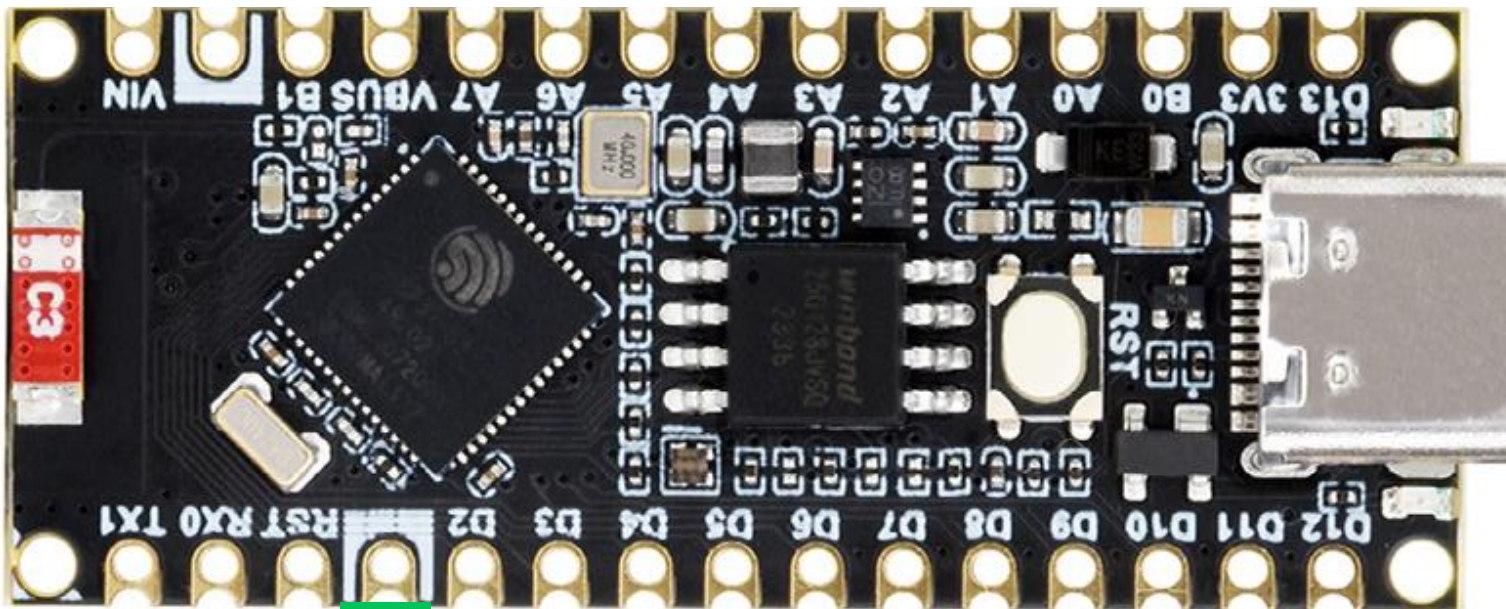


TX1 RX0 RST GND D2 D3 D4 D5 D6 D7 D8 D9 D10 D11 D12



Back

VIN GND B1 VBUS A7 A6 A5 A4 A3 A2 A1 A0 B0 3V3 D13

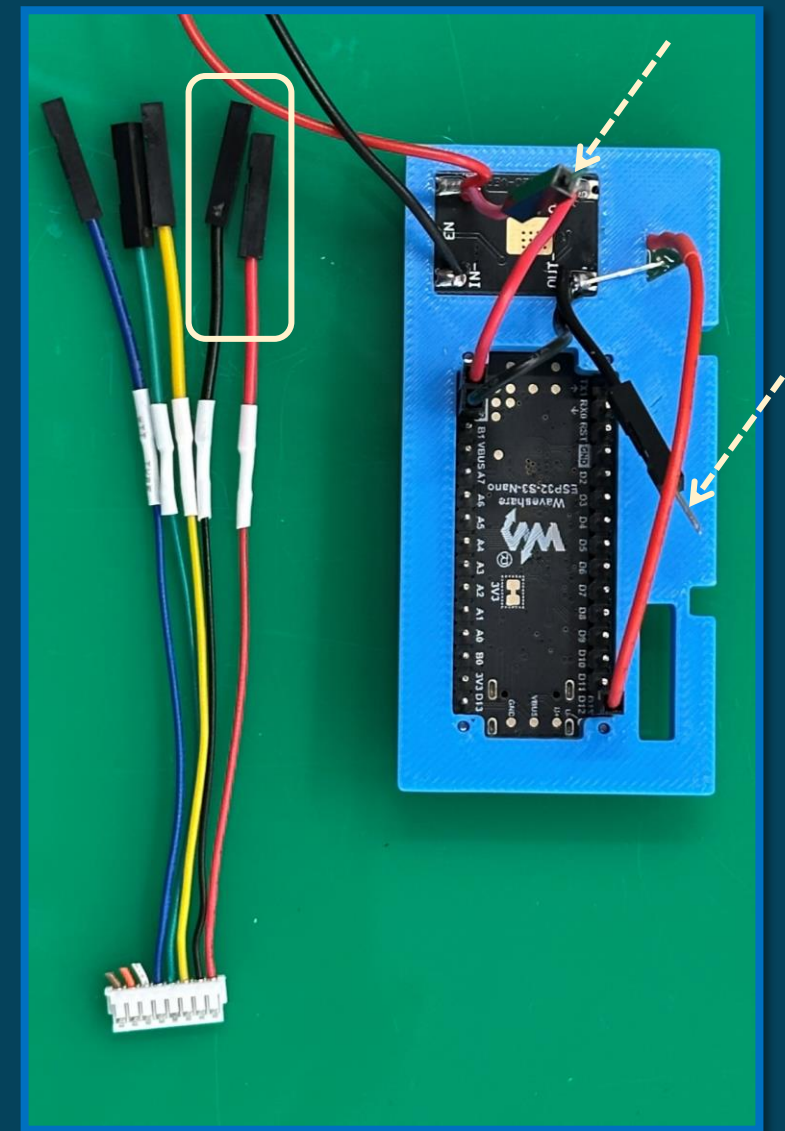
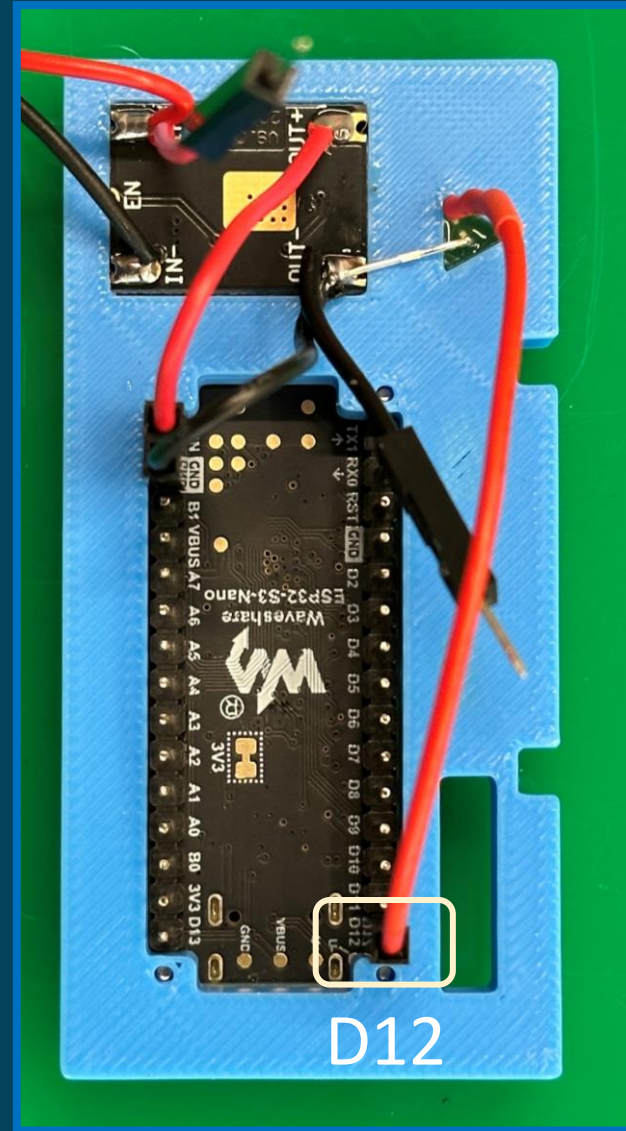
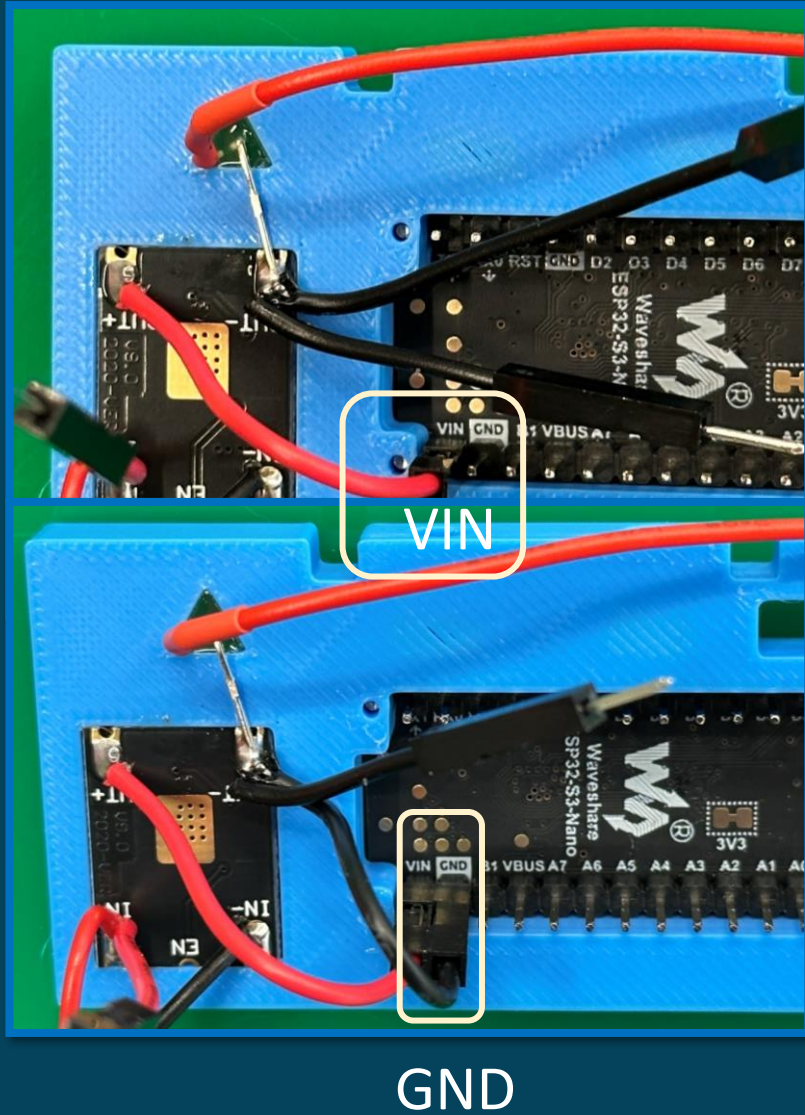


