Project Embedded Control

Self Balancing Triangle

Students:
Grande Alessandro Francesco
Campobasso Maria, Campanale Raffaele

Contents

1 Objectives of the project

The aim of this project is to create a self-balancing triangle using the STM32F446RE microcontroller and implement an appropriate control loop. Inspired by other projects, the mechanical design was created using Blender software and 3D printed with the C3Lab printer at the Politecnico di Bari. The control loop was implemented using STM32CubeIDE software. The main challenge of this project was achieving real-time stabilization using PID control, taking into account mechanical imprecisions and sensor noise.

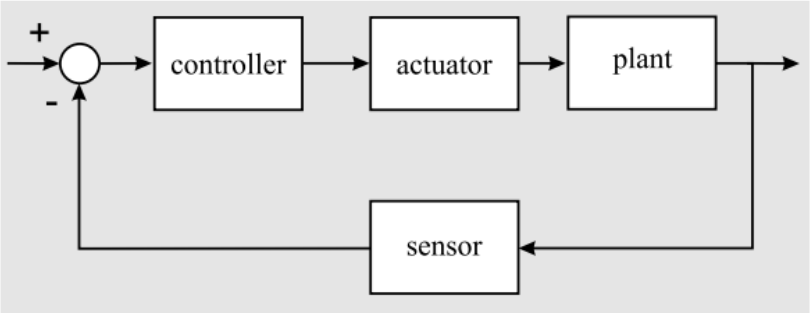

Figure 1: Embedded Control Loop.

2 Selected Hardware

The MPU6050 is a MEMS (Micro-Electro-Mechanical System) sensor that integrates a 3-axis accelerometer and a 3-axis gyroscope, making it a 6-degree-of-freedom (6DOF) device. This makes it ideal for applications requiring motion and orientation detection, such as robotics and drones. The MPU6050 uses the I^2C (Inter-Integrated Circuit) protocol for communication with microcontrollers such as Arduino, ESP32, STM32, and other embedded devices.

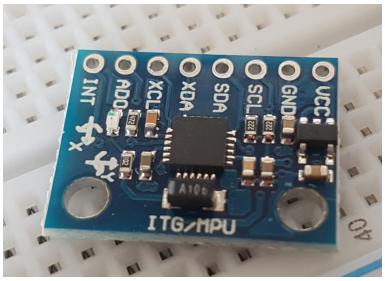

Figure 2: MPU6050 Sensor.

The Nidec 24H is a series of brushless DC motors designed for high efficiency, long lifespan, and precise position control. The Nidec 24H motor operates at 24V, with an operational range from 22.8V to 25.2V. One of its main features is its low operating current, reaching up to 0.7A, making it energy-efficient. The motor has an output power of approximately 11W and a nominal torque of around 25 mN·m. A 70 cm flywheel was attached to this motor, in accordance with the 3D-printed self-balancing triangle model. The choice of this motor was driven by its nominal speed of 4200 RPM, a no-load speed of up to 5800 RPM, as well as its lightweight and compact dimensions. Additionally, this motor features a PWM pin, allowing for modulation of the supply voltage, enabling speed variation via a PWM signal at 20 kHz frequency. The Nidec 24H also includes an encoder powered at 5V, but it was not used as feedback for the control loop.

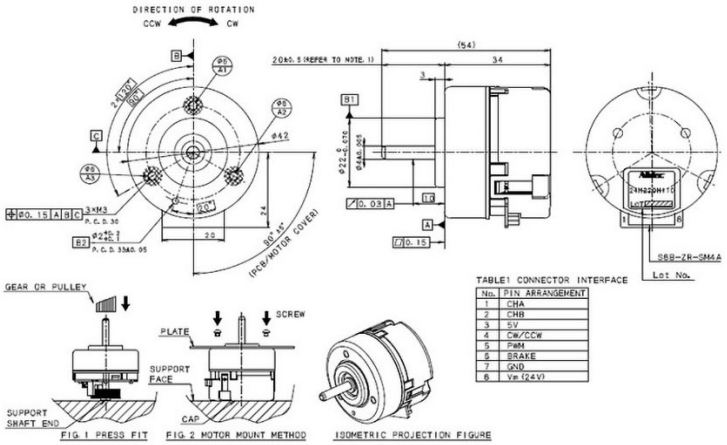

Figure 3: Nidec Motor.

