DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Università
degli Studi
di Padova

Department of Information Engineering

Course of Modelling and Control of Electric Drives

A self balancing robot using ESP32 and MPU6050

Lecturer
Prof. Bianchi Nicola

Students
Bano Massimo, Sperandio Nicola

ID Numbers
2139251-2156437

Accademic Year 2025-2026

Abstract

This projects aims to design and implement a self-balancing robot using an ESP32 microcontroller and an IMU (Inertial Measurement Unit) sensor. The digital controller is based on a Proportional-Integral-Derivatived (PID) algorithm, which is tuned in order to achieve stability in the equilibrium position. The chassis of the robot is 3D printed, and the robot is powered by two DC motors. Throughout this work, the theoretical concepts behind the design of a self-balancing robot are explored, including the mathematical modeling of the system, the design of the PID controller, and the implementation details. At the end of this project, in the appendix section, the code for MATLAB/Simulink modelling and the microcontroller code provided.

Contents

Chapter 1

Introduction

We first start by introducing the concept of a self-balancing robot, which is a type of two-wheels robot that can maintain its balance while standing upright: this is achieved by continuisly measure the pitch angle of the robot and consequently adjust the motor's speed to keep the robot balanced. The main components of the self-balancing robot are reported in Table 1.1.

Component	Description
ESP32	Microcontroller
MPU6050	Inertial Measurement Unit (IMU) sensor
DFRobot DC 6V	DC Motors
DRV8871	Motor Driver
Molicel P42A 3.6V 45A	Series of two batteries for power supply.

Table 1.1: Main components of the self-balancing robot

1.1 ESP32 Microcontroller

The ESP32 (reported in Figure 1.1) is a powerful microcontroller developed by Espressif Systems, widely used in IoT applications due to its built-in Wi-Fi and Bluetooth capabilities. It features a dual-core processor, ample memory, and various peripherals, making it suitable for real-time control tasks required in self-balancing robots.

We chose the ESP32 for our self-balancing robot project because of its processing power but moreover for its higher clock speed (240 MHz) compared to other microcontrollers like Arduino Uno (16 MHz) or Arduino Mega (16 MHz). This allows faster control loop execution, which is crucial for maintaining balance in real-time.

The microcontroller can be programmed using the Arduino IDE but we chooose for another IDE, called PlatformIO, which offers more advanced features and better project management capabilities. This IDE can be integrated into Visual Studion Code and so we can take trace of all the changes with Git version control system.

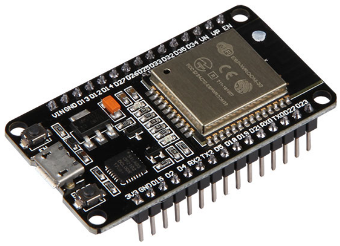

Figure 1.1: ESP32 development board

1.2 MPU6050

For the IMU sensor, we selected the MPU6050 (shown in Figure 1.2) which combines a 3-axis gyroscope and a 3-axis accelerometer. This sensor provides all the data through the I2C communication protocol. The MPU6050 range of measurement are reported in Table 1.2.

Sensor	Range of Measurement
Accelerometer	±2g, ±4g, ±8g, ±16g
Gyroscope	±250, ±500, ±1000, ±2000 °/s

Table 1.2: MPU6050 range of measurement

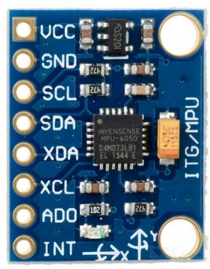

Figure 1.2: MPU6050 Inertial Measurement Unit (IMU) sensor

In our application we configure the accelerometer to a range of $\pm 4g$ for reasons that will be explained in Chapter ... and the gyroscope to a range of 250 $^{\circ}/s$ since we need high sensitivity and we don't expect higher angular velocities.

1.3 Motor Driver and DC Motors

1.3.1 DC motor

For the DC motor we selected the DFRobot DC 6V (shown in Figure 1.3), a motor that comes with a 120:1 gear ratio, providing high torque at low speeds, which is ideal for balancing applications. The motor is powered by a 6V power supply and can draw a stall current of up to 1.2A.