Front

TX1 RX0 RST GND D2 D3 D4 D5 D6 D7 D8 D9 D10 D11 D12

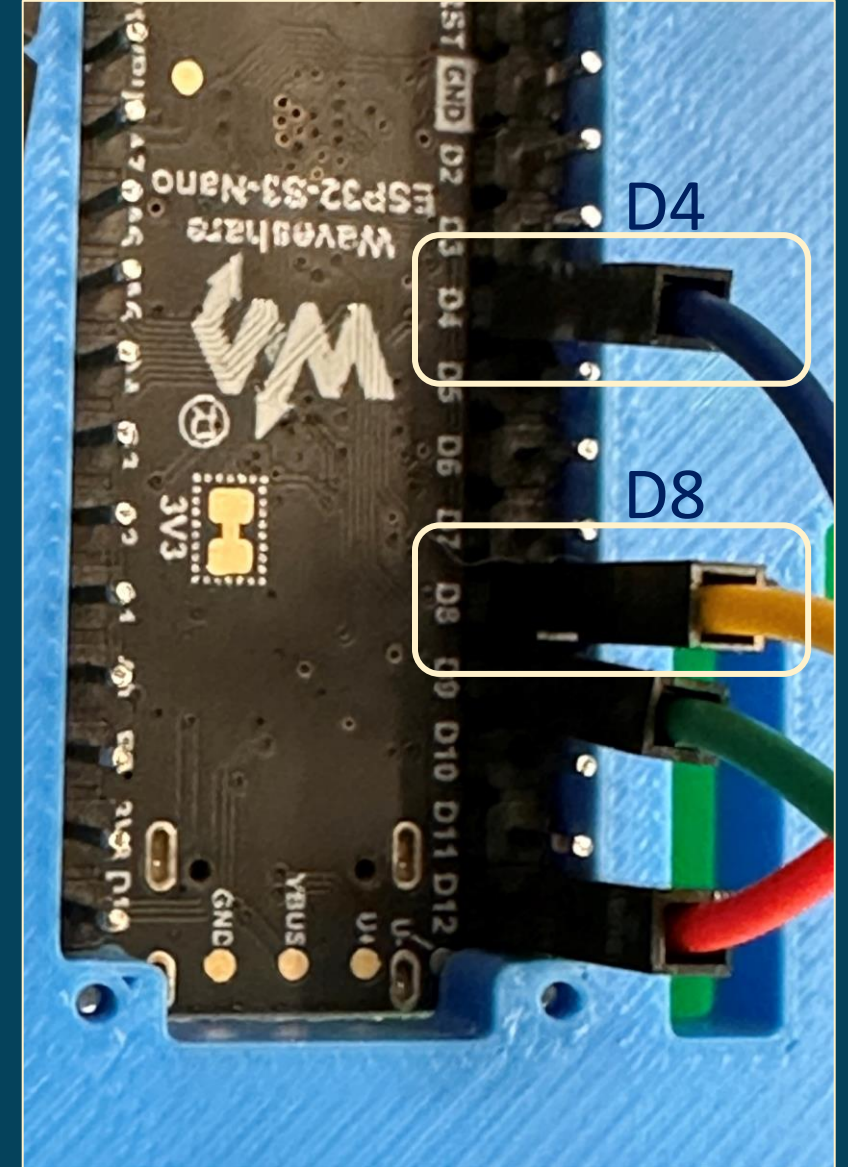
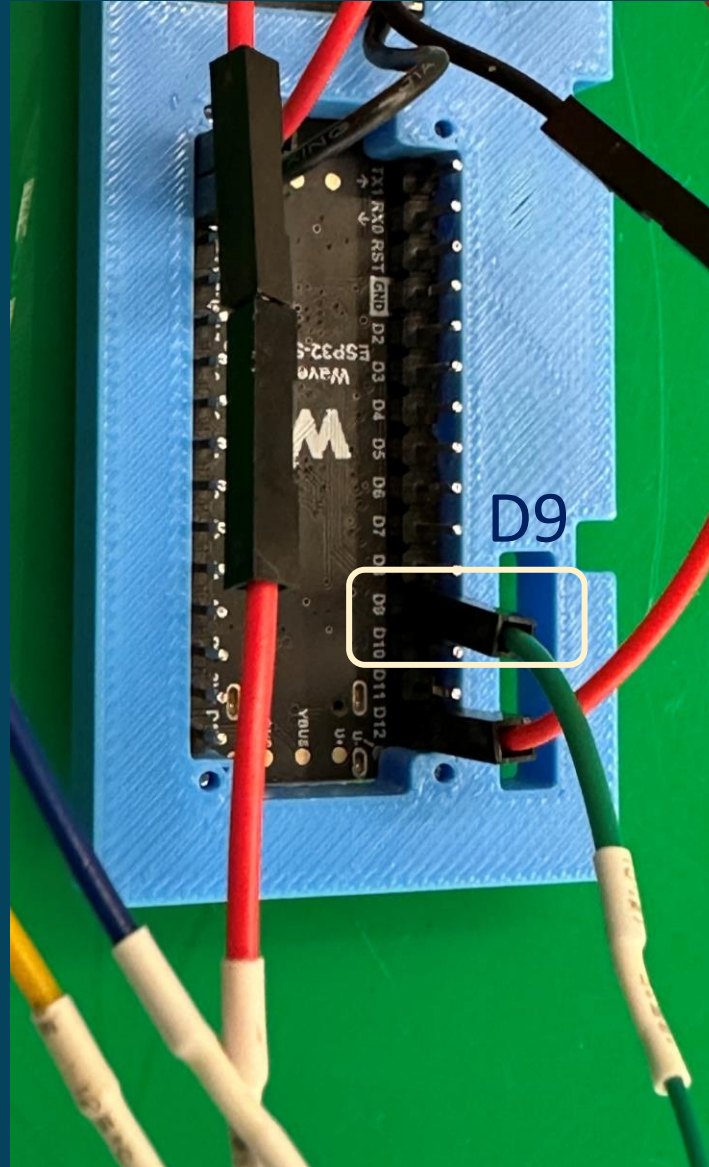
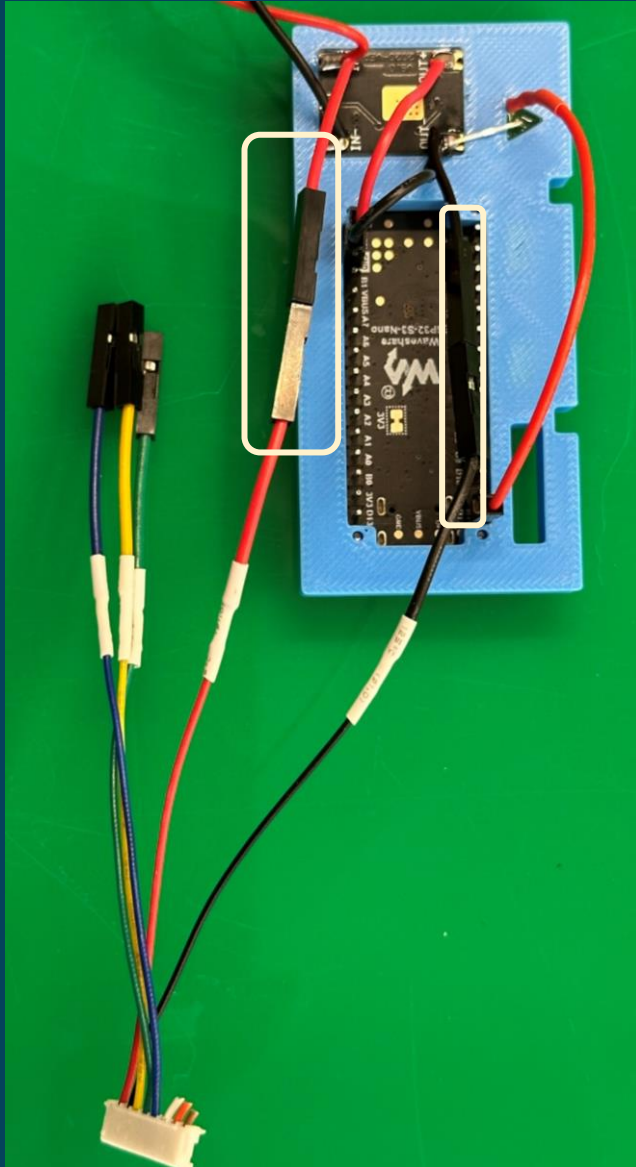
Assembly of the device

3 Attach the Dupont Cables (1)



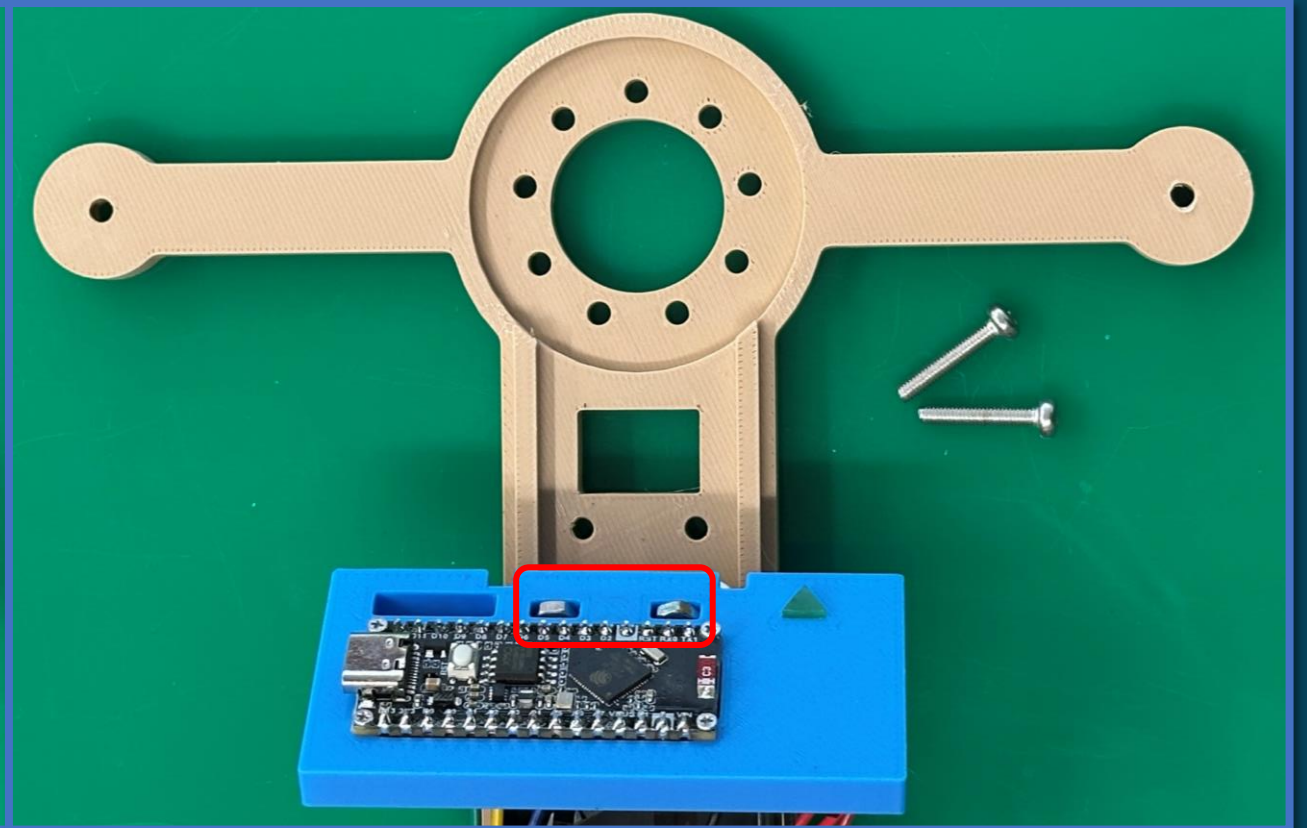
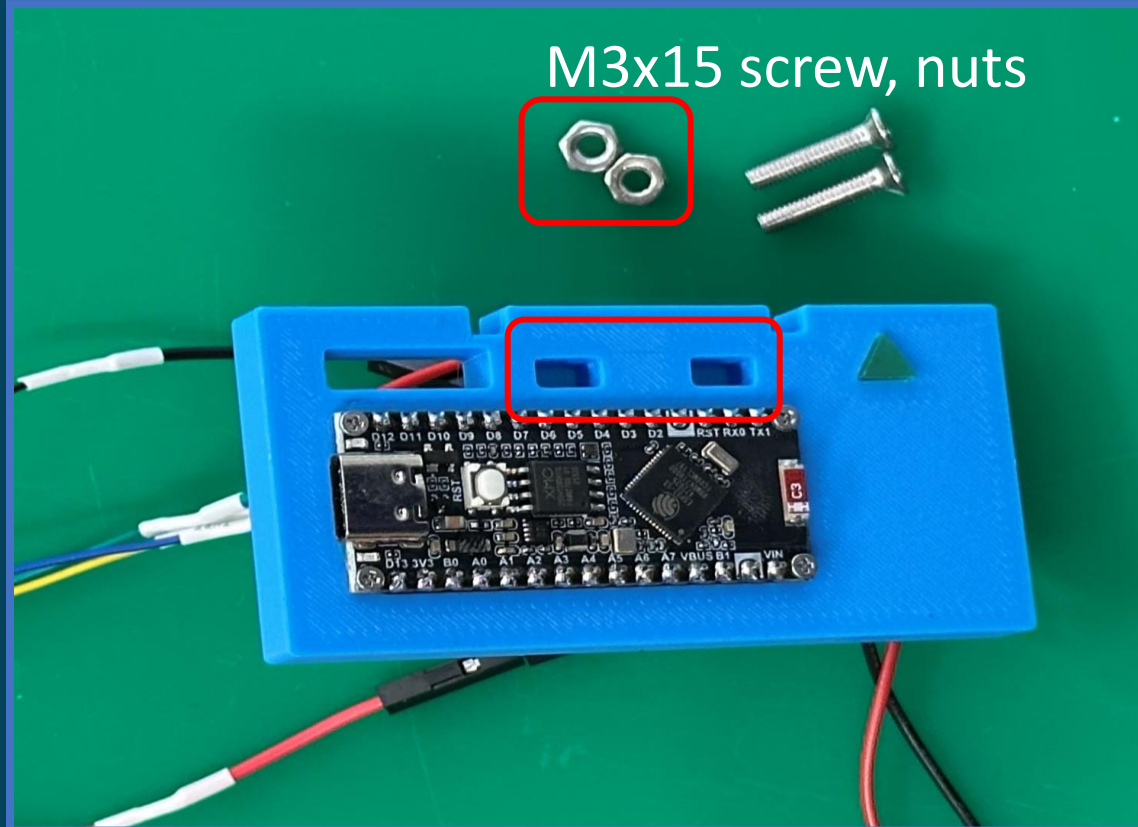
Assembly of the device

