DRISHTI
A Revolutionary Concept

Makernova 3.0

Self Balancing Bot

August 2025

Team Members

U24EC022 Om Mehta
U24EE029 Ayush Rana
U24EC098 Prabal Pratap Singh
I24PH011 Prashant Kumar
U24EC106 Rupreet R
U24EC111 Mahin Prajapati
U24EE074 Palash Bhatt
U24EV047 Sanjana Dadapuram
U24EV045 Chandana Dharamsoth
U24EC144 Thanuja Anumula

Mentors

Param Pandya
Devaam Dalal
Harshit Gupta

Problem Statement:

Self-Balancing Bot: Intro to the PID algorithm and other basic sensors for the bot and use of IMU for the bot balancing.

Acknowledgement

Firstly, we are deeply grateful to our academic mentors, for their invaluable guidance, constructive feedback and continuous encouragement throughout this project. Their expertise and insights have been instrumental in shaping our understanding of how the bot will work. Our appreciation also goes to the Drishti Lab of SVNIT for providing us with the resources, facilities and support needed to carry out this project. Finally, we would like to acknowledge the authors and researchers whose work has laid the foundation for our project. Thank you all for your support and encouragememnt.

Abstract

This document presents the development process of a **two-wheeled coaxial self-balancing robot** created during *Makernova 3.0* of the *Drishti* Club. The robot employs an **Inertial Measurement Unit (IMU)** to obtain orientation data, which is processed through a **Proportional–Integral– Derivative (PID)** control algorithm. The algorithm determines the corrective motor actions required to maintain balance, ensuring that the robot remains upright. The documentation outlines the design, control approach, and implementation of the system.

Contents

List of Figures

0.1 Understanding the Problem

The self-balancing robot has been enormously recognized which is based on electronic device and embedded control and being used as a human transporter in many areas. The self-balancing BOT is based on the Inverted Pendulum model (IP), i.e. is a pendulum that has its center of mass above its pivot point. In order to balance at two-wheeled inverted pendulum robot it is necessary to have accurate information of the live tilt angle from using a measurement on it. Furthermore, a controller needs to be implemented to compensate for said tilt. The system is non-linear and unstable with one input signal and several output signals, i.e. the input signal is the tilt of the bot while the output signal being voltage signals for motor driver. It is virtually impossible to balance the pendulum in the inverted position without applying some external force to the system, hence, a PID-controller can be incorporated to control the pendulum angle.
To balance the bot, the bot will run in the direction in which the pendulum falls. The more the angle pendulum is displaced, the faster the bot will be.
To obtain the angle of tilt of the pendulum, an Inertial Measurement Unit (IMU) sensor will be used. This angle data might be distorted. To remove this distortion, its data will be passed through a filter which will generate clean usable data. Now this data will be given to the micro controller of the system, which will have the PID algorithm coded into it, which will use this angle value to generate signals for the motor driver, which will drive the motor accordingly to balance the bot.