Since the power levels involved were relatively low, various connections were made using jumper wires. To manage the control and power signals for the motor, a breadboard was used, which, thanks to the use of LEDs, proved to be particularly useful during the debugging phases of the self-balancing triangle. To power the Nidec 24V motor, a bench power supply was used, which can be replaced by a 500mAh 3S1P LiPo battery.

3 Design choices

Based on the 3D models from other makers, particularly from a project related to a self-balancing cube, the 3D model of the self-balancing triangle has been split into two parts to allow assembly with M5 screws and to simplify the installation of the Nidec 24H motor and the MPU6050 sensor. The model's shape is an isosceles triangle, designed to reduce the centrifugal force required to keep it balanced, with well-beveled angles. This choice was made to improve the stability and balance point of the self-balancing triangle. A series of 3D models were created until the right one was found. The main challenge was printing the screw holes and the pillars that help secure the structure.

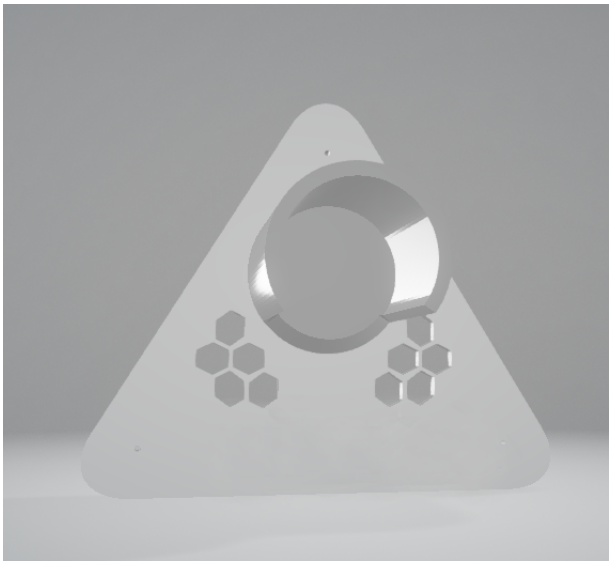

Figure 4: Front Model 3D.

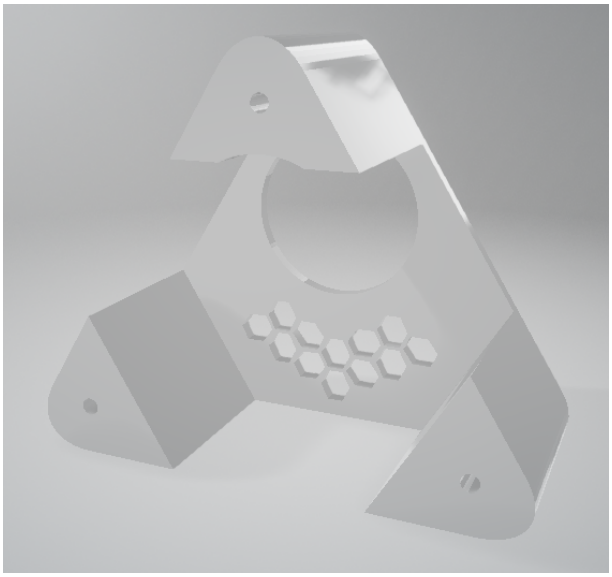

Figure 5: Front Model 3D.

Regarding the wiring and pin configuration on the STM32 microcontroller, the pins were assigned to optimize the connection of the various components. In particular, the SDA pin of the I2C1 interface was relocated to pin PB9. Since the PA5 pin is preconfigured to be connected to the STM32's L2D LED and, to control the rotation direction of the Nidec 24H motor, a voltage signal ranging from 0 to 5V is needed, it was decided to use this pin, also taking advantage of the feedback from the LED itself. The PWM signal, used to regulate the motor's speed, was configured on pin PA0. Moreover, it is essential to connect the ground of the STM32, the MPU6050 sensor, and the Nidec 24H motor: a task accomplished via the breadboard to ensure a proper ground reference. The pin nomenclature follows that of STM32CUBEMX; the STM32 manual was used to map these pins. The 5V pin was used for enabling the Nidec 24H, while the 3.3V pin was used to power the MPU6050.