3 Attach the Dupont Cables (1)



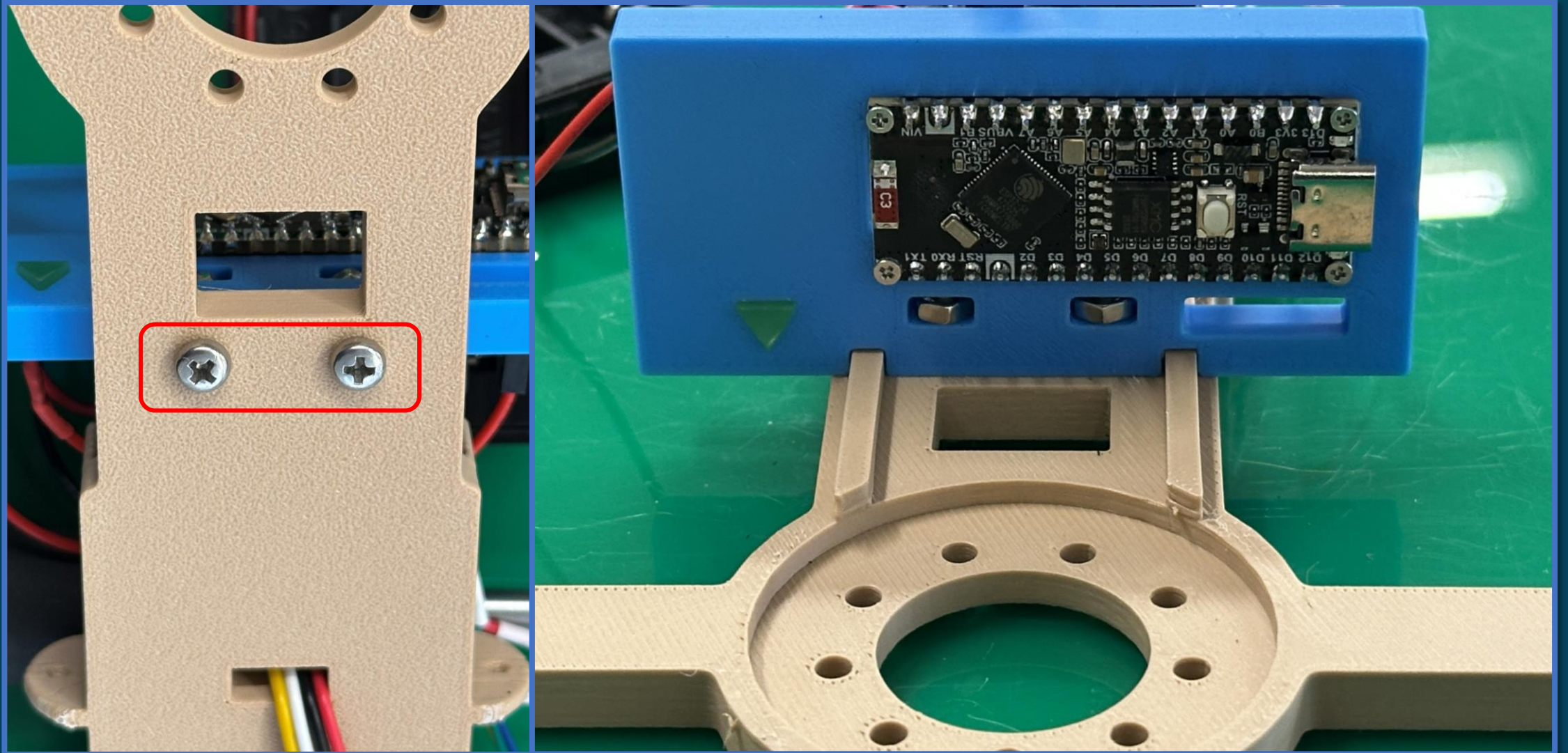
Assembly of the device

- Secure the **MCU bracket** to the main body of the device



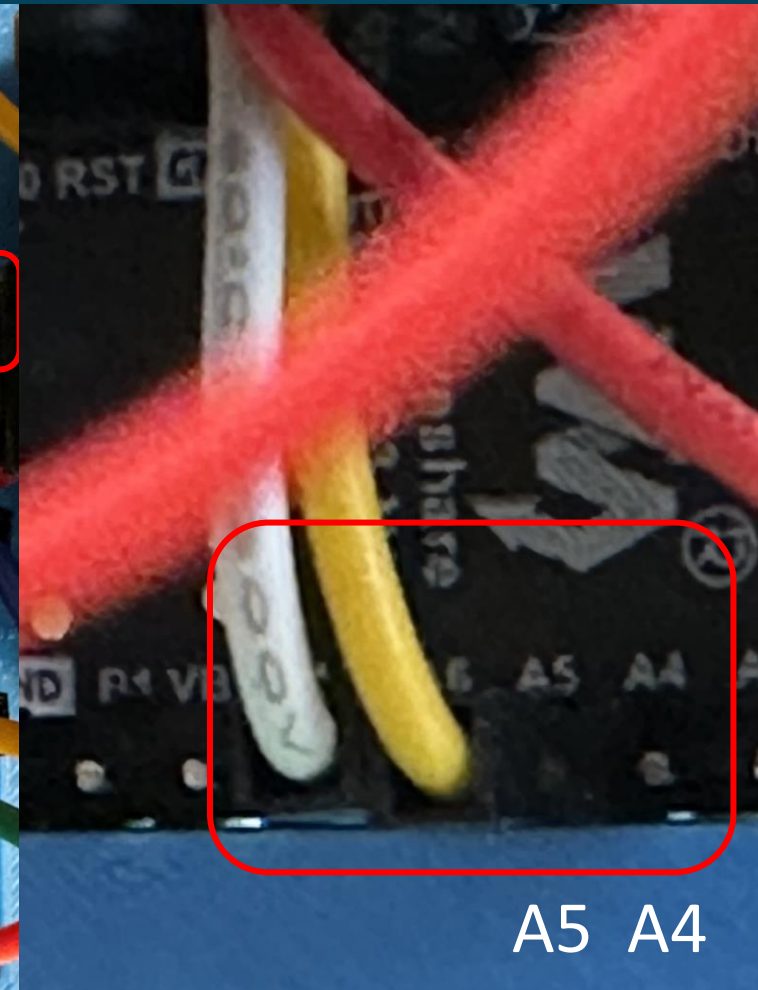
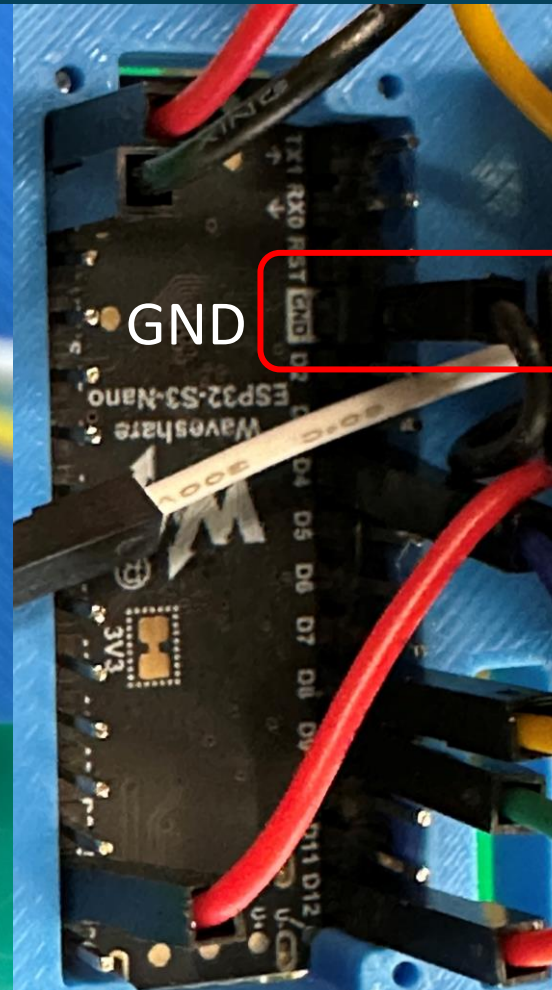
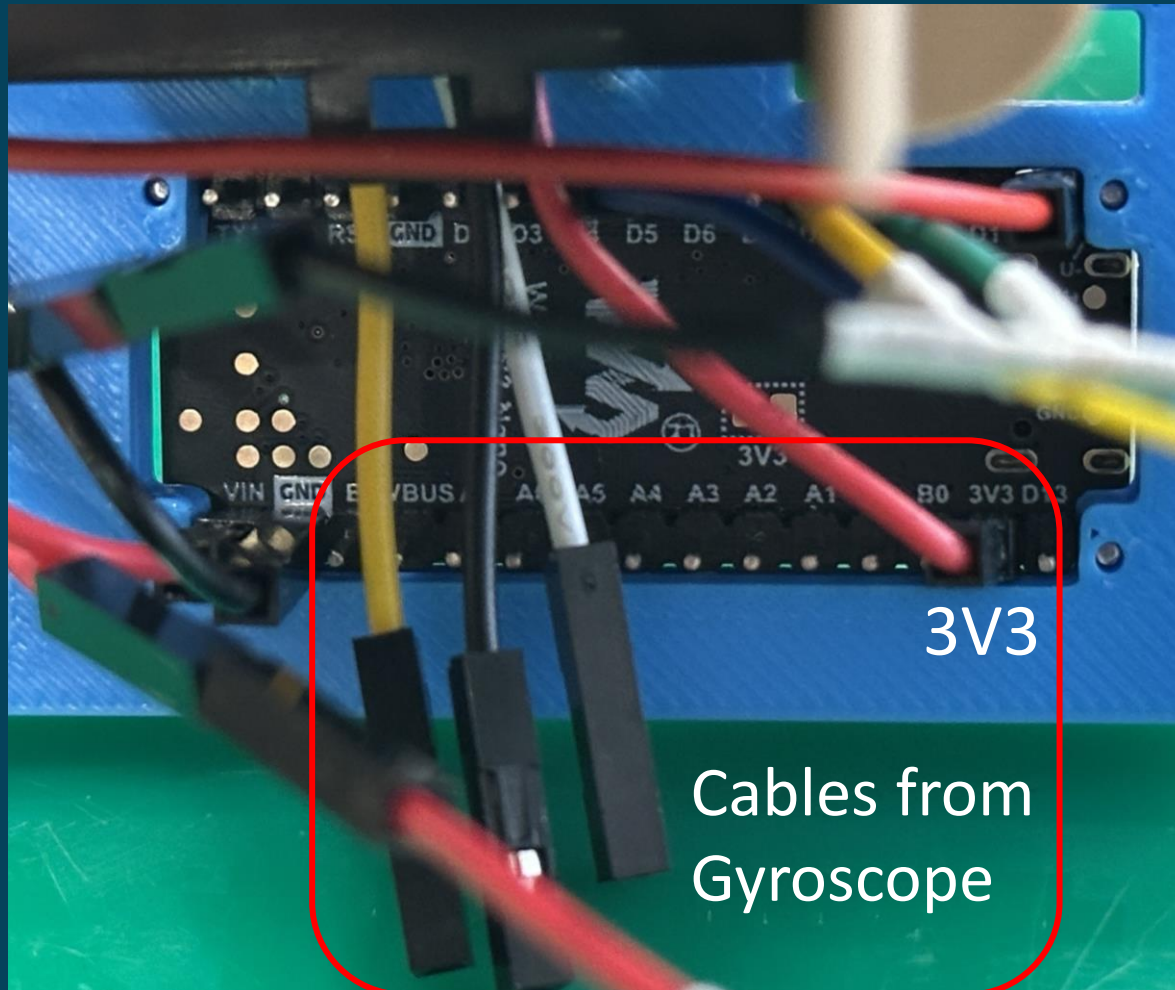
Assembly of the device