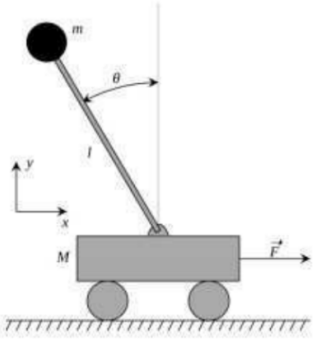

Figure 1: Inverted Pendulum

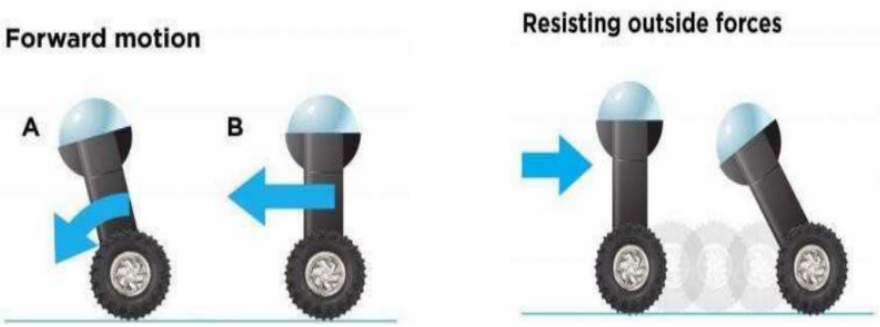

Figure 2: Balancing Technique of Bot

0.2 Components

Component Name	Basic Specifications	Purpose	Quantity
ESP 32	Type: Xtensa dual-core 32-bit LX6 microprocessor. Clock Speed: 160 or 240 MHz. Performance: Up to 600 DMIPS. 2.4 GHz Wifi v4.2 Bluetooth	Microcontroller	1
MPU 6050	Type: 6-axis (3-axis accelerometer, 3-axis gyroscope) MEMS sensor.	Inertial Measurement Unit	1
Rhino Smart Dual DC 20A	Motor Channels: 2 Operating Voltage: 6V to 30V DC Continuous Current: 20A per channel Peak Current: 60A (for 10 seconds) Motor Output PWM Frequency: 20 kHz PWM and DIR Inputs: Yes H-Bridge Design: Full discrete NMOS	Motor Driver	1
DC geared Motors	Rated Voltage: 12VDC No load current (mA) : ¡ 100 No load speed (r.p.m) : 75 ± 7.5 Rated load torque (kgf.cm) : 3 Rated current (mA) : ¡ 600 Rated load speed (r.p.m) : 50 ± 5 Gear Ratio: 60:1	Motors	2
Optical Wheel Encoders	-	Wheel Encoders	2
Chassis	Planes (3 piece) - Length - 25cm Width - 15cm Poles - 30 cm (4 piece) Motor Clamps x2	-	1
Battery	LiPo 3S (11.1V)	-	1
Buck Converter	Input range of 4.75-35V Output range of 1.25-26V 2A rated current (3A max) 150kHz switching frequency	Voltage Regulator	1

0.3 Block Diagram of the Bot

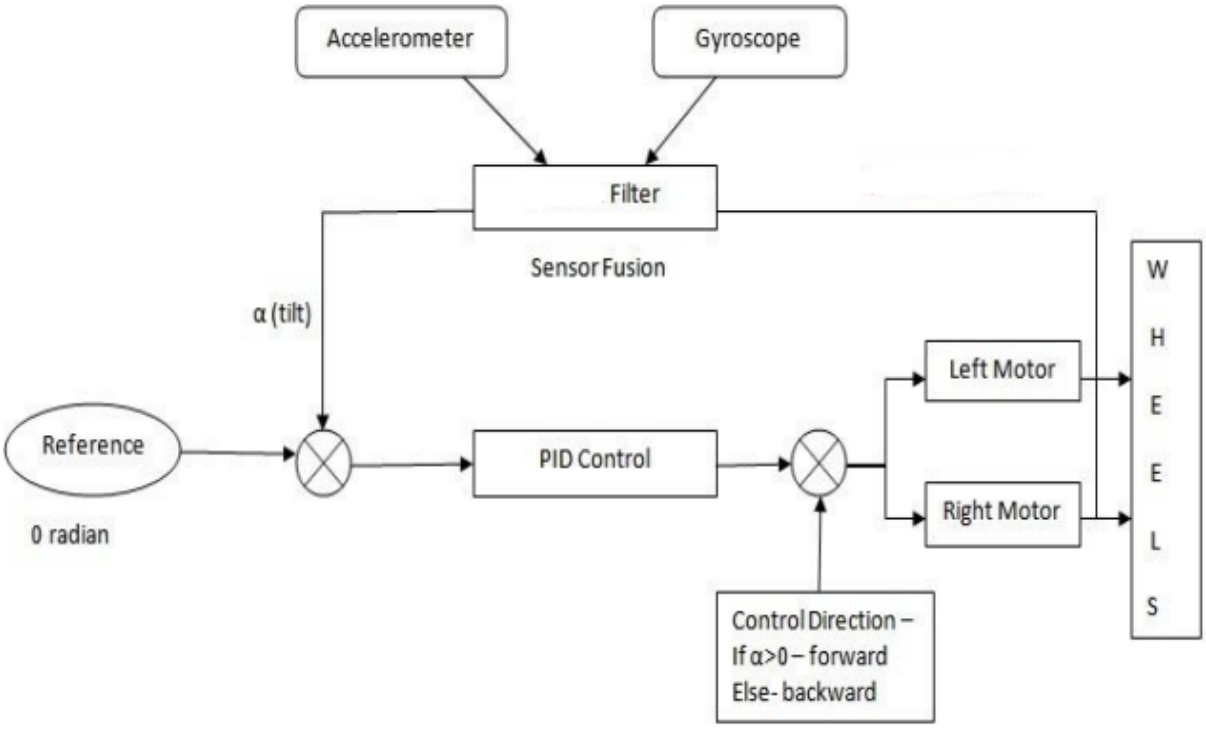

Figure 3: Block Diagram of Hardware

Description:
The block diagram represents the working of a self-balancing robot. The accelerometer and gyroscope sensors measure the tilt angle (α), which is processed using a filter through sensor fusion to obtain accurate orientation data. This tilt is compared with the reference position (0 radians), and the error is given to the PID controller. The PID controller generates control signals to drive the left and right motors, adjusting their speed and direction. Based on the tilt ($\alpha < 0$ or $\alpha > 0$), the control direction is set for forward or backward motion, enabling the wheels to maintain balance.