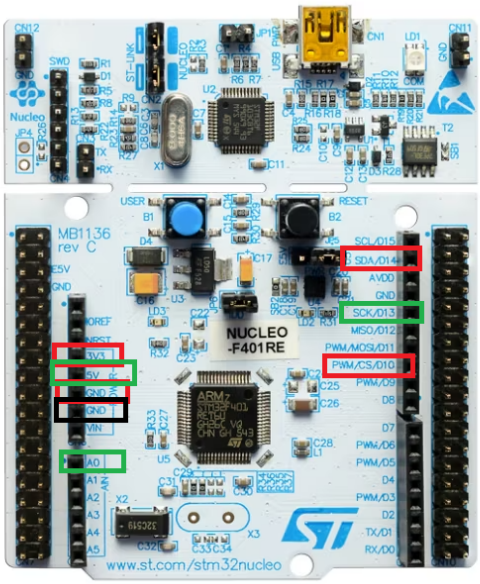

Figure 6: Image of pin settings. Red for MPU6050, Green for NIDEC24H and Black for common ground.

The Kalman filter was selected to combine accelerometer and gyroscope data to calculate the pitch angle. The gyroscope provides high-frequency data with noise, while the accelerometer corrects long-term drift. This approach ensures stable and accurate angle measurements critical for PID control. To achieve precise speed modulation and maintain real-time control, the PWM (Pulse Width Modulation) signal was utilized. The TIM2 peripheral was configured to generate PWM signals, which allowed the Nidec 24H motor to respond dynamically to changes in the pitch angle. The use of three separate timers (TIM10, TIM7 and TIM6) ensured that different operations, such as sensor reading, Kalman filter and PID calculations, were executed at precise intervals. TIM10 was dedicated to updating sensor data, while TIM6 handled the control loop calculations. Furthermore, the Nidec 24H motor was chosen for its low current consumption and efficiency, ensuring that the project could operate for extended periods when powered by a LiPo battery or bench power supply. The system was designed to be modular, allowing easy replacement or upgrades of components such as sensors or motors. This also facilitated debugging and future improvements.

The Blue Button, configured as an interrupt with Falling Edge Trigger Detection, was used to turn the Self-Balancing Triangle on and off.

4 Controller Design

First, the MPU6050 sensor was configured. Analyzing the datasheet, the full-scale range of the gyroscope was set to ±250 °/s, and that of the accelerometer to ±2g, writing the appropriate values to the dedicated registers of the sensor.

GYRO_CONFIG

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1B	27	XG_ST	YG_ST	ZG_ST	FS_SEL[1:0]		-	-	-

Figure 7: Gyro Config Register.

ACCEL_CONFIG

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1C	28	XA_ST	YA_ST	ZA_ST	AFS_SEL[1:0]		-		

Figure 8: Accel Config Register.

A PID controller was selected for its simplicity and effectiveness in balancing systems, allowing precise error correction through proportional, integral, and derivative terms. The proportional term reacts to the current error (difference between the setpoint and the measured pitch angle). A higher proportional gain (Kp) results in faster error correction but may introduce overshoot or oscillations. In this project, a Kp of 30 was initially chosen and later sperimentally fine-tuned. The integral term accumulates the past errors over time to correct for any steady-state offset. A small integral gain (Ki = 0.1) was used to avoid excessive oscillations and wind-up issues. Anti-windup logic was implemented to constrain the integral term within a specific range (-40.0 to 40.0). The derivative term predicts future errors based on the rate of change of the error. This helps in damping oscillations and improving stability. A derivative gain (Kd = 0.5) was chosen to balance responsiveness and smoothness. To ensure that the motor operates within its capabilities, the PID output was clamped between -100 and 100. This prevents extreme values that could cause instability or damage to the motor. The tuning process involved adjusting Kp, Ki, and Kd iteratively:

- Step 1: Start with a high Kp to observe the system's response. Adjust Kp to achieve fast error correction.
- Step 2: Introduce Ki to eliminate steady-state error. Keep Ki small to prevent instability.

- Step 3: The derivative term predicts future errors by considering the rate of change of the error, helping to dampen oscillations. A value of 0.5 was found to provide adequate damping, ensuring smooth corrections without excessive delay.

