SELF BALANCING ROBOT

A Design Lab Project Report Submitted
in Fulfillment of the Requirements
for the Course of

EE396: Design Laboratory

by

Bharti Rajora (210102024)
Kishan Kumar (210102047)

under the guidance of

Dr. Chayan Bhawal

to the

DEPARTMENT OF ELECTRONICS AND ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI
GUWAHATI - 781039, ASSAM

CERTIFICATE

*This is to certify that the work contained in this thesis entitled "**Self Balancing Robot**" is a bonafide work of **Bharti Rajora** (**Roll No. 210102024**) and **Kishan Kumar** (**Roll No. 210102047**), carried out in the Department of Electronics and Electrical Engineering, Indian Institute of Technology Guwahati under my supervision and it has not been submitted elsewhere for a degree.*

Supervisor: **Dr. Chayan Bhawal**

Assistant Professor,

May, 2024 Department of Electronics and Electrical & Engineering,

Guwahati. Indian Institute of Technology Guwahati, Assam.

Acknowledgements

We would like to express our sincere gratitude to Dr. Chayan Bhawal for his invaluable guidance, support, and mentorship throughout the completion of the Self Balancing Robot project as part of the EE396 Design Lab course. Dr. Bhawal's expertise, encouragement, and insightful feedback have been instrumental in shaping the development of this project from its inception to its fruition. His unwavering dedication to fostering innovation and excellence in engineering education has inspired me and enriched my learning experience immeasurably.

Abstract

Self-balancing robots have garnered attention for their impressive stability on two wheels. This report outlines the design and analysis of a self-balancing robot developed using an L293D motor driver, an Arduino Uno, and an MPU6050. By leveraging these components, the robot can measure its tilt angle and adjust motor speeds via a Proportional-Integral-Derivative (PID) control algorithm. The mechanical and control aspects of the robot's completion are discussed thoroughly. Test results evaluating the robot's balancing performance are provided, along with suggestions for future design enhancements. This project presents an exciting endeavor in crafting self-balancing robots using easily obtainable components. Moreover, it sheds light on the critical role that control systems play in the realm of robotics, demonstrating their indispensable significance in real-world applications.

Contents

Chapter 1

Introduction

The introductory chapter describes the Problem Statement and the Objective of the project: Self Balancing Robot conducted within the context of a Bachelor's Degree in the Department of Electronics and Communication Engineering at IIT Guwahati.

The concept of an inverted pendulum serves as a foundational principle for understanding the dynamics of self-balancing systems. Inverted pendulum applications are plenty; for example, the human body is an inverted pendulum balancing the upper body around our ankle joints in every step. The inverted pendulum system, as described, represents an unstable equilibrium where a vertical pendulum needs to be maintained upright by continuously adjusting its base.

A real-world example of self-balancing technology is evident in the Segway Personal Transporter. Operating on a similar principle to the discussed robot, the Segway serves as an eco-friendly commuting option in urban areas and tourist destinations. Its self-balancing capability ensures rider stability across various terrains and inclines. Beyond personal use, the Segway has found applications in sectors such as security, law enforcement, and hospitality, facilitating efficient transportation over large areas.

1.1 Problem Statement

The problem statement is to design, construct, and develop the physical representation of a Self-Balancing Robot along with the program and analysis of the Control Module.

The construction and development of a self-balancing robot involves planning, precise design, and execution. In order to construct a physical representation of an inverted pendulum, a two-wheeled self-balancing robot will be designed and programmed –which includes choosing sufficient hardware components such as motors, wheels, sensors (accelerometers and gyroscopes), microcontrollers (Arduino or Raspberry Pi), batteries and other electronic and hardware components, etc. The Programming and Analysis include implementing functions for reading sensor data, calculating motor control signals, and maintaining balance.

1.2 Objective

The primary objectives of developing a self-balancing bot are to achieve and maintain a stable upright position by continuously adjusting the cart's position based on real-time sensor feedback. This involves integrating sensors like MPU6050 to accurately measure the pendulum's angle θ, angular velocity $\dot{\theta}$, and Body's position x, ensuring the bot responds quickly to changes in its environment or disturbances to maintain balance. Energy efficiency is a key consideration in the design, aiming to operate the control system and actuation mechanisms efficiently while conserving energy. The developed system is designed to be robust, capable of handling uncertainties, external disturbances, and variations in operating conditions while maintaining stability. Safety features are implemented to prevent potential hazards and ensure the bot's safe operation in various environments. Furthermore, the design emphasizes scalability and adaptability, with modular components and algorithms that can be easily adapted or scaled for different applications or environments.

