Report – PID Fundamentals

October 2023

Dinh-Son Vu

School of Science, Engineering, and Technology

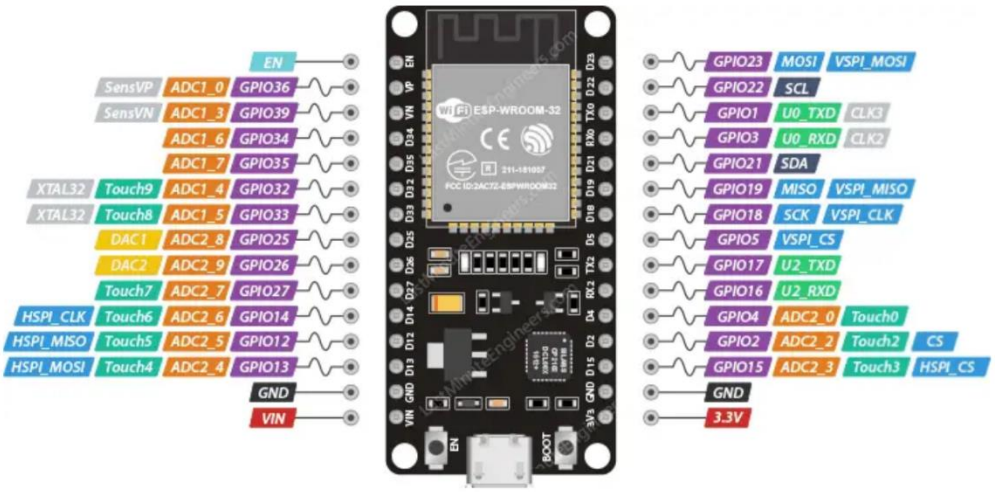
Table of Contents

1. Introduction 3
2. ESP32 Pinout 3
3. Presentation of the self-balancing robot 4
4. Feedback system 5
5. PID Tuning 6
   1.1. PID tuning of the self-balancing robot 6
   1.2. PID tuning of the drone 7

1. Introduction

This document introduces the use of the PID library with the ESP32. PID is a controller used in closed-loop systems, which requires at least a motor and a sensor. For this example, a self-balancing robot is used.

2. ESP32 Pinout



Label	GPIO	Safe to use?	Reason
D0	0	⚠	must be HIGH during boot and LOW for programming
TX0	1	✗	Tx pin, used for flashing and debugging
D2	2	⚠	must be LOW during boot and also connected to the on-board LED
RX0	3	✗	Rx pin, used for flashing and debugging
D4	4	✓	
D5	5	⚠	must be HIGH during boot
D6	6	✗	Connected to Flash memory
D7	7	✗	Connected to Flash memory
D8	8	✗	Connected to Flash memory
D9	9	✗	Connected to Flash memory
D10	10	✗	Connected to Flash memory
D11	11	✗	Connected to Flash memory
D12	12	⚠	must be LOW during boot
D13	13	✓	
D14	14	✓	
D15	15	⚠	must be HIGH during boot, prevents startup log if pulled LOW
D15	15	⚠	must be HIGH during boot, prevents startup log if pulled LOW
RX2	16	✓	
TX2	17	✓	
D18	18	✓	
D19	19	✓	
D21	21	✓	
D22	22	✓	
D23	23	✓	
D25	25	✓	
D26	26	✓	
D27	27	✓	
D32	32	✓	
D33	33	✓	
D34	34	⚠	Input only GPIO, cannot be configured as output
D35	35	⚠	Input only GPIO, cannot be configured as output
VP	36	⚠	Input only GPIO, cannot be configured as output
VN	39	⚠	Input only GPIO, cannot be configured as output

3. Presentation of the self-balancing robot

A self-balancing robot is an unstable system that is staying in the upward position, as shown in Figure 1. It uses an inertial measurement unit (IMU) based on the MPU6050 to measure the inclination angle. Two motors move the robot back and forward to adjust the inclination of the robot to the upward position.