The PID parameters were adjusted iteratively by testing the system in real-world conditions. Each iteration involved observing the system's behavior, making adjustments to the gains, and retesting. The final tuned values (Kp = 30, Ki = 0.1, Kd = 0.5) were validated by testing the triangle under various conditions, including different inclinations and sudden disturbances. The system consistently maintained balance and returned to the upright position within a short time frame. The setpoint was chosen as 0.15 degrees, representing the desired upright position, whereas the PID output is scaled to match the motor speed range, ensuring that the motor can respond appropriately to correct angular deviations. The motor direction is controlled using a GPIO pin based on the sign of the PID output. Positive PID output indicates the motor should rotate in one direction, while negative output indicates the opposite direction and the duty cycle is calculated based on the absolute value of the PID output. This approach adjusts the motor speed proportionally to the error magnitude, providing smooth and controlled movements.

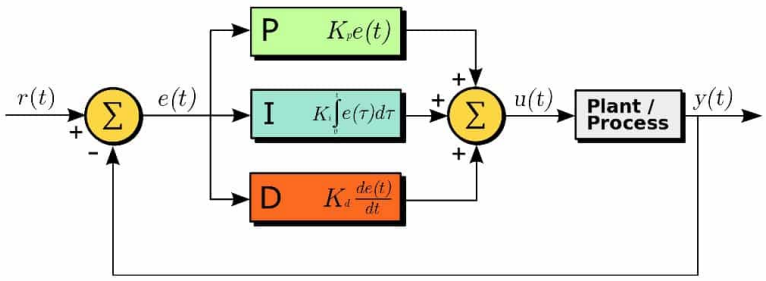

Figure 9: Pid Block Diagram.

The motor direction is controlled based on the value calculated by the PID controller, pid_output, which is the sum of the proportional, integral, and derivative terms of the PID. This is implemented using PWM combined with polarity control via a GPIO pin. Specifically, if pid_output >0, it means the

system must act in one direction (for example, the motor should accelerate to correct the error in a clockwise direction). If pid_output <0, it means the system must act in the opposite direction (for example, the motor should correct in a counterclockwise direction).

Once the direction is determined, the PWM duty cycle is set to regulate the motor speed. The duty cycle is calculated based on the absolute value of pid_output.

5 Quantitative experimental results

At the beginning of the project, a complementary filter was used, which calculates a weighted sum between the angle measured by the accelerometer and that derived from the gyroscope. However, even after calibration of the MPU6050 sensor, the system response was excessively slow. To overcome this problem, the use of the Kalman filter was chosen, using the angle measured by the accelerometer and the angular velocity along the Y axis as inputs.

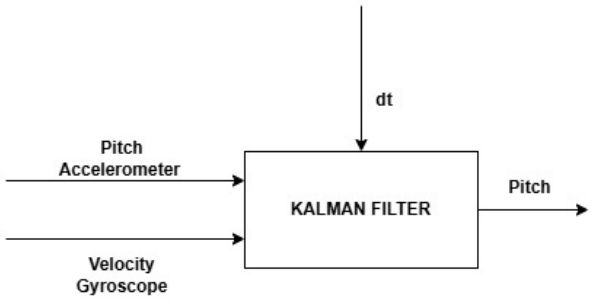

Figure 10: Kalman Filter Block Diagram.

The Kalman filter proved particularly suitable for implementing closed-loop control of the system.

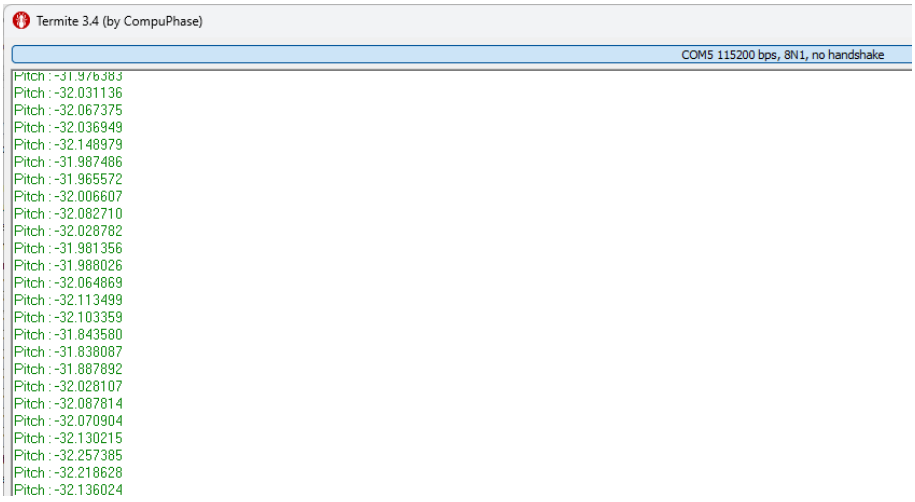

Figure 11: Values obtained from the Kalman Filter using the UART Serial Communication.