4 Secure the **MCU bracket** to the main body of the device



Assembly of the device

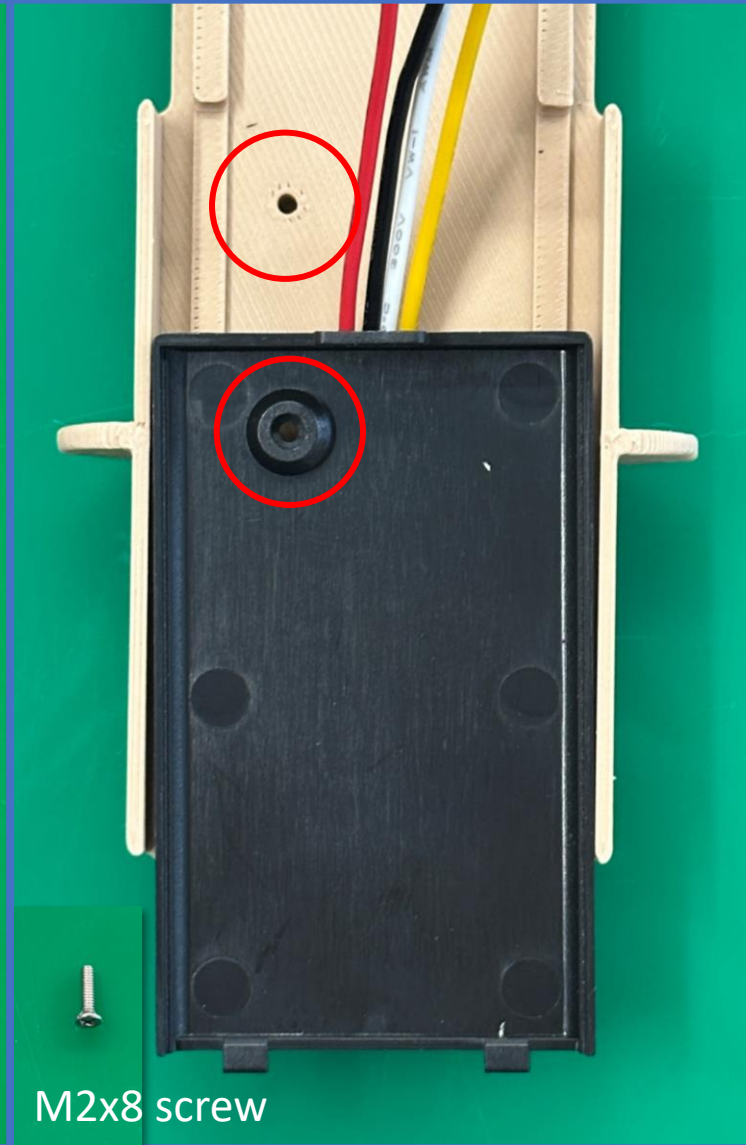
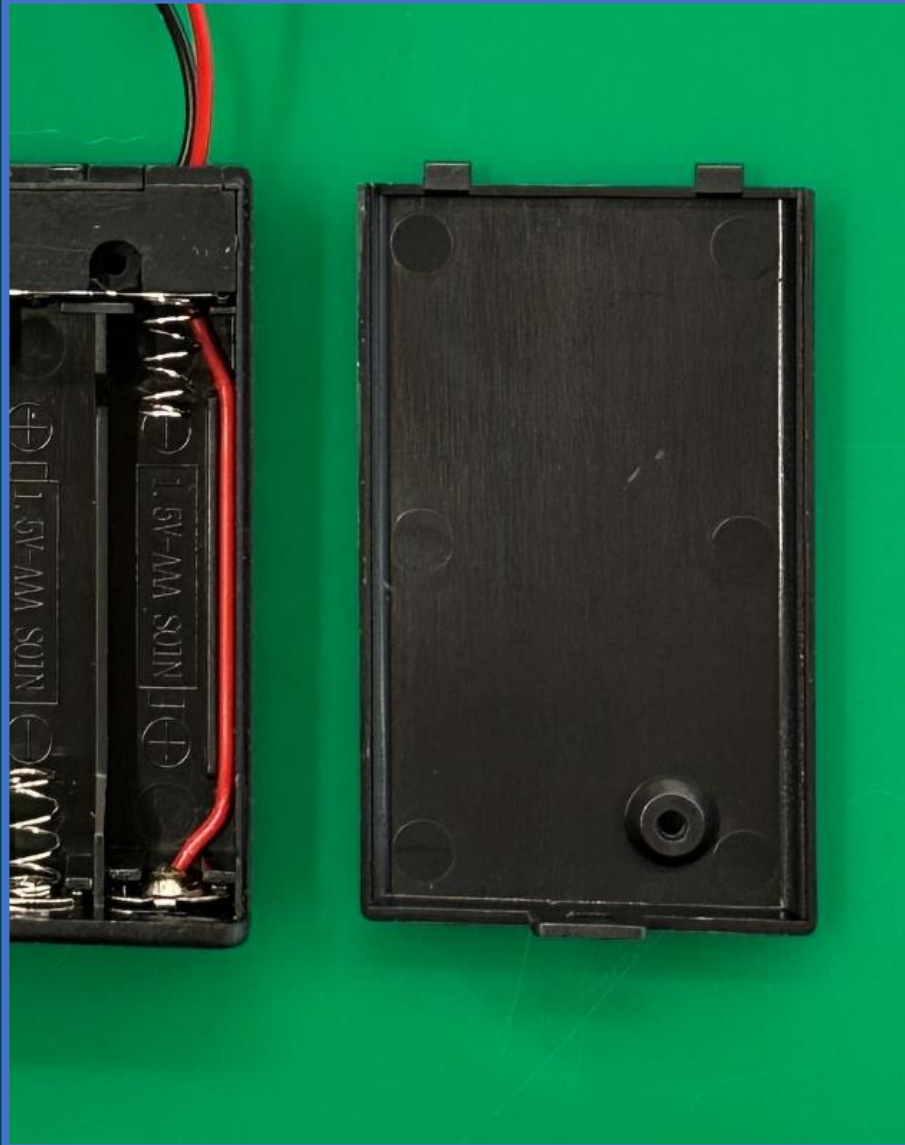
5 Attach the Dupont Cables (2)



Assembly of the device

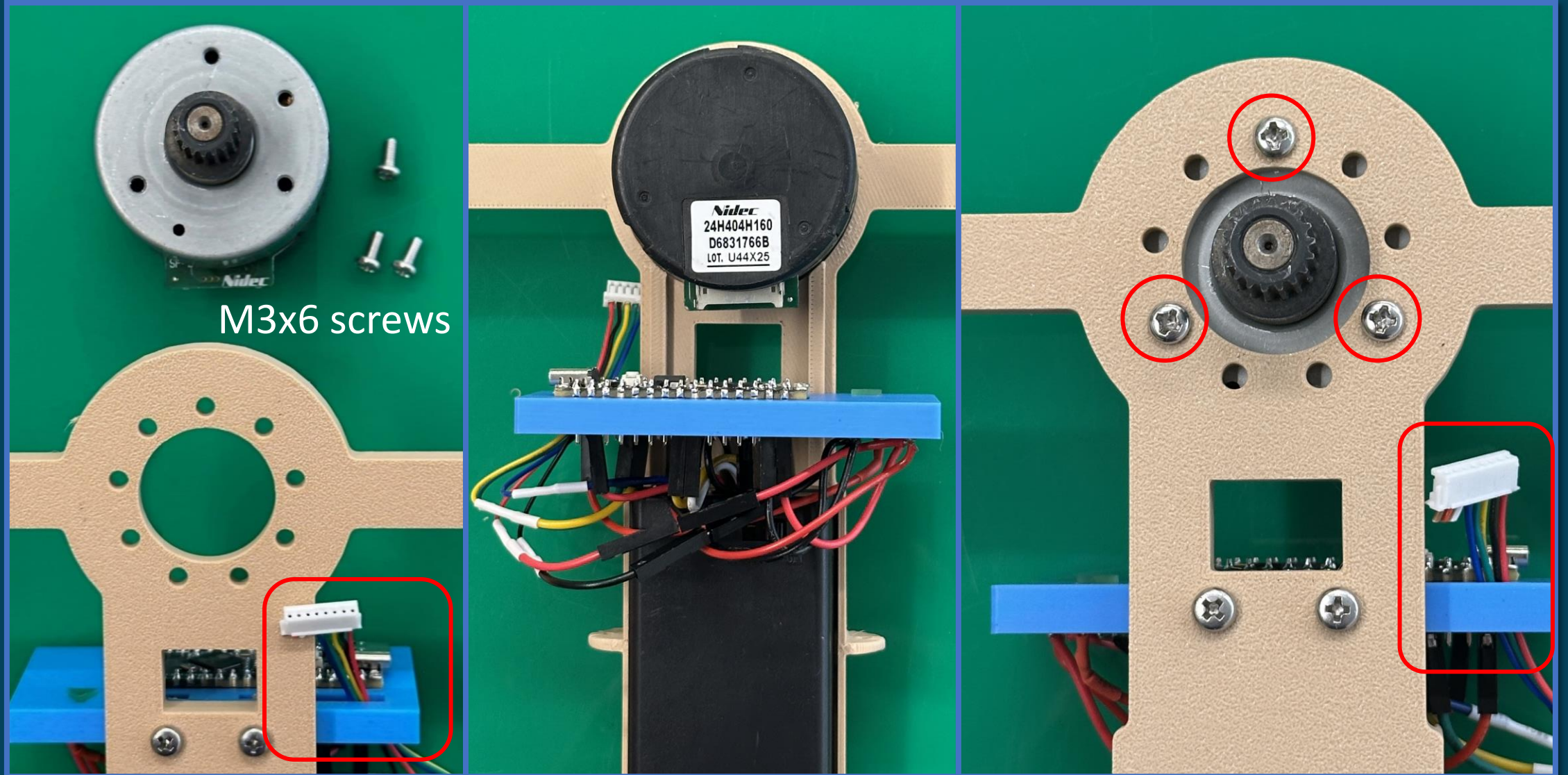
6

Secure the **battery box cover**



Assembly of the device

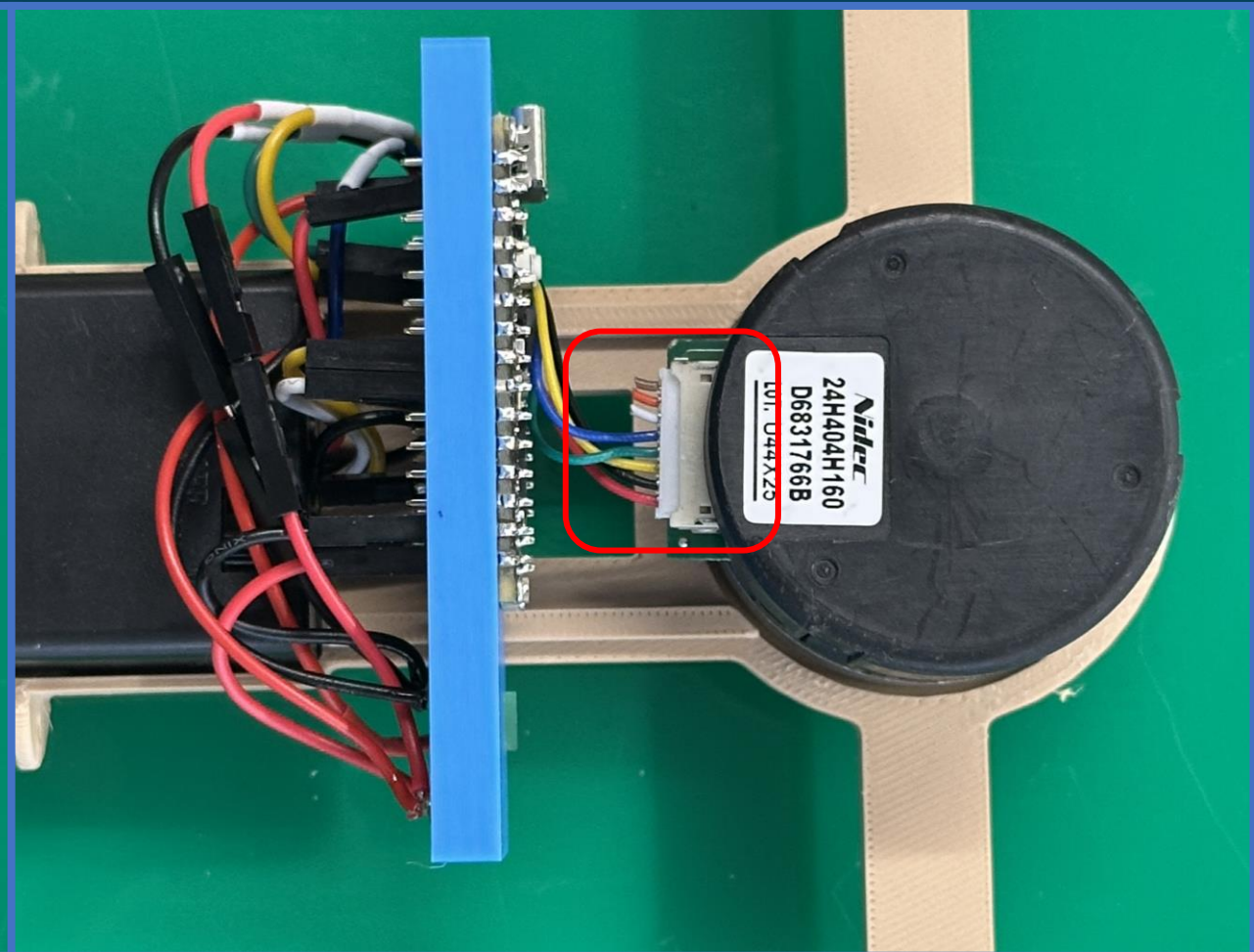
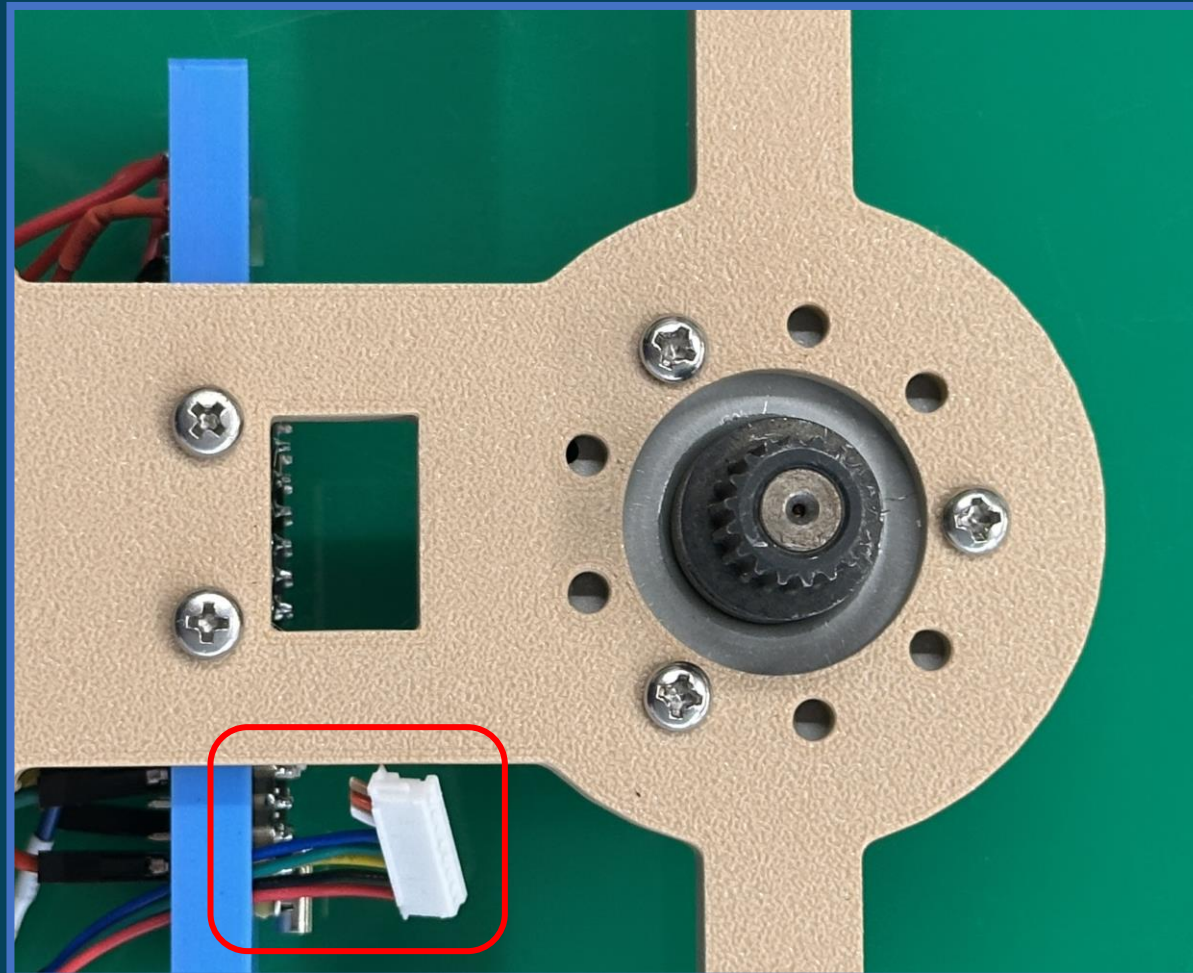
7 Secure the BLDC Motor