0.4 Circuit Diagram

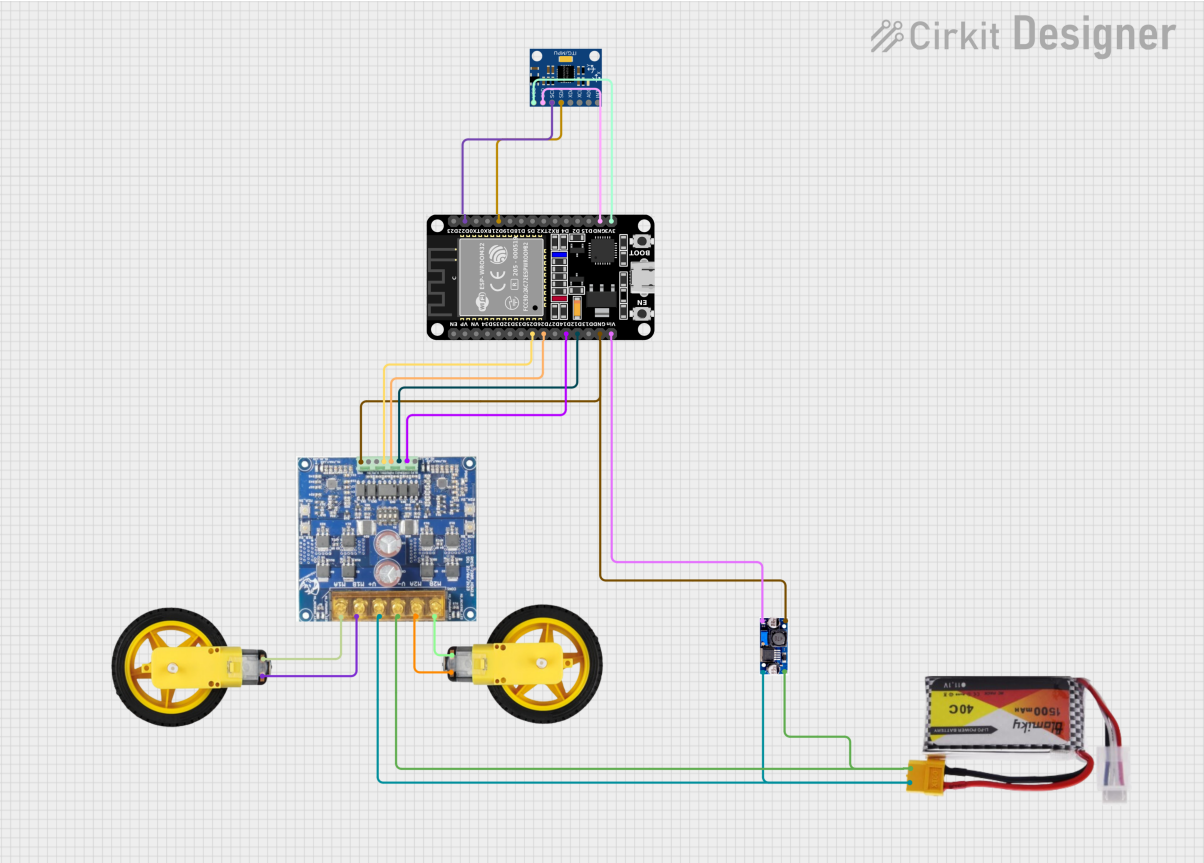

Figure 4: Circuit Diagram of Hardware

Description:
The circuit is divided into two parts: the *power circuit* and the *main circuit*. The power circuit, as the name suggests, supplies the appropriate voltage to the components of the main circuit. It consists of a battery and a buck converter. The battery ideally provides 12 V, which is stepped down to 5 V by the buck converter.
The main circuit includes the ESP32, MPU6050, RCMS 2305, and DC geared motors. The V_{in} and GND pins of the ESP32 are connected to the buck converter to receive 5 V. The SDA and SCL pins of the MPU6050 are connected to pins D21 and D22 of the ESP32 to enable I2C communication for angle data. The PWM and DIR pins of the Rhino motor driver are connected to pins D12, D14, D26, and D25 of the ESP32 to receive control signals. Finally, the Rhino motor driver is connected to the motors, completing the circuit.

1 Hardware Implementation

1.1 Micro-controller

1.1.1 Introduction

A microcontroller (MCU) is a small computer on a single chip that contains a processor, memory, and input/output ports. It is designed to perform specific control tasks in electronic devices, such as reading sensors, processing data, and controlling outputs like motors or displays. They are programmed using languages like C, C++, or assembly.

1.1.2 ESP32

The ESP32 is a low-cost, low-power microcontroller with Wi-Fi and Bluetooth built in. It's widely used for robotics, smart devices, and automation projects.