Chapter 2

Theory

2.1 Model Definition

The Balancing robot's physical challenge is often represented by the extensively studied inverted pendulum. Typically, it's depicted as a rigid rod attached to a frictionless pivot point on a cart that moves along a single direction. This model assumes that the wheelbase acts like a cart sliding effortlessly on a frictionless surface. This description draws inspiration from the MathWorks tutorial on inverted pendulum. For simplicity, the pendulum's movement is constrained to a single direction, with the angle changing within the xy-plane.

2.2 Control Theory

A PID controller, often referred to as a three-term controller, is a feedback control system extensively used in various industrial applications and systems requiring continuous adjustment. This controller calculates an error signal, e(t), by comparing a desired setpoint (SP) with a measured process variable (PV). Based on this error, the PID controller applies corrective actions through its proportional (P), integral (I), and derivative (D) terms. The PID system offers precise and prompt adjustments to maintain desired control. A common example is cruise control in vehicles. When driving uphill, without PID, the speed might

drop due to constant engine power. However, the PID algorithm quickly adjusts the engine power to maintain the desired speed, minimizing delays and overshooting. These settings are fine-tuned to find the best values for the robot.

2.3 Components Required

2.3.1 L293D Motor Driving Module

An L293D motor driving module is capable of driving 2 DC motors with directional and speed control. We are required to run two DC motors with speed control and direction, therefore, one L293D Motor Driving Module is sufficient for the robot.

2.3.2 MPU6050

MPU6050 sensor module has a 3-axis Gyroscope, a 3-axis Accelerometer, and a Digital Motion Processor. The 3-axis Gyroscope is used to detect rotational velocity along the X, Y, and Z axes while the 3-axis Accelerometer is used to detect the angle of tilt or inclination along the X, Y, and Z axes.

2.3.3 DC Gear Motors

A DC gear motor is a type of electric motor that combines a DC motor with a gearbox to achieve specific torque and speed characteristics. The DC motor provides the rotational power, while the gearbox reduces the motor's speed and increases its torque output by transmitting power through a series of gears. Though it comes in various sizes and power ratings, we have used 12V rated voltage and 120RPM revolving speed motor.

2.3.4 Arduino UNO

Arduino UNO is a microcontroller which has 14 digital input/output pins(of which 6 can be used as PWM outputs), 6 analog inputs, a USB connection. using Arduino UNO is very

easy and has all the requirements that make it fit for selection.

2.3.5 9V Battery

A 9-volt battery is used to power the DC motors through the motor driving module. This is a great fit for regularly used devices such as motorized toys, flashlights, portable game controllers, shavers, CD players, toothbrushes, etc.

2.3.6 A pair of wheels

2.3.7 Connecting Wires

Chapter 3

Methodology

3.1 Design

When constructing the self-balancing robot the main problem is choosing compatible components and making them work together – which requires both hardware and software knowledge. The construction of the bot involves drilling four holes and installing long nuts. A DC gear motor is then mounted on the board, followed by adding a second level to it. The whole motive is to maintain symmetry which adds to the stability of the robot, otherwise, it will be more difficult to balance. Then, the next important point to note is that the center of mass of the robot should be as low as possible. To take care of that the heaviest possible component is placed at the lowest level. In our case, it is the 9V battery that is the heaviest and hence is placed at the lowest base.

3.2 Connections

The two DC gear motors are wired in parallel to the same motor driver, to simplify the software implementation. This wiring will lead to the robot not being able to rotate around its vertical axis. An overview of the wiring is illustrated in the figure. The two motors are connected to M1, M2, M3, and M4 of L293D Motor Drive. The first level includes an

L293D motor driver and an MPU6050 sensor. The motor drive pins IN1, IN2, IN3, and IN4 are connected to digital Arduino pins 9, 8, 6, and 7 respectively. The power supply is provided by a 9V Battery, connecting it with a 12V port of L293D. The gyro sensor is connected to the Serial Clock Line (SCL), Serial Data Line (SDA), and The Interrupt (INT) ports on Arduino to communicate via the I2C protocol.