Assembly of the device

8

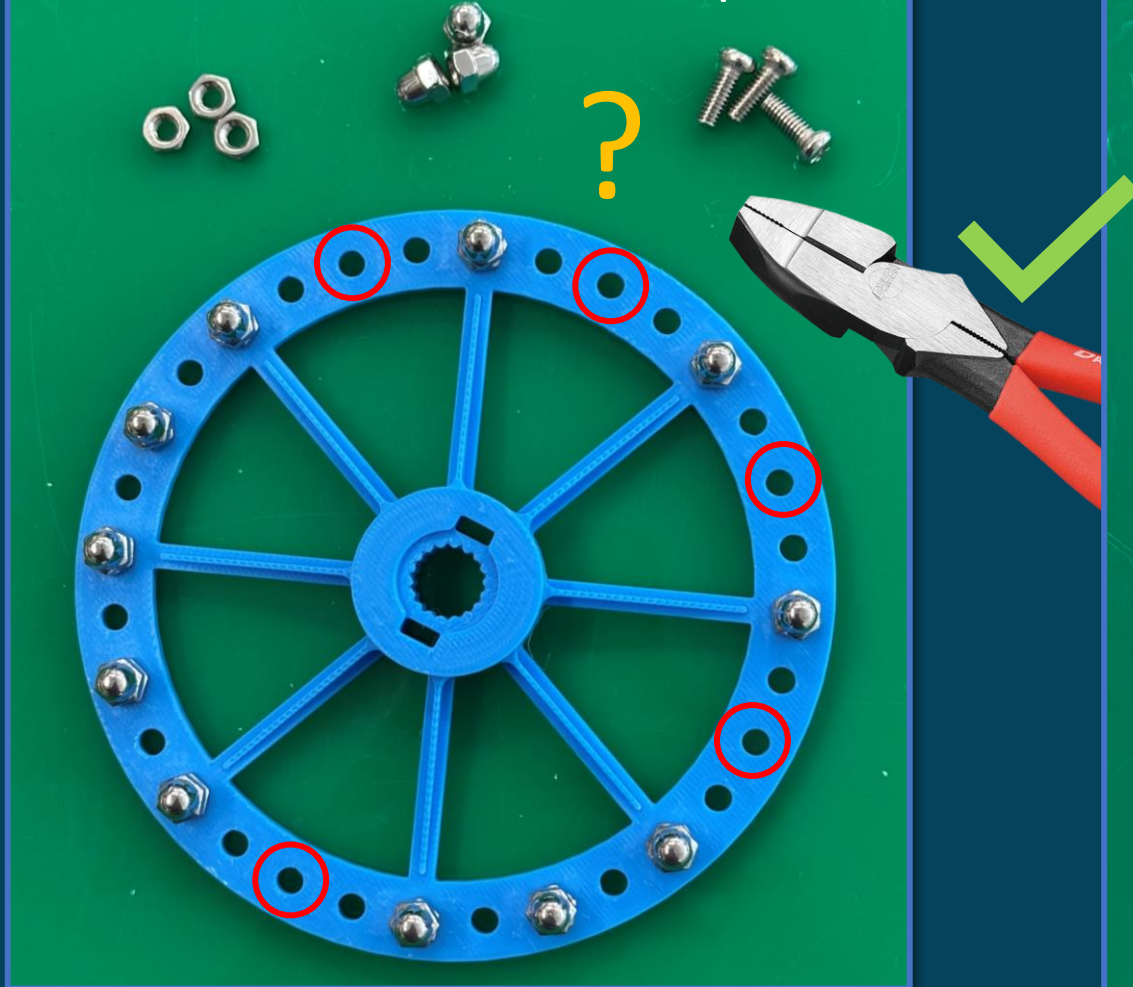
Attach the PH2.0-8P connector to the BLDC motor



Assembly of the device

9 Secure the screws around the edge of the **reaction wheel** (for what purpose?)

M4x10 screws, nuts and caps

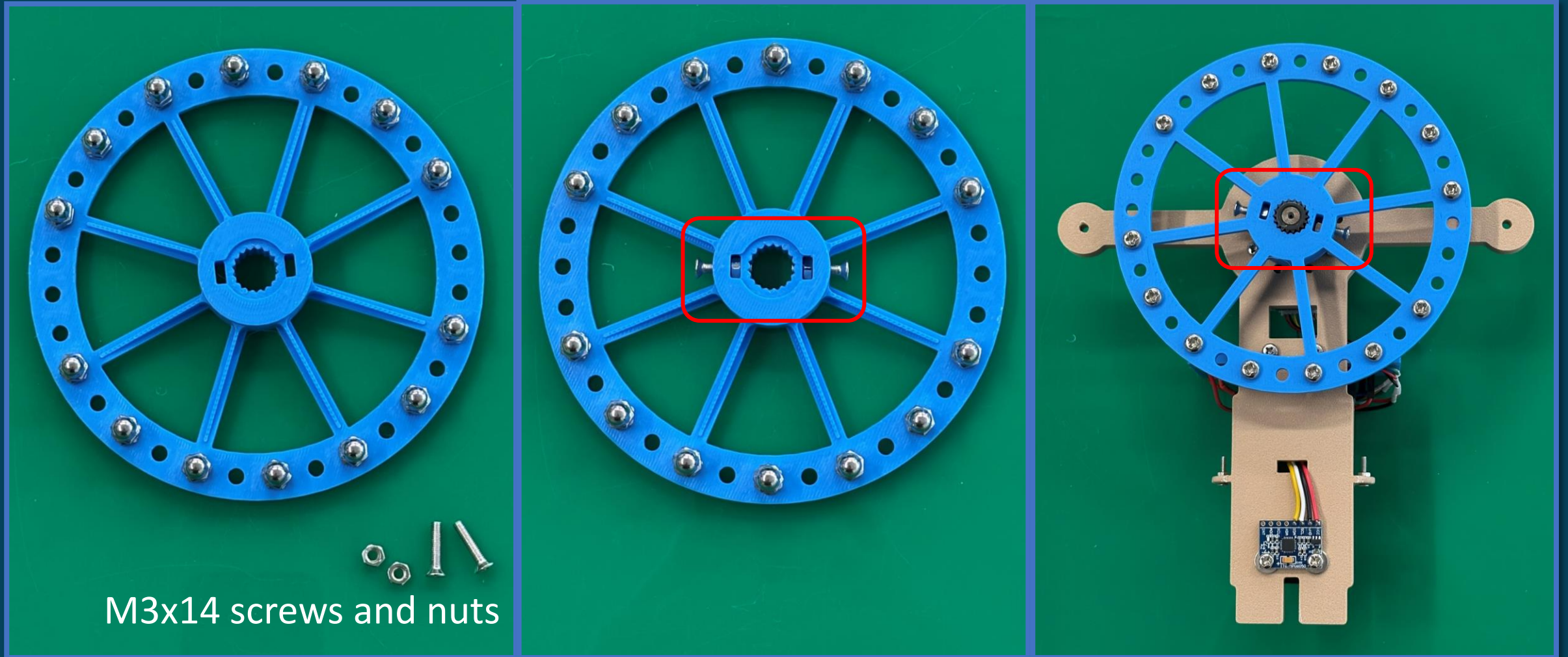


Why were these locations excluded?