Figure 1.1: ESP32

• Dual-core Xtensa LX6 processor (up to 240 MHz) – fast and efficient
• Integrated Wi-Fi (802.11 b/g/n) – for IoT & wireless connectivity
• Bluetooth 4.2 & Bluetooth Low Energy (BLE) – supports both classic and BLE
• High memory: 520 KB SRAM
• High-Resolution ADC & DAC
• Read sensors like MPU6050, BNO055, ultrasonic sensors, etc., with higher precision

ESP32 Advanced Peripheral Interfaces

• 34 programmable GPIOs
• 12-bit SAR ADC up to 18 channels
• Two 8-bit DAC
• 10 touch sensors
• Four SPI interfaces
• Two I2S interfaces
• Three UART interfaces
• RMT (TX/RX)
• 25 PWM pins to control things like motor speed or LED brightness
• Two 8-bit DACs to generate true analog voltages

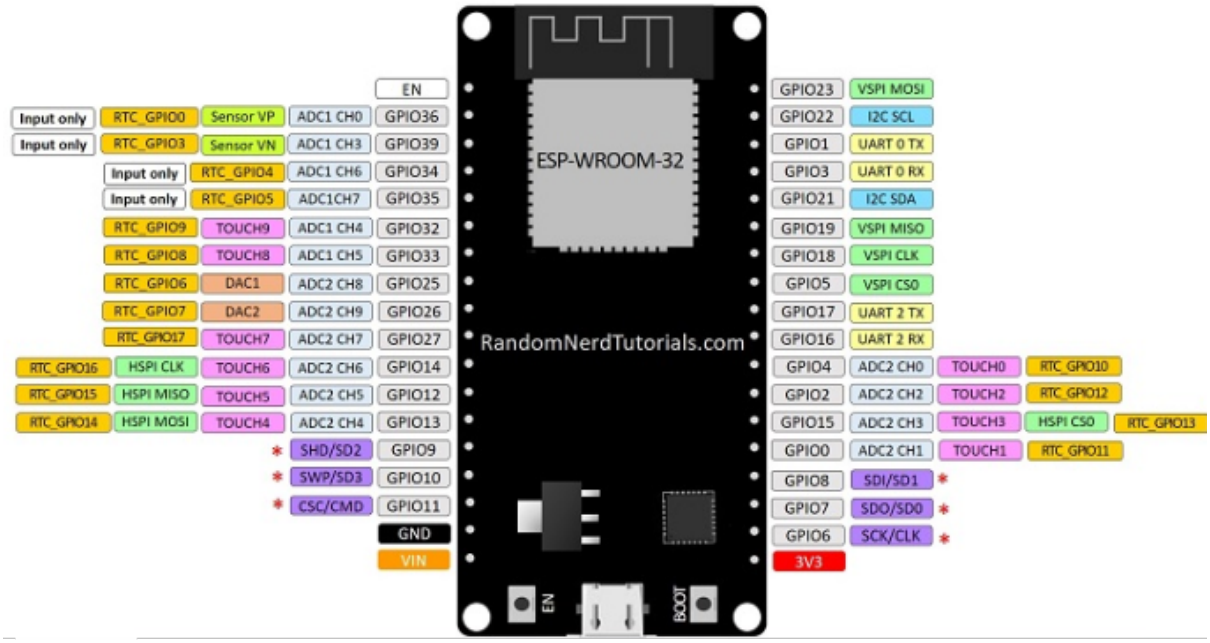

Figure 1.2: Pindiagram of ESP32

1.2 Inertial Measurement Unit

1.2.1 Introduction

An IMU (Inertial Measurement Unit) is a device that measures the motion and orientation of an object using sensors such as an accelerometer, gyroscope, and sometimes magnetometer. In a self-balancing bot, the IMU is the core sensor that detects the tilt angle of the robot so that the controller (PID) can take corrective actions to maintain balance.

1.2.2 MPU6050

Introduction

The MPU6050 is a 6-DOF sensor that integrates:
• 3-axis Accelerometer → Measures linear acceleration and tilt.
• 3-axis Gyroscope → Measures angular velocity.

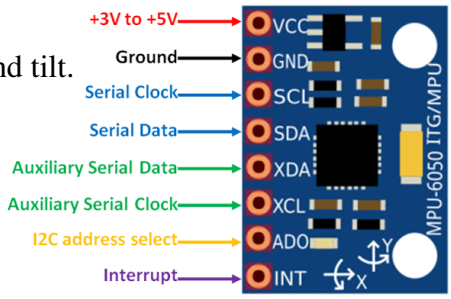

Figure 1.3: MPU6050

Key Features

• Communicates via I^2C.
• High sampling rate (up to 1 kHz).
• Low cost and widely available.
• Requires sensor fusion algorithms (Kalman or Complementary Filter) in the microcontroller to obtain a stable tilt angle.