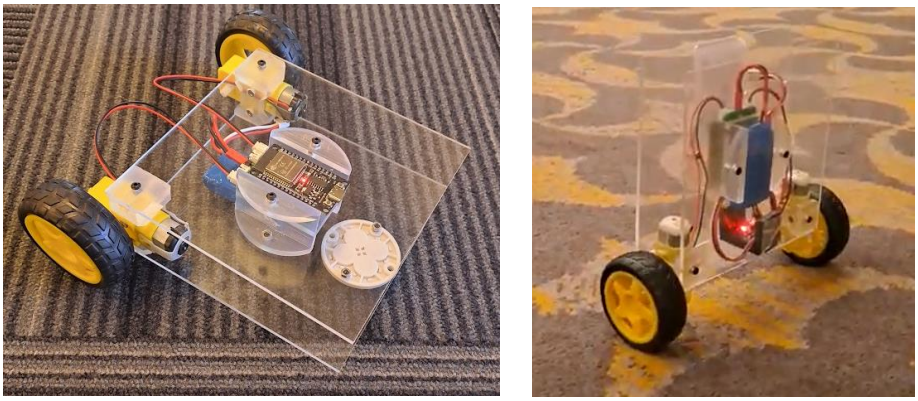


Figure 1: Self balancing robot in the horizontal position (at rest) and in the upward position (actively balancing)

The code uses the PID library. It is possible to code your own controller, however the library is a good introduction to understand the PID concept and their effect. Please look at the following link to understand the different functions of the library: https://playground.arduino.cc/Code/PIDLibrary/

This mini project is a good introduction to mechatronics design as it involves:

- Mechanical assembly
- Electrical wiring
- Microcontroller coding
- Control systems

Moreover, the self-balancing robot project involves several libraries:

- MPU6050 library
- PID library

Even though it is possible to code everything in a single file, it is much better to separate the main file into smaller pieces of codes. Please have a look on the following GitHub link: https://github.com/vudinhso/DroneProject/tree/main/04-PID-SelfBalancing. It includes:

- main.cpp: the main file
- MyMotorConfig.h: the library used to configure the DC motors
- MyMPU.h: the library used to configure the MPU6050
- MyPID.h: a library used to configure the PID
- MySerial.h: a library used to configure the serial communication

4. Feedback system

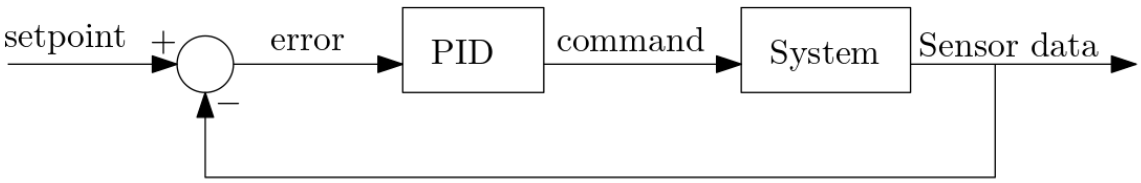


Figure 2: Feedback control system

A typical feedback control systems is given in Figure 2. The variables are defined as follows:

- Setpoint: this is the desired angle. In the upward position, the angle is 0°.
- Sensor data: this is the angle from the MPU6050, which give the inclination of the robot
- Error: this is the difference between the desired angle (setpoint) and the current angle of the system (sensor data). If there is no error, there is no needed to move the motor. If there is an error, then a command should be sent to the motor.
- Command: this is the signal sent to the motors to adjust the inclination of the robot

The system blocks are defined as follows:

- System: it corresponds to the self-balancing robot. It takes a motor signal as an input (between [0 – 255]) and it outputs the angle from the MPU6050.
- PID: it corresponds to the controller of the system. An aggressive controller stabilize the robot with a lot of oscillations. A stable controller brings the system to its equilibrium point within an acceptable time.