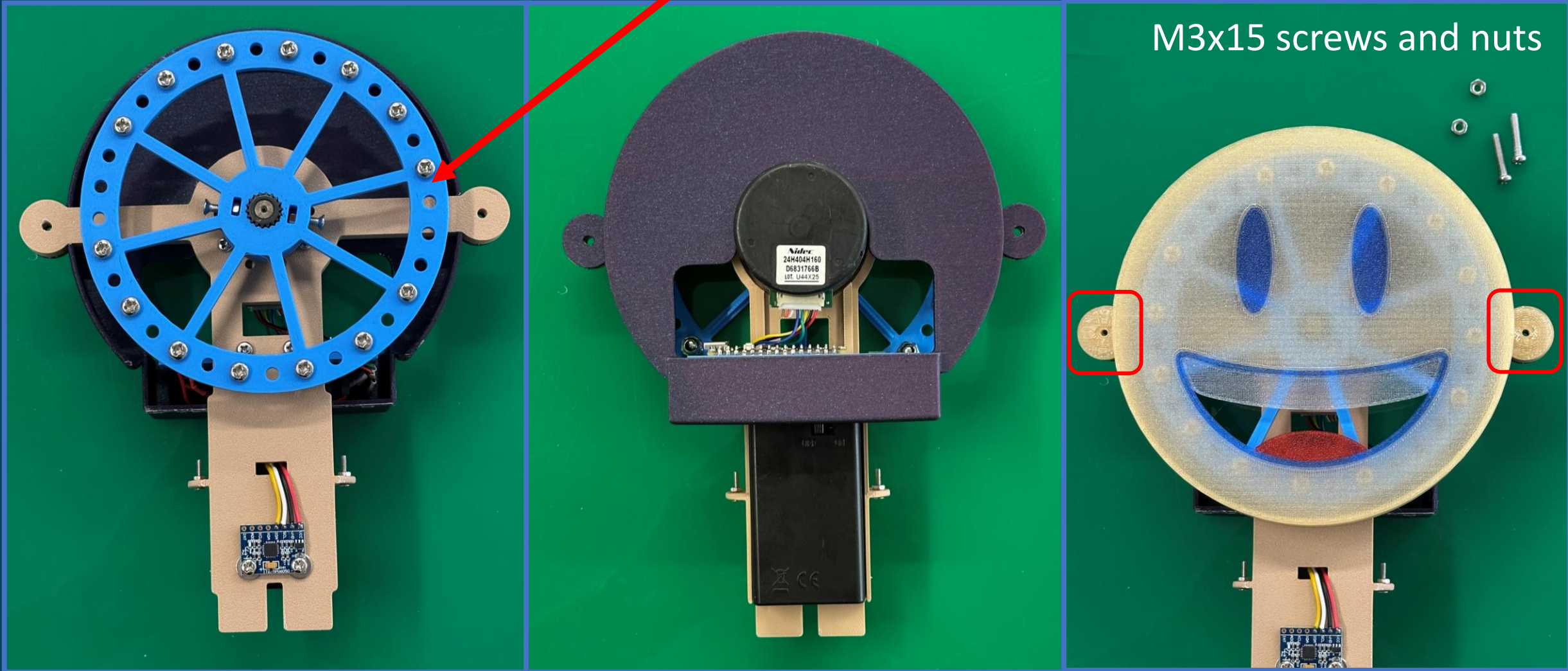
Assembly of the device

10 Attach the **reaction wheel** firmly to the **BLDC motor**



Assembly of the device

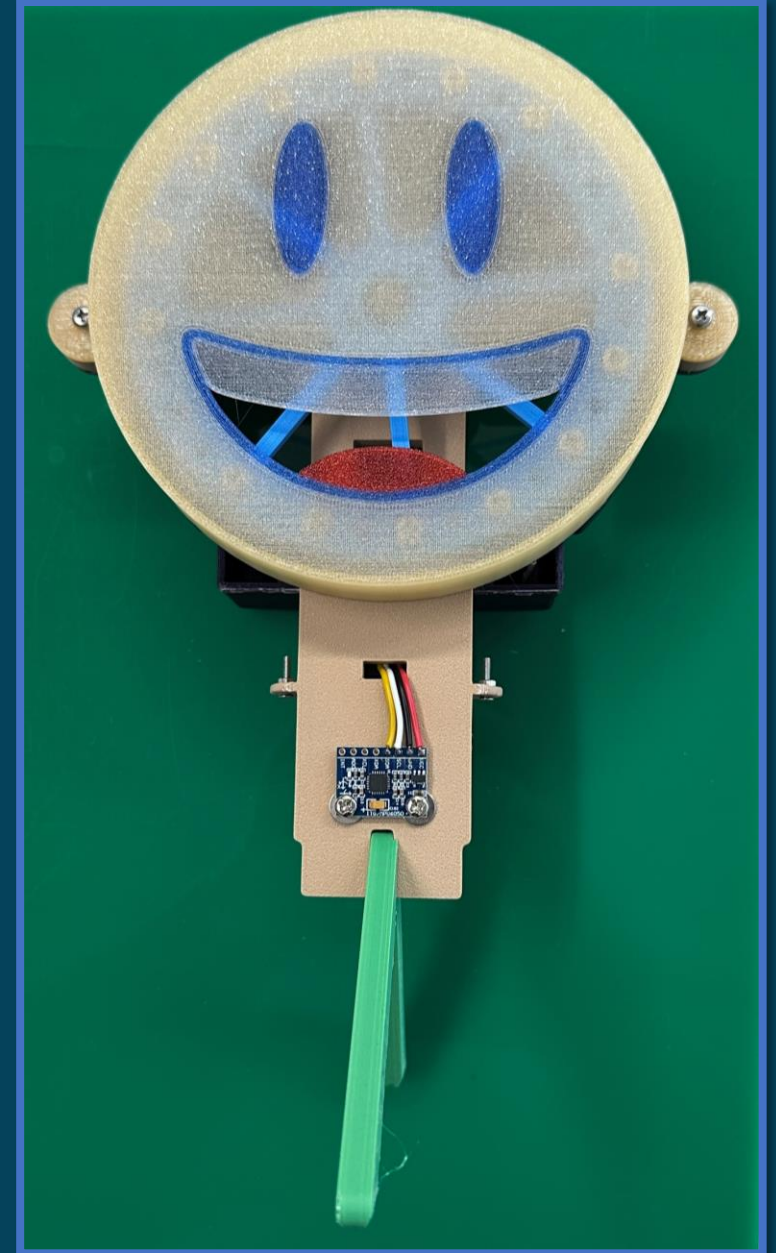
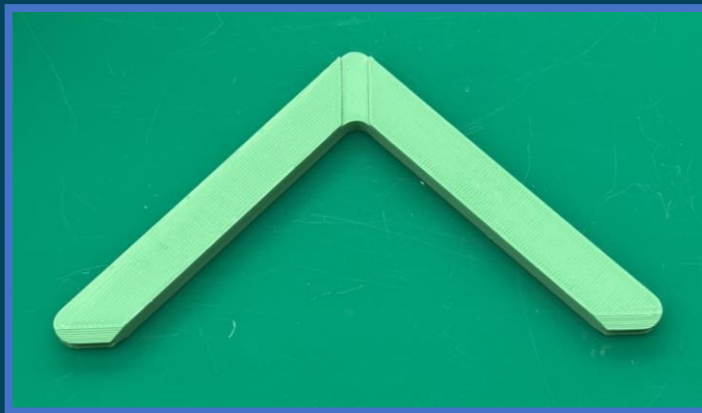
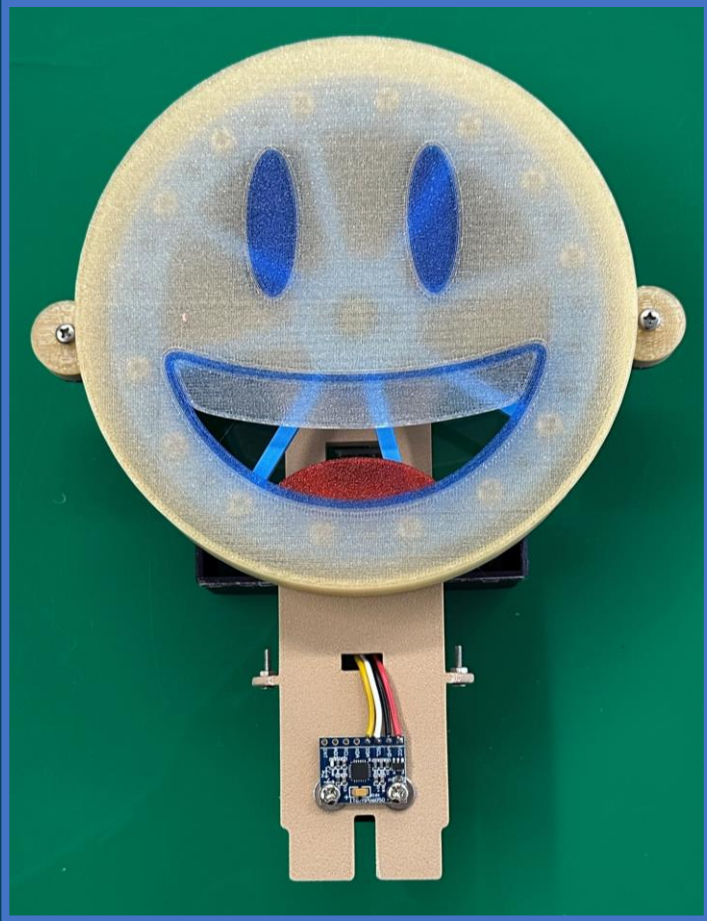
11 Install the protective covers (Prevent the wheel from getting contact with the covers)



Assembly of the device

12

Place the battery to the battery case and attach the leg to the main body of the device



Coding to operate the Device

```
1 #include <Wire.h>
2 #include <MPU6050_tockn.h>
3
4 MPU6050 mpu6050(Wire);
5 #define BRAKE 8 // Stop motor pin
6 #define PWM 9 // speed control pin
7 #define DIRECTION 4 // rotation dir pin
8 #define LEDSIGNAL 12 // LED pin
9
10 const int freq = 20000; // 20 kHz
11 const int resolution = 8; // 8-bit (0-255)
12 float loop_time = 10;
13
14 float Kp = 76;
15 float Kd = 5.3;
16 float Kx = 0.05;
17
18 int pwm_s = 0;
19 int motor_speed;
20 int32_t timer;
21 long currentT, previousT;
22
23 //IMU offset values
24 int AcX_offset = -1300;
25 int AcY_offset = 300;
26 int GyZ_offset = 0;
```

```
71 if (pwm <= 0) {
72     digitalWrite(DIRECTION, LOW);
73     pwm = -pwm;
74 } else {
75     digitalWrite(DIRECTION, HIGH);
76 }
77 ledcWrite(0, map(pwm, 0, 255, 255, 0));
78 }
79
80 void loop() {
81     currentT = millis();
82     if (currentT - previousT >= loop_time) {
83         angle_calc();
84         if (vertical) {
85             digitalWrite(BRAKE, HIGH);
86             gyroZ = GyZ / 131.0; // Convert to deg/s
87
88             gyroZfilt = alpha * gyroZ + (1 - alpha) * gyroZfilt;
89             pwm_s = -constrain(Kp * device_angle + Kd * gyroZfilt + Kx * -motor_speed, -255, 255);
90             Motor_control(pwm_s);
91             motor_speed += pwm_s;
92             previousT = currentT;
93         } else {
94             Motor_control(0);
95             digitalWrite(BRAKE, LOW);
96             motor_speed = 0;
97 }
```

Self-BalanceWithReaction_wheel_ESP32_S3_Nano | Arduino IDE 2.3.5

Arduino IDE

File Edit Sketch Tools Help



Arduino Nano ESP32



Self-BalanceWithReaction_wheel_ESP32_S3_Nano.ino



```
1 #include <Wire.h>
2 #include <MPU6050_tockn.h>
```

Install Arduino Nano ESP32 board

The screenshot shows the Arduino IDE interface. At the top, a dropdown menu is set to 'Arduino Nano ESP32'. The 'BOARDS MANAGER' tab is active on the left. A red box highlights the 'arduino nano esp32' package, with a red circle '2' next to it. Another red box highlights the 'INSTALL' button, with a red circle '3' next to it. Below this, the 'esp32' package by Espressif Systems is shown with a '3.3.6 installed' status and an 'UPDATE' button. The main editor area shows a code file named 'Arduino_ReactionWheel_ESP32-S3-Nano.ino' with the following code:

```
3 // -----PINS-----
4 #define PIN_DIR 2
5 #define PIN_STEP 3
6 #define PIN_SLEEP 4
7 #define PIN_LED 13
8 // -----
9
10 // -----STEPPER-----
11 #include "AccelStepper.h"
12 AccelStepper myStepper(1, PIN_STEP, PIN_DIR);
13 double motorSpeed = 0;
14 #define MICROSTEPPING 4 /* 1 or 2 or 4 or 8 or 16 or 32 */
15 #define ACCELERATION 1750 /* Steps per s */
16 #define MAX_SPEED (600 * MICROSTEPPING)
17 // -----
18
19 // ----- UDP -----
20 const char *remoteIP = "192.168.4.2";
21 uint16_t remotePort = 2222;
22 WiFiUDP Udp;
23 //-----
24
```

Install the library required for sensor

Self-BalanceWithReaction_wheel_ESP32_S3_Nano | Arduino IDE 2.3.5

File Edit Sketch Tools Help

Arduino Nano ESP32

LIBRARY MANAGER

mpu6050_tockn

Mpu6050_tockn

MPU6050_tockn by tockn

1.5.2 installed

Arduino library for easy communicating with the MPU6050. It can get accel, gyro, and angle data. More info

1.5.1

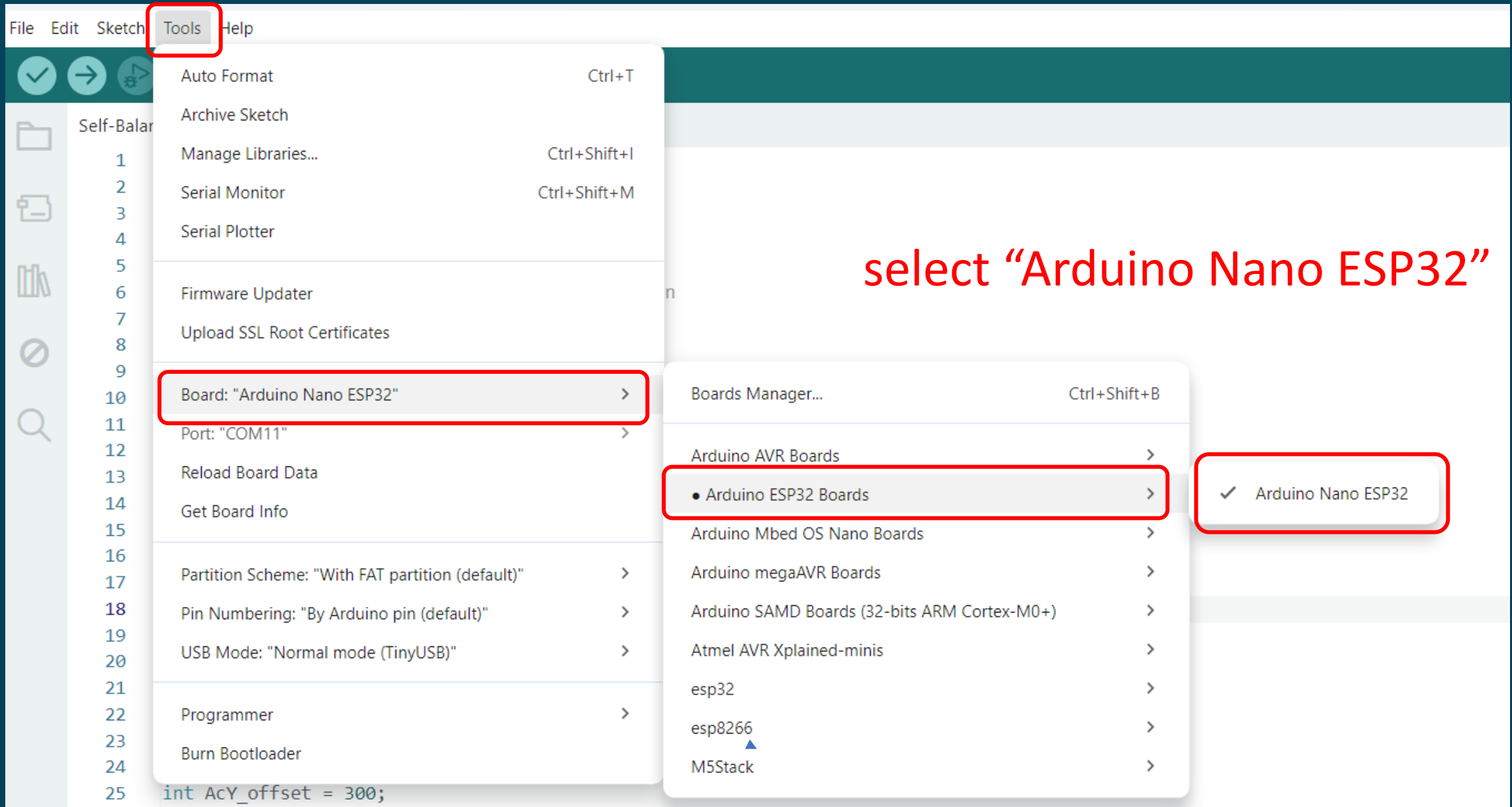
INSTALL

TinyMPU6050 by Gabriel Milan
<gabriel.gazola@poli.ufrj.br>
Tiny implementation for MPU6050

Self-BalanceWithReaction_wheel_ESP32_S3_Nano.ino

```
1 #include <Wire.h>
2 #include <MPU6050_tockn.h>
3
4 MPU6050 mpu6050(Wire);
5 #define BRAKE 8 // Stop motor pin
6 #define PWM 9 // speed control pin
7 #define DIRECTION 4 // rotation dir pin
8 #define LEDSIGNAL 12 // LED pin
9
10 const int freq = 20000; // 20 kHz
11 const int resolution = 8; // 8-bit (0-255)
12 float loop_time = 10;
13
14 float Kp = 76;
15 float Kd = 5.3;
16 float Kx = 0.05;
17
18 int pwm_s = 0;
19 int motor_speed;
```

Coding with the Arduino IDE



Coding with the Arduino IDE

Self-BalanceWithReaction_wheel_ESP32_S3_Nano.ino

```
1  #include <Wire.h>
2  #include <MPU6050_tockn.h>
3
4  MPU6050 mpu6050(Wire);
5  #define BRAKE 8           // Stop motor pin
6  #define PWM 9            // speed control pin
7  #define DIRECTION 4      // rotation dir pin
8  #define LEDSIGNAL 12     // LED pin
9
10 const int freq = 20000;   // 20 kHz
11 const int resolution = 8; // 8-bit (0-255)
12 float loop_time = 10;
13
14 float Kp = 76;
15 float Kx = 0.05;
16 float Kd = 5.3;
17
18 int pwm_s = 0;
19 int motor_speed;
20 int32_t timer;
21 long currentT, previousT;
22
23 //IMU offset values
24 int AcX_offset = -1300;
25 int AcY_offset = 300;
26 int GyZ_offset = 0;
27 int AcX, AcY, GyZ, gyroZ;
28
```

Pin 8: Acts as an electronic **Brake** to lock the motor when needed
Pin 4 & 9: Handle the direction and speed of the motor
Pin 12: Flashes an LED at startup to tell you the calibration is done

The loop() runs every 10 milliseconds

Kp: The Angle Gain (Proportional)

It handles the current tilt error. Higher values make the motor push harder to fix the angle.