Implementation

To implement the MPU6050 in our bot, the SDA and SCL pins of the sensor were connected to the ESP32 to establish data communication. The *MPU6050 light* was utilized to simplify sensor integration, while the Wire library was included to enable the I2C communication protocol on the ESP32. A variable of type byte was declared and assigned the return value of mpu.begin(). This status value is used to verify whether the MPU6050 is functioning correctly, allowing the program to abort execution if initialization fails. The function mpu.calcOffsets() was then called to calculate the sensor offsets. This calibration ensures that the current orientation of the sensor is taken as the reference position, corresponding to 0 degree. Within the void loop() function, mpu.update() was first used to refresh the MPU6050 data in every iteration. The function mpu.getAngleY() was then employed to retrieve the required angle measurement.

1.3 Motor Driver

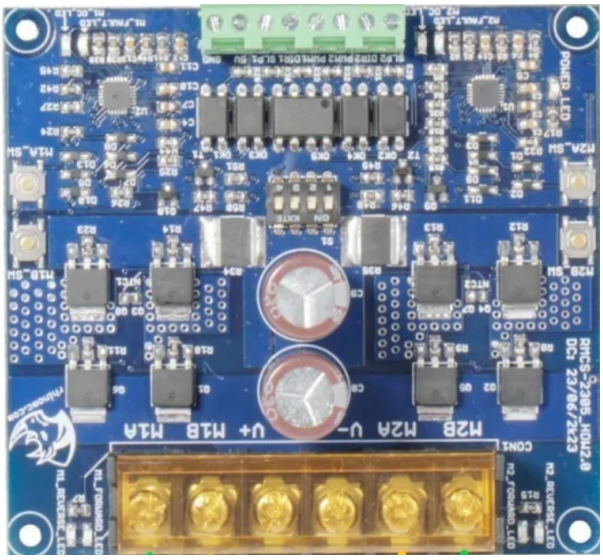

Figure 1.4: RCMS 2305

1.3.1 Introduction

The Rhino Smart Dual DC Motor Driver 20 Amp is designed for controlling two high-power brushed DC motors.

1.3.2 Specifications

Motor Channels: 2
Operating Voltage: 6V to 30V DC
Continuous Current: 20A per channel
Peak Current: 60A (for 10 seconds)
Motor Output PWM Frequency: 20 kHz
Logic Level Compatibility: 1.8V, 3.3V, 5V, and 12V logic inputs
PWM and DIR Inputs: Yes

H-Bridge Design: Full discrete NMOS
Dimensions: 88.90mm x 78.74mm
Reverse Polarity Protection: No.

1.3.3 Functional Overview

A Rhino 20A motor driver provides bidirectional control for one or two high-power brushed DC motors (depending on single or dual-channel models) with continuous operation at 20 amps between 6V and 30V. It uses a PWM and DIR (Direction) input for control, compatible with controllers like esp32 and Arduino UNO. Key functional features include built-in over-current and temperature protection, along with onboard test buttons and LEDs, for convenient testing and robust operation.

1.3.4 Features and Capabilities

Overcurrent Protection: Prevents damage when the motor stalls or draws excessive current. The maximum current limit adjusts based on board temperature (higher temperature = lower limit).
Temperature Protection: Shuts down the driver to prevent overheating.
Undervoltage Shutdown: Ensures safe operation under low voltage conditions.
Onboard Test Buttons and LEDs: Allows for quick functional testing without a host controller.
Compatibility: Works with a variety of host controllers. The driver uses PWM and DIR inputs to control the speed and direction of the motors. The PWM input controls the motor speed, and the DIR input controls the direction.

1.3.5 Implementation

To implement motor driver in our bot, we connect the required pins to ESP 32 and motors. Then we simply set the connected PWM pin value of the ESP 32 to an integer from 0 to 255 (for our case, it will be the result of the PID expression implemented) and connected DIR pin value of the ESP 32 to HIGH or LOW according to our need.

1.4 Motors

1.4.1 Requirements of Motor

1. Precise control needed
2. High Torque at Low Speed
3. Fast Response
4. Compact & Simple Drive
5. Encoders for feedback

1.4.2 Geared DC Motors

Figure 1.5: Geared DC Motor

Introduction:

DC gear motor is a type of electric motor that combines a standard DC motor with a gearbox. This integration allows for increased torque output and reduced rational speed, which makes it suitable for applications that require high torque and precise output.