Figure 1.3: DFRobot DC 6V motor with 120:1 gear ratio

1.3.2 DRV8871

To control the DC motors, we use the DRV8871 motor driver (shown in Figure 1.4), which is capable of handling motor supply voltages from 6.5V to 45V. It's a double full-bridge driver, allowing for bidirectional control of the motors.

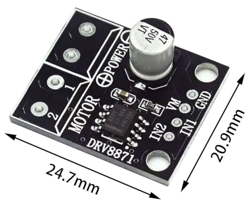

Figure 1.4: DRV8871 motor driver

It's capable of delivering up to 3.6A continuous current per channel, which is more than sufficient for our DC motors.

Chapter 2

Modelling and simulation

Now that we described the hardware components of our self-balancing robot in Chapter 1, we can proceed to model the system and simulate it's behavior in MATLAB/Simulink environment.

2.1 Physical model

The best way to model a self-balancing robot is to treat it as a *inverse pendulum*, like the one reported in Figure 2.1

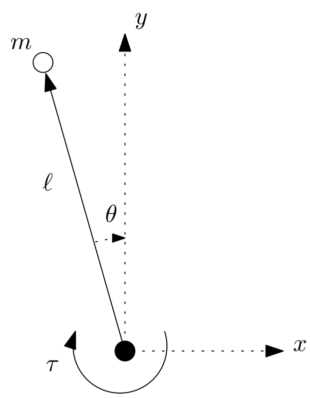

Figure 2.1: Inverse pendulum schematic representation

Where:

- m is the mass of the pendulum
- ℓ is the length of the pendulum
- θ is the angle of the pendulum with respect to the vertical axis
- τ is the torque applied to the base of the pendulum

The equations that govern the motion of the inverse pendulum can be derived using Newton's laws.

$$m\ell^2\ddot{\theta}(t) = mg\ell sin(\theta) + \tau(t) \tag{2.1}$$

Where g is the acceleration due to gravity and $\ddot{\theta}(t)$ is the angular acceleration of the pendulum. To linearize the equation 2.1, we can use the small angle approximation, which states that for small angles (in radians), $sin(\theta) \approx \theta$. This leads to the following linearized equation:

$$m\ell^2\ddot{\theta}(t) = mg\ell\theta + \tau(t) \tag{2.2}$$

From equation 2.2, we can derive the transfer function of the system by taking the Laplace transform, assuming zero initial conditions:

$$G_{pendulum}(s) = \frac{\theta(s)}{\tau(s)} = \frac{\frac{1}{m\ell^2}}{s^2 - \frac{g}{\ell}} \tag{2.3}$$

In equation 2.3 we can see that the system has two poles at $s = \pm\sqrt{\frac{g}{\ell}}$, indicating that the system is unstable, as one of the poles is in the right half of the s-plane.

2.2 DC motor modelling

To model the DC motors used in our project, we can use the following equations that describe the electrical and mechanical dynamics of a DC motor:

$$v(t) = L\frac{di(t)}{dt} + Ri(t) + K_\phi\omega(t) \tag{2.4}$$

By applying the Laplace transform to equation 2.4, and considering also the gear ratio $K_G = 120$ we obtain:

$$G_{motor}(s) = \frac{\tau(s)}{v(s)} = k_G \cdot \frac{K_\phi}{R + sL} \tag{2.5}$$

The obtained transfer function is a first order system.

2.3 Complete model

The complete model is the product of the two transfer functions obtained: we have to consider that we do not consider the *back EMF* of the motor in the transfer function: this is a simplification of the model. In Figure **??** the complete block diagram is reported, where $G(s) = G_{pendulum}(s) \cdot G_{motor}(s) = \frac{\theta(s)}{v(s)}$.

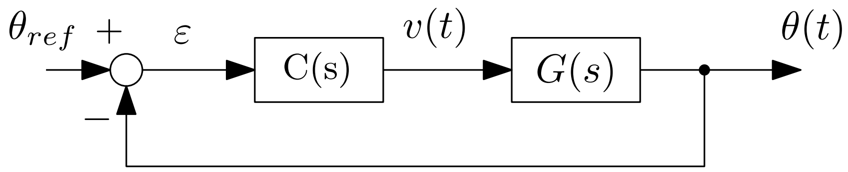

Figure 2.2: Complete block diagram of the self-balancing robot model