Too low: The robot will feel "mushy" and slowly fall over because the motor isn't pushing hard enough.

Too high: The robot will vibrate or oscillate violently back and forth as it over-corrects.

Kd: The Angular Velocity Gain (Derivative)

It looks at the speed of the fall (angular velocity) to dampen the movement and prevent overshooting.

Too low: The robot will overshoot the center point and keep swinging.

Too high: The motor might feel jittery because it's reacting to every tiny vibration in the sensor.

Kx: The Speed Feedback

It monitors the accumulated motor speed to try and keep the device from drifting across the plane.

Too low: The robot balances but "wanders" or drifts across the table.

Too high: The robot will sacrifice its balance just to keep the wheels still, likely causing it to fall over.

Coding with the Arduino IDE

```
24 int AcX_offset = -1300;
25 int AcY_offset = 300;
26 int GyZ_offset = 0;
27 int AcX, AcY, GyZ, gyroZ;
28
```

These correct for the physical mounting of the sensor.
By setting AcX_offset = -1300, the code will subtract 1300 whenever it gets the reading from the sensor

```
29 #define Gyro_amount 0.996 // percent of gyro in complementary filter
30 float alpha = 0.40;
31 float gyroZfilt;
32 float device_angle;
33 float Acc_angle;
34 bool vertical = false;
35
```

```
36 void angle_calc() {
37     mpu6050.update();
38     AcX = mpu6050.getRawAccX() + AcX_offset;
39     AcY = mpu6050.getRawAccY() + AcY_offset;
40     GyZ = mpu6050.getRawGyroZ() - GyZ_offset;
41
```

The code uses a **Complementary Filter** to figure out the angle from **0° point**: It reads the **Accelerometer** (stable but noisy) and the **Gyroscope** (smooth but drifts).

$\text{device_angle} = (0.996 * \text{Gyro}) + (0.004 * \text{Accel})$

99.6% of the result comes from the Gyroscope. This keeps the movement smooth and ignores the "noise" and vibrations of the motors.

0.4% of the result comes from the Accelerometer. This tiny amount acts as a "correction" that slowly pulls the angle back to reality, fixing the Gyroscope's drift over time.

```
42 device_angle += GyZ * loop_time / 1000 / 65.536;
43 Acc_angle = atan2(AcY, -AcX) * 57.2958; // + keepAngle;
44 device_angle = device_angle * Gyro_amount + Acc_angle * (1.0 - Gyro_amount);
45 if (abs(device_angle) > 9) {
46     vertical = false;
47 }
48 if (abs(device_angle) < 0.3) {
49     vertical = true;
50 }
51 }
```

The "Vertical" Logic

Balance Mode: If the angle is very close to upright (less than 0.3°), vertical becomes true and the balancing loop starts.

Safety Cutoff: If the device falls past 9°, it cuts the power (vertical = false) to prevent the motor from burning out.

Coding with the Arduino IDE

```
53 void setup() {  
54   Serial.begin(115200);  
55   pinMode(BRAKE, OUTPUT);  
56   pinMode(DIRECTION, OUTPUT);  
57   pinMode(LED SIGNAL, OUTPUT);  
58   digitalWrite(BRAKE, LOW);  
59   digitalWrite(PWM, LOW);  
60   delay(10);  
61   ledcSetup(0, freq, resolution);  
62   ledcAttachPin(PWM, 0);  
63   Wire.begin();  
64   mpu6050.begin();  
65   mpu6050.calcGyroOffsets(true);  
66   delay(3000);  
67   ledLight();  
68 }
```

`mpu6050.calcGyroOffsets(true);`

This line automatically calculates the gyro offset every time you turn the device on (which is why you must keep the device perfectly still for the first few seconds of power-up).

```
70 void loop() {  
71   currentT = millis();  
72   if (currentT - previousT >= loop_time) {  
73     angle_calc();  
74     if (vertical) {  
75       digitalWrite(BRAKE, HIGH);  
76       gyroZ = GyZ / 131.0; // Convert to deg/s  
77       gyroZfilt = alpha * gyroZ + (1 - alpha) * gyroZfilt;  
78       pwm_s = -constrain(Kp * device_angle + Kx * -motor_speed + Kd * gyroZfilt, -255, 255);  
79       Motor_control(pwm_s);  
80       motor_speed += pwm_s;  
81       previousT = currentT;  
82     } else {  
83       Motor_control(0);  
84       digitalWrite(BRAKE, LOW);  
85     }  
86   }  
87 }
```

Smooth out the data:

It takes **40%** of the new, raw measurement (gyroZ).

It takes **60%** (1 - 0.40) of the previous filtered value (gyroZfilt).

`constrain` chops that number down to the maximum limit (255) so the software doesn't crash.

Discussion

- Will AI technology help self-balancing systems work better?
- Which tasks can AI help with?
- Should traditional PID controllers be replaced by AI?

3 min

For AI to generate the code, what should we tell it?

- Physical layout of the self-balancing device (**set point = 0°**)
- Essential components used to provide the control on its own
 - MCU (**ESP32-S3-Nano Development Board**)
 - Accelerometer & Gyroscope Module (**MPU6050**) and its placement **orientation** (**spinning about z-axis**)
 - BLDC motor and the assigned MCU **pins** that control its speed, direction of spin, and brake
- Integrated development environment (IDE) to be used (**Arduino**)
- Algorithm to be used (**PI, PD, PID, others or do not specified**)

Generate Arduino code for a self-balancing device that employing the ESP32-S3-Nano Development Board as its microcontroller, with the MPU-6050 oriented such that the x-axis faces downward and the z-axis extends outward; the BLDC motor driving a reaction wheel receives its PWM speed signal via MCU pin 9, while pin 8 manages the braking mechanism and pin 4 governs rotational direction; the target set point is defined as a yaw angle of zero, and a PID control algorithm shall be implemented.

Try the following prompt to request AI for code generation

Generate Arduino code for a self-balancing device that employing the ESP32-S3-Nano Development Board as its microcontroller, with the MPU-6050 oriented such that the x-axis faces downward and the z-axis extends outward; the BLDC motor driving a reaction wheel receives its PWM speed signal via MCU pin 9, while pin 8 manages the braking mechanism and pin 4 governs rotational direction; the target set point is defined as a yaw angle of zero, and a PID control algorithm shall be implemented.

Consideration while asking AI:

- Non-deterministic, particularly in Large Language Models (LLMs), the **same prompt** can produce **different outputs** at **different times**
- May use prompt interactively. Try asking for something essential and ask AI to modify or adjust its output

Google AI Mode Search



AI Mode ▾ All Images Videos News More ▾



Generate Arduino code for a self-balancing device that employing the ESP32-S3-Nano Development Board as its microcontroller, with the MPU-6050 oriented such that the x-axis faces downward and the z-axis extends outward; the BLDC motor driving a reaction wheel receives its PWM speed signal via MCU pin 9, while pin 8 manages the braking mechanism and pin 4 governs rotational direction; the target set point is defined as a yaw angle of zero, and a PID control algorithm shall be implemented.

This Arduino code implements a **PID control loop** for a self-balancing reaction wheel using an **ESP32-S3-Nano**.

Hardware Orientation & Logic

- **MPU-6050:** Oriented with the **X-axis downward** (gravity vector) and **Z-axis outward**. In this specific orientation, tilt/balance deviations are measured as rotation around the **Y-axis** (Pitch) or **Z-axis** (Yaw) depending on your mechanical frame's definition of "forward." Given your "yaw zero" set point, this code treats the Z-axis rotation as the primary balancing feedback.
- **Pins:** Motor Speed (Pin 9), Brake (Pin 8), and Direction (Pin 4).
- **Control:** A Complementary Filter combines accelerometer and gyroscope data to provide a stable angle estimate, which the PID controller uses to drive the BLDC motor.

GPIO & RTC GPIO - ESP32-S3 - — ESP-IDF Programming Guide v6 ...

The ESP32-S3 chip features 45 physical GPIO pins (GPIO0 ~ GPIO21 and GPIO26 ~ GPIO48). Each pin can ...

Espressif Systems

ESP32 - Wikipedia

GPIO Pins. 34. 43. 45. 20. 22. 30. 26. 50. SAR ADCs. 2 × 12-bit. 2 × 13-bit. 2 × 12-...



Wikipedia

Design and Control of Two-wheeled Self-Balancing Robot using Arduino

PID controller controls it through the program written in Arduino IDE. A detailed description of constituting...

IEEE

Arduino Two Wheel Self-



Google AI Mode Search



AI Mode ▾ All Images Videos News More ▾

Implementation Code

cpp

```
#include <Wire.h>
#include <MPU6050.h>

// --- Pin Definitions ---
const int PIN_PWM = 9;      // Motor speed (PWM)
const int PIN_BRAKE = 8;    // Motor brake
const int PIN_DIR = 4;      // Motor direction

// --- PID Constants (Tune these for your hardware) ---
double Kp = 25.0;
double Ki = 0.5;
double Kd = 1.2;

// --- Control Variables ---
double setPoint = 0.0;      // Target yaw angle
double inputAngle = 0.0;    // Current angle
double outputPWM = 0.0;     // PID output
double errorSum = 0, lastError = 0;
unsigned long lastTime;

// --- MPU6050 & Filter Variables ---
MPU6050 mpu;
float gyroZ, accX, accY, angleZ = 0;
const float alpha = 0.98;   // Complementary filter coefficient
```

Arduino Two Wheel Self-Balancing Robot - DigiKey

10 Mar 2025 — Arduino Two Wheel Self-Balancing Robot * Now, let's briefly...

DigiKey



Modeling and Simulation of Self-stabilizing Platform for Industria...

20 Feb 2025 — The work presented addresses the creation of a self-...

Springer Nature Link



Development and Prototyping of a Two-Wheeled Self- Balancing Robot

23 Nov 2024 — A complementary filter is used with a triple-axis gyroscope and accelerometer to accurately...

Maldives Journal of Engineering a...

Balancing Robot Maths !

This block calculates the pitch angle (how much the robot is leaning) of the...

Instructables



Measure Angles Easily with MPU6050 and...

Measure Angle

Google AI Mode Search



```
void setup() {
  Serial.begin(115200);
  Wire.begin();

  // Initialize Motor Pins
  pinMode(PIN_PWM, OUTPUT);
  pinMode(PIN_BRAKE, OUTPUT);
  pinMode(PIN_DIR, OUTPUT);
  digitalWrite(PIN_BRAKE, LOW); // Release brake

  // Initialize MPU6050
  mpu.initialize();
  if (!mpu.testConnection()) {
    Serial.println("MPU6050 connection failed");
    while (1);
  }

  // Initial Calibration: Assume start position is roughly zero
  lastTime = millis();
}

void loop() {
  unsigned long currentTime = millis();
  double dt = (double)(currentTime - lastTime) / 1000.0;

  // 1. Read Sensor Data
  int16_t ax, ay, az, gx, gy, gz;
  mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);

  // 2. Calculate Angle (Complementary Filter)
  // With X down and Z out, rotation around Z is the balancing axis.
  // Accel angle using atan2 for stability
  double accelAngleZ = atan2((double)ay, (double)ax) * 180 / PI;
  double gyroRateZ = (double)gz / 131.0; // FS_SEL=0 (250 deg/s)

  inputAngle = alpha * (inputAngle + gyroRateZ * dt) + (1 - alpha) * accelAngleZ
```



Springer Nature Link

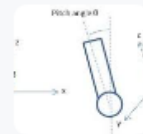
Development and Prototyping of a Two-Wheeled Self- Balancing Robot

23 Nov 2024 — A complementary filter is used with a triple-axis gyroscope and accelerometer to accurately...

Maldives Journal of Engineering a...