Advantages:

1. High Torque at Low Speed
2. Simple Speed Control
3. Fast Response
4. Compact & Lightweight
5. Cost-Effective
6. Easy to Interface
7. Availability in Wide Range
8. Encoders Available
9. Continuous Rotation

Disadvantages:

1. Lower Efficiency
2. Bulky for High Torque Applications
3. Noise and Vibration
4. Limited Precision (without encoder)
5. Speed-Torque Tradeoff
6. Overheating Under High Load

1.5 Wheel Encoders

1.5.1 Introduction

A wheel encoder is an electromechanical sensor that measures rotational or linear movement by converting it into digital or analog electrical signals, enabling precise tracking of position, speed, and direction in applications like robotics, industrial automation, and vehicles. These sensors work by detecting changes in a coded pattern on a rotating disk or by measuring the rotation of a wheel that rides on a surface, using optical or magnetic principles.

1.5.2 Magnetic Encoder

Working Principle: A magnet rotates with the mechanical component, altering the magnetic field detected by a sensor, such as a Hall effect or magnetoresistive sensor. The sensor converts these magnetic field variations into electrical signals representing position.

Components:
A rotating magnetic element (a wheel or magnet) and a magnetic sensor (Hall effect or magnetoresistive).

Applications:
Automotive systems, heavy machinery, wind turbines, and any industrial application where reliability is crucial.

Advantages:
Robust in harsh conditions; unaffected by dust, oil, moisture, and extreme temperatures.

Disadvantages:
Typically has lower resolution and accuracy compared to optical encoders, though this is improving.

2 Algorithms and Protocols

2.1 Filters

2.1.1 Complementary Filter

Filter is a sensor fusion technique that combines data from two or more sensors with complementary characteristics to produce a more accurate and reliable estimate of a system's state. It's particularly useful when dealing with sensors that have different strengths and weaknesses, such as gyroscopes and accelerometers. The term "complementary" refers to the fact that the sensors used in this technique have characteristics that offset each other's weaknesses. In many applications, complementary filters are used to estimate the orientation (e.g., roll, pitch) of a device using data from a gyroscope and an accelerometer. The gyroscope data is passed through a high-pass filter to remove low-frequency drift. The accelerometer data is passed through a low-pass filter to remove high-frequency noise.
Combining: The filtered gyroscope and accelerometer data are then combined (typically using a weighted sum) to produce a more accurate estimate of the orientation.

Advantages:

Computational Efficiency: Complementary filters are generally computationally simpler than other sensor fusion techniques like Kalman filters, making them suitable for resource-constrained devices.
Good Performance: They can provide accurate and stable estimates of system state in many applications, especially when dealing with relatively slow-changing dynamics.

Disadvantages:

Tuning: Complementary filters require careful tuning of the filter parameters (e.g., cutoff frequencies) to optimize performance for a specific application and sensor setup.
Limited to Certain Applications: They are best suited for situations where the sensor dynamics are relatively slow and predictable.
The mathematical equation for a complementary filter can be expressed as:

$$angle = \alpha(angle + gyro * dt) + (1 - \alpha) * accel \tag{2.1}$$

Here's a breakdown of the equation:
alpha α : This is the complementary filter coefficient, a value between 0 and 1, which determines the weighting of the gyroscope and accelerometer data.

Explanation:

The equation works by taking the previous angle estimate, adding the change in angle calculated from the gyroscope (gyro * dt), and then blending this with the angle calculated from the accelerometer (accel). The alpha value determines how much weight is given to each sensor's contribution.
Gyroscope (High Frequency):
The gyroscope provides a fast response to changes in orientation, but it is prone to drift over time due to integration errors.
Accelerometer (Low Frequency):
The accelerometer is less susceptible to drift but is more affected by noise and disturbances and is only accurate at low frequencies.
By using a complementary filter, the high-frequency components from the gyroscope are combined with the low-frequency components from the accelerometer, resulting in a more stable and accurate orientation estimate. The alpha value is typically chosen to favor the gyroscope for short-term changes and the accelerometer for long-term stability.

2.2 PID Algorithm

2.2.1 Introduction

The **Proportional-Integral-Derivative (PID)** algorithm is a widely used control loop feedback mechanism in industrial control systems. It calculates an "error value" as the difference between a desired *setpoint (SP)* and a measured *process variable (PV)*, and then applies a correction based on three distinct terms:
Proportional (P) Term: This term generates an output proportional to the current error. A larger error results in a larger proportional response, providing an immediate corrective action.
Integral (I) Term: This term addresses the accumulation of past errors. It integrates the error over time, which helps to eliminate any steady-state error or offset that the proportional term alone cannot correct.
Derivative (D) Term: This term anticipates future errors by considering the rate of change of the error. It provides a damping effect, which helps to reduce overshoot and oscillations in the system, leading to a more stable and faster response.