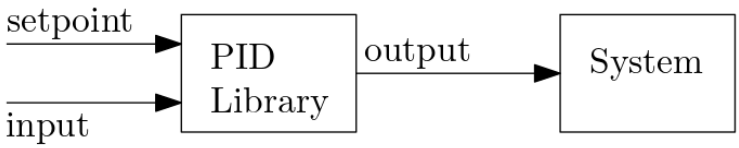


Figure 3: PID Library from Arduino

The PID library from Arduino uses the following terminology:

- Setpoint: the desired angle (usually 0°)
- Input: the measured angle from the MPU6050
- Output: the command sent to the motors.

5. PID Tuning

This section explains the method to tune the PID parameters for the self-balancing robot and the drone.

1.1. PID tuning of the self-balancing robot

At first, the PID parameters are set to zero. Control systems allow you to perform simulation to select suitable PID gains, however the transfer function of the system is not known. Thus, the PID parameter are determined experimentally (trial-and-error).

1. Increase the p-value until the robot can balance slightly with the help of the user ($P \approx 4$).
2. Increase the d-value until the robot can attenuate the overshoot of the robot (usually, it is not needed, but $D \approx 0.01$).
3. Increase the i-value until the robot can balance itself ($I \approx 2$).

The robot should be able to balance, but the PID controller is not magic: your system may not balance perfectly:

- The battery may be discharged, which reduces the power available for the robot
- The motor may be too weak to compensate the inertia of the robot
- The wheel may slip, which hinder the stability of the system
- The PID controller is not the best controller in terms of performance and stability. However, it is the easiest in terms of implementation on an embedded system.

A self-balancing robot is more stable on carpet than on hard surface. Adding double face tape to the wheels improve the stability of the system. It is important to keep track of the behavior of the robot while tuning the PID controller. A table can be used to summarize the performance and stability of the system.

Kp	Ki	Kd	Performance	Stability
1	0	0	Poor performance, very slow to balance	Does not balance yet
10	150	5	Balance to the upward position rapidly	The system is shaking a lot
…				

1.2. PID tuning of the drone

The PID tuning of the drone is challenging for several reasons:

- The pitch, yaw, roll angle must be controlled
- The thrust must be controlled
- Instability may damage the system and lead to injuries

Usually, the drone uses a control technique used a cascade control, which corresponds to two PID loops: one inner loop, one outer loop (cf. Figure 4). The theory is usually covered in more advanced control system (graduate level). Just understand the principle of a cascade control: the inner loop is faster to react than the outer loop, hence improving the stability of the system.

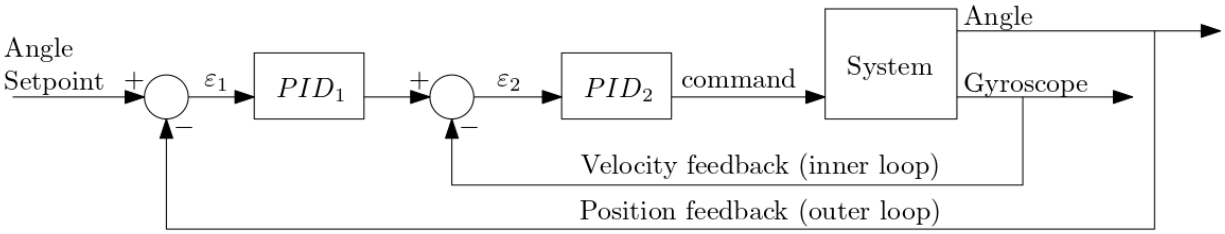


Figure 4: Cascade control for the angle orientation of a drone

In total, the number of PID controller that must be implemented is listed as follows:

- Two PID for the pitch
- Two PID for the roll
- Two PID for the yaw

So, a total of six PIDs. Usually, the operator can control the thrust of the system, but thrust is not within a feedback loop: the operator usually increase and decrease the thrust with a remote control.

Understand that for autonomous navigation, additional controllers must be implemented:

- Position control: the GPS module gives the latitude, longitude, and altitude of the drone. Thus, three additional PID should be implemented to navigate from the current position of the drone to the position desired by the operator.

However, this part is not included in the project.