Balancing Robot Maths !

This block calculates the pitch angle (how much the robot is leaning) of the...



Instructables

Measure Angles Easily with MPU6050 and...

11 Sept 2023 — what's going on guys welcome back to th...



YouTube · Shilleh

Angle Controller — SunFounder Showcase of Creative Projects...

To install the library, use the Arduino Library Manager and search for...



SunFounder

MPU 6050 Tutorial | How to Program MPU 6050 With Arduin...

15 May 2020 — Step 4: Install Libraries Before we start Coding, we will need a...



Instructables

Google AI Mode Search



```
// 2. Calculate Angle (Complementary Filter)
// With X down and Z out, rotation around Z is the balancing axis.
// Accel angle using atan2 for stability
double accelAngleZ = atan2((double)ay, (double)ax) * 180 / PI;
double gyroRateZ = (double)gz / 131.0; // FS_SEL=0 (250 deg/s)

inputAngle = alpha * (inputAngle + gyroRateZ * dt) + (1 - alpha) * accelAngleZ

// 3. PID Calculation
double error = setPoint - inputAngle;
errorSum += error * dt;
double dError = (error - lastError) / dt;

outputPWM = (Kp * error) + (Ki * errorSum) + (Kd * dError);
lastError = error;
lastTime = currentTime;

// 4. Drive Motor
controlMotor(outputPWM);

// Debugging
Serial.print("Angle: "); Serial.print(inputAngle);
Serial.print(" Output: "); Serial.println(outputPWM);

delay(10); // 100Hz loop
}

void controlMotor(double speed) {
  // Set Direction
  if (speed >= 0) {
    digitalWrite(PIN_DIR, HIGH);
  } else {
    digitalWrite(PIN_DIR, LOW);
  }
  speed = -speed; // Absolute value for PWM
}
```



Springer Nature Link



Development and Prototyping of a Two-Wheeled Self-Balancing Robot

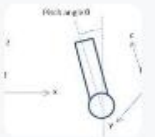
23 Nov 2024 — A complementary filter is used with a triple-axis gyroscope and accelerometer to accurately...

Maldives Journal of Engineering a...



Balancing Robot Maths !

This block calculates the pitch angle (how much the robot is leaning) of the...



Instructables

Measure Angles Easily with MPU6050 and...

Measure Angle



11 Sept 2023 — what's going on guys welcome back to th...

YouTube · Shilleh



Angle Controller — SunFounder Showcase of Creative Projects...

To install the library, use the Arduino Library Manager and search for...



SunFounder

MPU 6050 Tutorial | How to Program MPU 6050 With Arduin...

15 May 2020 — Step 4: Install Libraries Before we start Coding, we will need a...



Instructables

Google AI Mode Search



```
// Constrain PWM (ESP32-S3 uses 8-bit or higher resolution)
int pwmValue = constrain((int)speed, 0, 255);

// Apply Speed
analogWrite(PIN_PWM, pwmValue);

// Safety: If angle is too extreme, brake the wheel
if (abs(inputAngle) > 45) {
  digitalWrite(PIN_BRAKE, HIGH);
  analogWrite(PIN_PWM, 0);
} else {
  digitalWrite(PIN_BRAKE, LOW);
}
}
```

Use code with caution.



Key Setup Tips

- **Library:** This code uses the Electronic Cats MPU6050 Library. Ensure you install it via the Arduino Library Manager.
- **PWM Resolution:** The ESP32-S3-Nano typically uses `analogWrite` for ease, but for high-precision BLDC control, you may eventually want to use the [LEDC Peripheral](#) for higher frequency PWM.
- **Tuning:** Start with `ki` and `kd` at zero. Increase `kp` until the device oscillates, then add `kd` to dampen the movement. Espressif Systems +4

Would you like a specific **wiring diagram** to verify the I2C and power connections for the ESP32-S3-Nano?

10 sites

GPIO & RTC GPIO - ESP32-S3 - — ESP-IDF Programming Guide v6 ...

The ESP32-S3 chip features 45 physical GPIO pins (GPIO0 ~ GPIO21 and GPIO26 ~ GPIO48). Each pin can be used as a...

Espressif Systems

ESP32 - Wikipedia

GPIO Pins. 34. 43. 45. 20. 22. 30. 26. 50. SAR ADCs. 2 × 12-bit. 2 × 13-bit. 2 × 12-bit. 1 × 12...



Wikipedia

Design and Control of Two-wheeled Self-Balancing Robot using Arduino

PID controller controls it through the program written in Arduino IDE. A detailed description of constituting material...

IEEE

Show all

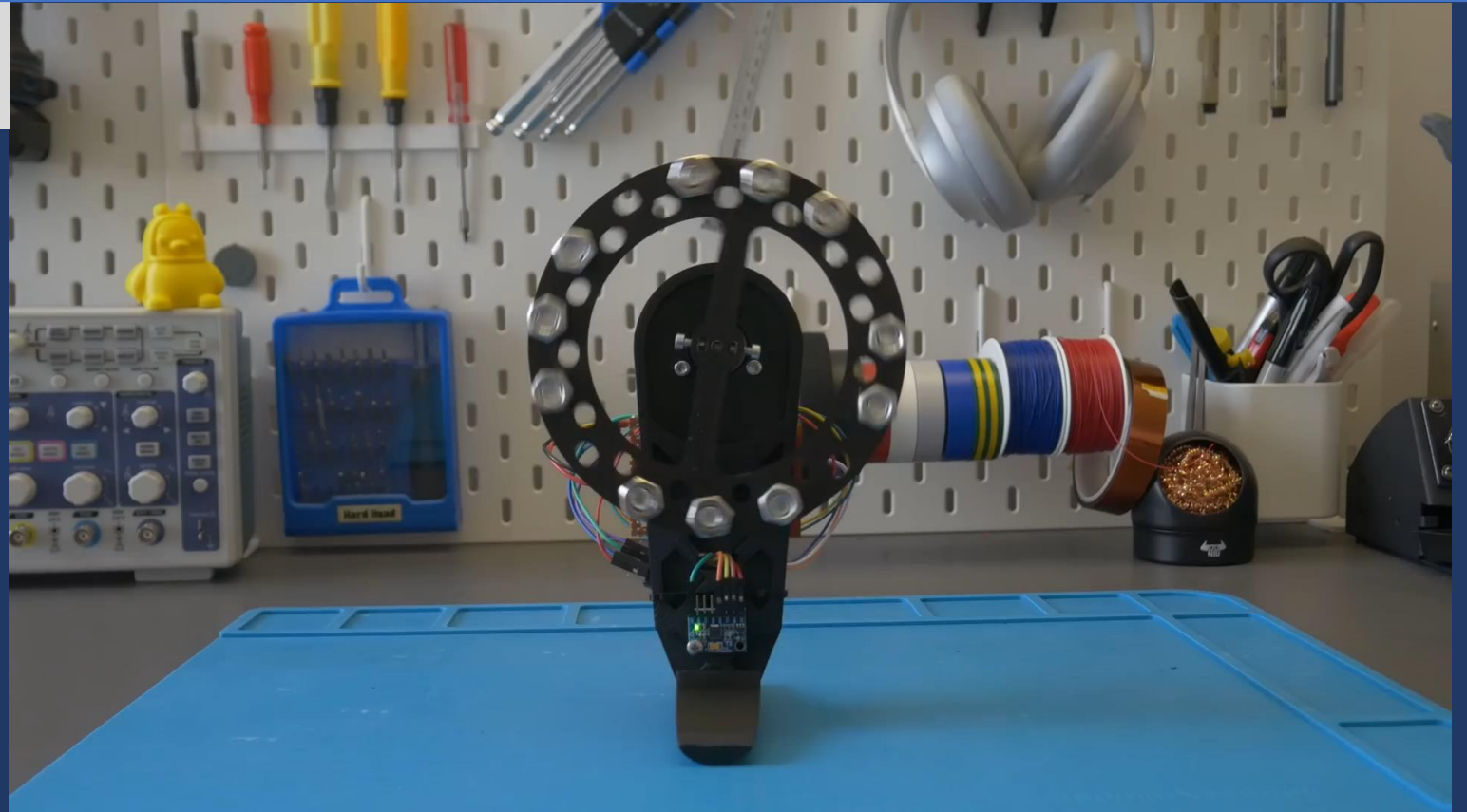
Outcome from implementing the AI-generated code

- Compilation through the Arduino IDE completed **without errors**
- **Finding the optimal values for K_p , K_i , and K_d was likely the core difficulty** that kept the device from reaching a self-balanced state.

Reaction Wheel Took Me 3 Years to Make Stable

That which at first seemed manageable (after watching a few YouTube tutorials) quickly turned out to be a much harder problem to solve than expected.

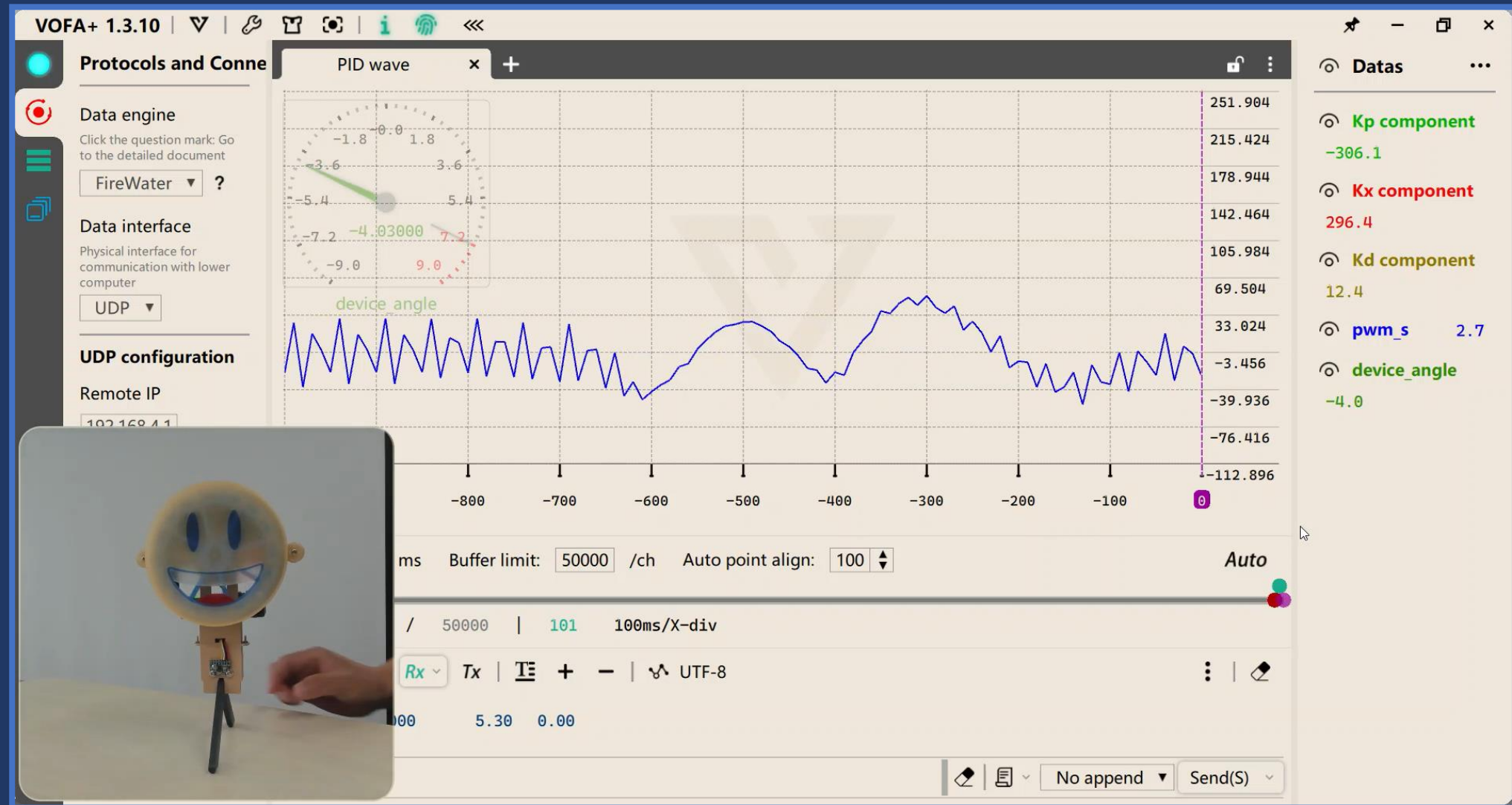
Having already constructed a custom Arduino-based drone prior to initiating this project in 2019, the maker experienced significant frustration during prolonged periods of stagnation in progress.



<https://www.youtube.com/watch?v=MHxw4yaNKVw&t=17s>

Finding the optimal values for K_p , K_i , and K_d was likely the core difficulty that kept the device from reaching a self-balanced state.

Any additional measures to support?



What Can We Learn from the PID? (Summary)

- Control is all about **decision-making** to reach a certain **goal**;
- By **observing the performance** of different controllers in various environments, we are able to learn things about **decision-making in general** and reflect it back on it.
- This **slightly philosophical part** is the most interesting part of the field of Systems & Control.
- Perhaps the most important thing we learn from the PID controller is that it is all about **finding a balance**.
- **Too little reaction** to error means it takes a **long time** to reach the goal; **too aggressive** a response throws the system into **chaos**.