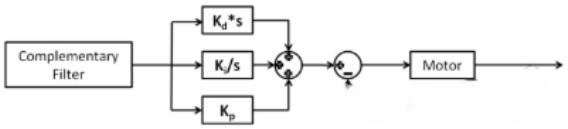

Figure 2.1: PID Algorithm

$$PIDCorrectionFactor = K_p * E + K_i * \int E(t)\,dt + K_d * \frac{dE(t)}{dt} \tag{2.2}$$

In this bot, our desired setpoint is 0 rad angle of the bot. The result of the PID expression will become the speed of motor. When the bot tilts forward, the difference between the desired setpoint and the actual angle will increase and will be positive. Hence, the proportional term will quickly increase which will increase motor's speed and will balance the bot. Now, the bot will start oscillating as proportion factor would not be enough due to inertia of the bot. To stop

this oscillation, we have differentiation factor. This will deal with the overshooting of the bot. After this, bot might later deviate from the correct angle. This maybe due to IMU bias, motor mismatch, friction or battery sag. To solve this issue, we add integration term. It integrates the error accumulated overtime and gives signals to solve it.

2.2.2 Implementation

To integrate this algorithm in our microcontroller, we would have to know the very basic idea of Integration and Differentiation.

dt

First off, we would need *dt* for both integration and differentiation, for that, we start one time variable in void setup (prev_time) using micros() and one in void loop (curr_time) again using micros(). When the loop will start, we will subtract these both variables to get our dt for that loop and will assign the value of the curr_time to prev_time for proper functioning of the code.

Integration

Integration would have to return the error accumulated to resolve the eventual drift. To do that, we will make two variables named integral and error and initialize it value to be 0.

$$TrapezoidalRule : I(t) = I(t-\Delta t) + \frac{f(t)+f(t-\Delta t)}{2} \cdot \Delta t \tag{2.3}$$

Now, one issue can occur which is that our bot will initially take some time to get upright from the ground, this accumulated error can off-balance the bot. Hence, to solve this, we put a constraint on the integration value of the integral variable which can be found through trial and error.

Differentiation

For differentiation, we use the formula

$$\frac{df}{dt} = \frac{f(t)-f(t-\Delta t)}{\Delta t} \tag{2.4}$$

So, we use the error variable we created for integral and subtract our current error signal with it and divide it with dt to get our differentiation value. And later we assign new values to error and time as usual.

Overall

We got the differentiation as well as the integration of the error values. Hence, we finally use the below formula to generate the speed of motor.

$$output = Kp * error + Ki * integral + Kd * derivative \tag{2.5}$$

This output might not be an integer but we need our value to be an integer between -255 to 255. So, we truncate the value to be an integer and then put constraint on it.

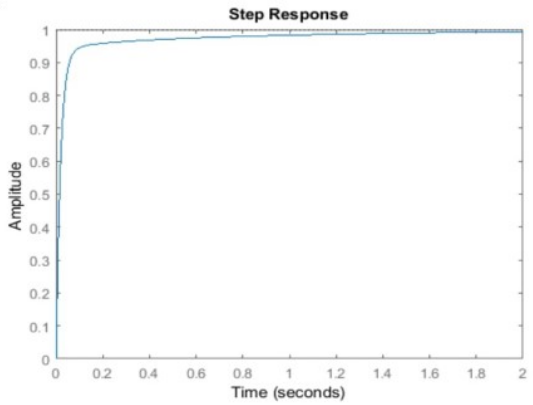

Figure 2.2: PID graph

Now, if the value of this output is positive, we need to make the bot move forward and if the output is negative, we need to make the bot move backwards. To acheive this, we use a simple if-else statement to give proper signal to our motor driver.
After tuning the bot as best as we can, we found that for our bot, value of K_p is 200 and of K_d is 60 while K_i is not used currently.

2.3 Communication Protcol

2.3.1 Introduction

A communication protocol is a set of rules that governs how data is transmitted and received between two or more devices, ensuring they can understand each other. It defines aspects like data format, transmission method, error handling, and synchronization. Essentially, it's the language that devices use to communicate, enabling them to exchange information effectively.

2.3.2 I²C (Inter-Integrated Circuit)

Overview

I²C is a synchronous, multi-master, multi-slave protocol. It is widely used for communication between a microcontroller and multiple peripheral devices, such as sensors and EEPROMs, using only two wires.

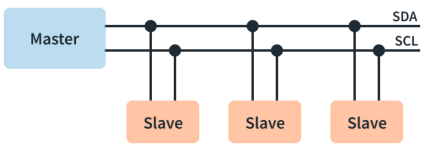

Figure 2.3: I2C Protocol

Features

Two-wire interface: SDA (Serial Data) and SCL (Serial Clock).
Multi-master, multi-slave support with a unique 7-bit address for each device (128 possible devices).
Low bandwidth, short range and speeds of 400kbps.

Communication Flow

1.The master initiates communication with a START bit.
2.An Address bit is sent for the slave.
3.Master sends a Read or Write bit (0 for write, 1 for read).
4.The addressed slave acknowledges (ACK) or does not acknowledge (NACK).
5.Data is transferred byte by byte, with each byte being acknowledged.
6.The master sends a STOP bit to end the communication.