The main issue with the project concerned the robot's structure. Since the frame was not particularly rigid, vibrations generated by the drive of

the Nidec 24H motor interfered with MPU6050 measurements, especially at high speed. Instead of redesigning the chassis, vibration damping grommets were used to mitigate such oscillations. In addition, a software approach was experimented with, implementing a Notch filter and a bandpass filter, but neither resulted in significant improvement.

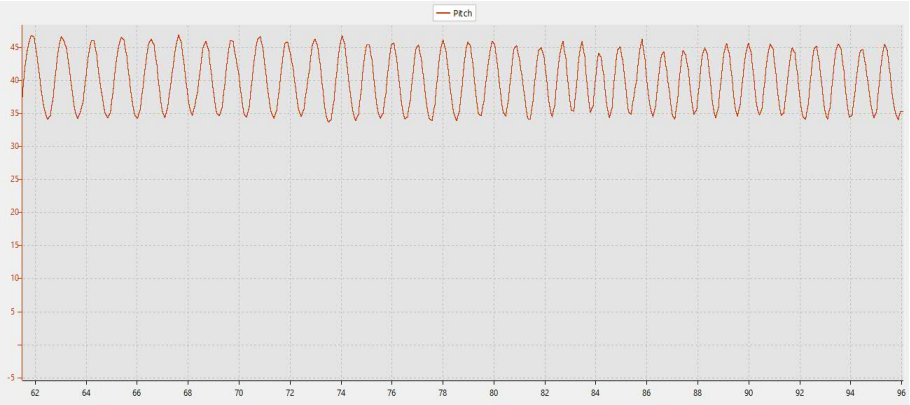

Figure 12: Pitch Angle graph at full speed.

Through the use of vibration dampers, motor interference on MPU6050 measurements has been significantly reduced.

Figure 13: Pitch Angle graph at full speed with dampers. At time t=48, the position of the triangle was changed to observe the system's response to the change.

6 Conclusions

The PID controller proved to be a reliable method for balancing, with proportional, integral, and derivative terms working together to correct errors dynamically. The implementation of anti-windup logic and output clamping enhanced the system's stability. The Kalman filter significantly improved the accuracy of angle measurements by combining accelerometer and gyroscope data. This ensured that the PID controller received precise input, minimizing noise and measurement errors. The iterative tuning process and real-time debugging via UART were critical in optimizing the system's performance. The ability to monitor variables such as pitch, gyroscope values, and PID output provided valuable insights during the development process. By combining design choices and control strategies, the self-balancing triangle achieves real-time stabilization with robust handling of sensor noise and mechanical imprecisions. Future work could include incorporating an encoder for additional feedback, enhancing the system's precision, and exploring alternative control algorithms such as LQR (Linear Quadratic Regulator) for comparison with the PID controller.

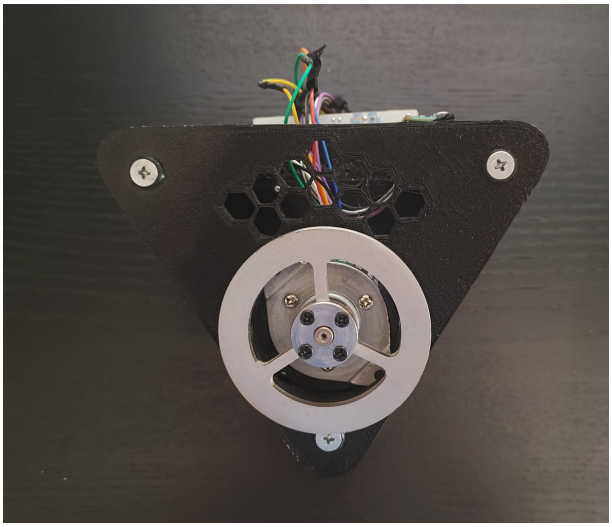

Figure 14: Self Balancing Triangle.