Advantages

Requires only 2 wires regardless of the number of devices.
Supports both multiple masters and slaves.
Excellent for short-distance communication on a PCB.

Limitations

Limited bus length (typically a few meters).
The protocol is more complex than UART, and requires pull-up resistors on the SDA and SCL lines.

Chassis

The chassis of the self-balancing robot has been designed and fabricated with a focus on strength, balance, and ease of component placement. The base of the structure measures 25 cm in length and 15 cm in width, providing a compact yet stable platform for the robot. To support the structure vertically, four poles of 30 cm height have been mounted at the corners, ensuring rigidity and forming a strong frame for holding additional planes and components.
The design incorporates a total of three planes: two planes at the top and bottom ends, and one at the middle. The end planes provide mechanical strength and structural stability, while the middle plane serves as a mounting platform for electronics such as the microcontroller (ESP32), motor driver, and sensor modules (MPU6050). This arrangement also helps in proper weight distribution, which is crucial for maintaining balance during operation.

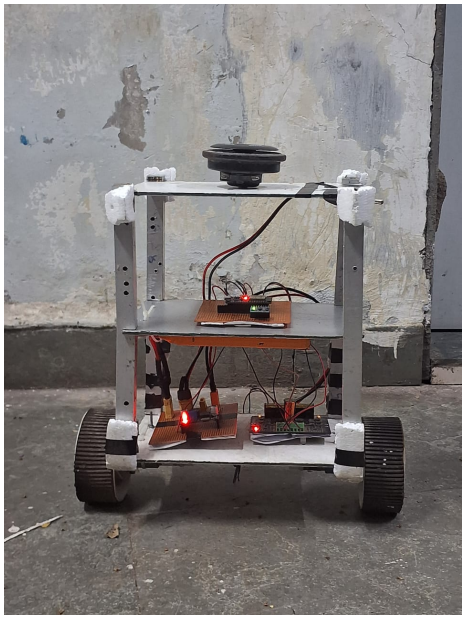

Figure 2.4: Chassis

Errors and Solutions

2.3.3 Motor Fixing Issue

Error:

The motors of the bot were not fixing properly. Screw tend to loosen up and tyres eventually fall off.

Cause:

Screw and wheel's diameter were not matching which leads to tyre falling off.

Solution:

Bontyl was used to stick the wheels on staff properly.

2.3.4 Power turning off randomly

Cause:

Solder of XT60 got loose.

Solution:

Resoldering of the XT60 in proper manner.

2.3.5 Robot Turning in Rounds

Error:

Initially robot starts turning in round continuously.

Cause:

It is unclear at the moment. It maybe due to sensor initialization problem.

Solution:

It has been observed that this error generally gets resolved sometime after turning on the bot. Currently working on it to remove this error.

2.3.6 Motor Imbalance

Error:

Robot drifts to one side while balancing.

Cause:

Unequal torque in motors

Solution:

Calibrate motors or introduce motor correction factors in the control algorithms.

Future Goals:

The self-balancing bot in its current stage demonstrates the fundamental principle of dynamic stability and real-time control using sensors and actuators. However, for practical applications and to enhance its robustness, the following future developments are envisioned:

2.3.7 Controlling using Dabble

The bot currently can only balance itself on one place and does not move as we desire. To achieve this, we can connect ESP 32 to our mobile device and control it using dabble software.

2.3.8 Balancing on Inclined Surfaces

The present design ensures stable operation primarily on flat ground. A significant future goal is to extend this capability to inclined planes and slopes. This requires modifications in the control algorithm to compensate for gravitational components acting along the slope.

2.3.9 Rough Surface Balancing

Real-world applications demand stability not only on smooth floors but also on uneven and rough terrains. Enhancements in the mechanical design such as suspension systems, larger wheels, or shock absorbers will be required. Control systems can be upgraded with sensor fusion techniques (IMU + gyroscope + accelerometer) to handle sudden disturbances. This development will allow the bot to function reliably in outdoor conditions and industrial environments.

Bibliography

2.3.10 Research Paper

- Development of Self Balancing Robot with PID Control (2017) – Shubhank Sondhia, Ranjith Pillai R., Sharat S. Hegde, Sagar Chakole, Vatsal Vora

2.3.11 Micro-controller

- Arduino UNO – Official

2.3.12 Inertial Measurement Unit

- BNO055 – Adafruit
- MPU6050 – Adafruit
- MPU6050 – DFRobot
- MPU Interfacing - Robu.in

2.3.13 Motor

- DC Motor – Pololu

2.3.14 Communication Protocols:

- I^2C – TI

2.3.15 Motor Driver:

- L298N – Datasheet
- Using Motor Driver – YouTube
- RMCS2310 – RoboKits

2.3.16 Control System:

- PID