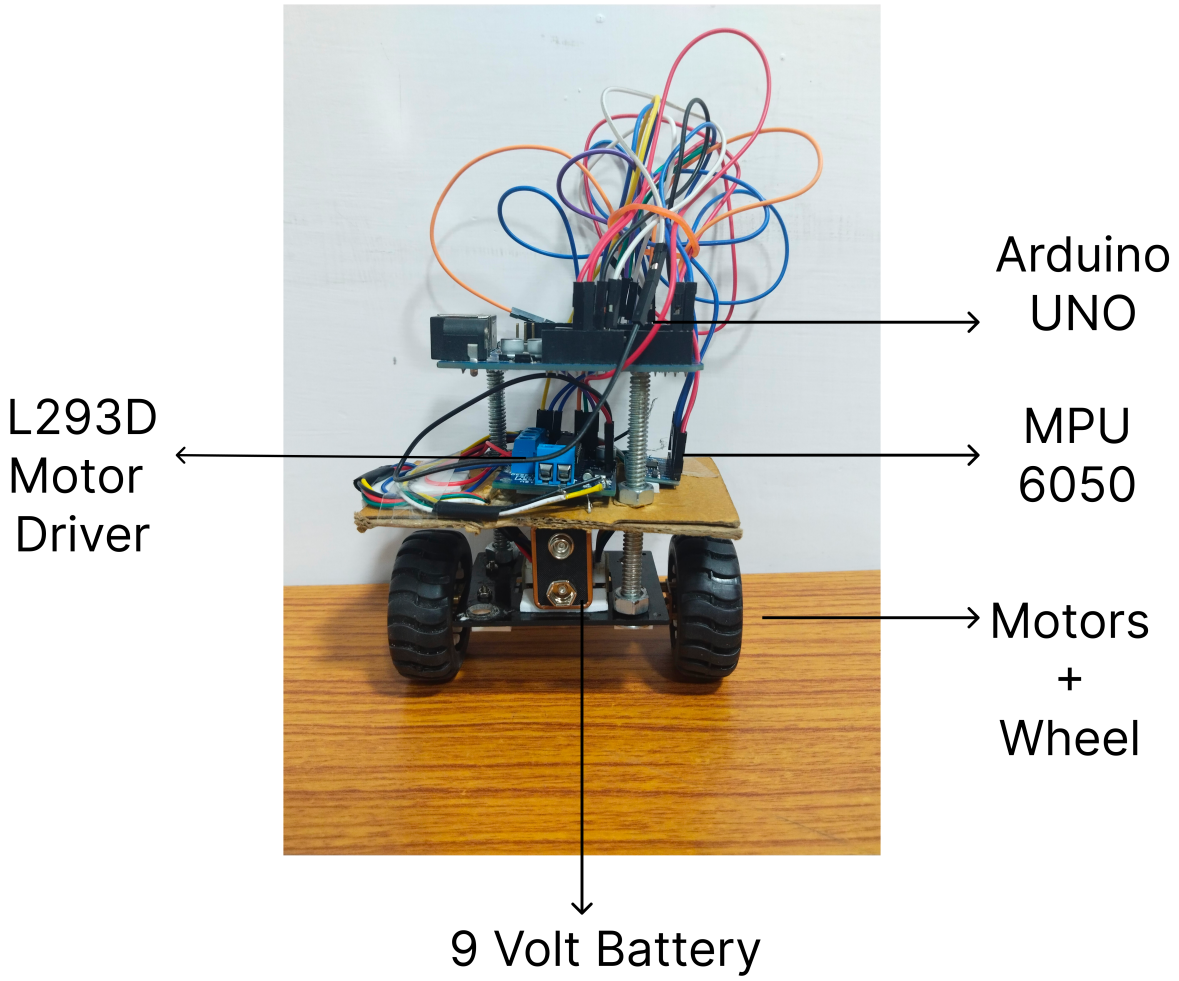

Fig. 3.1 Self-Balancing Bot

3.3 Circuit Diagram

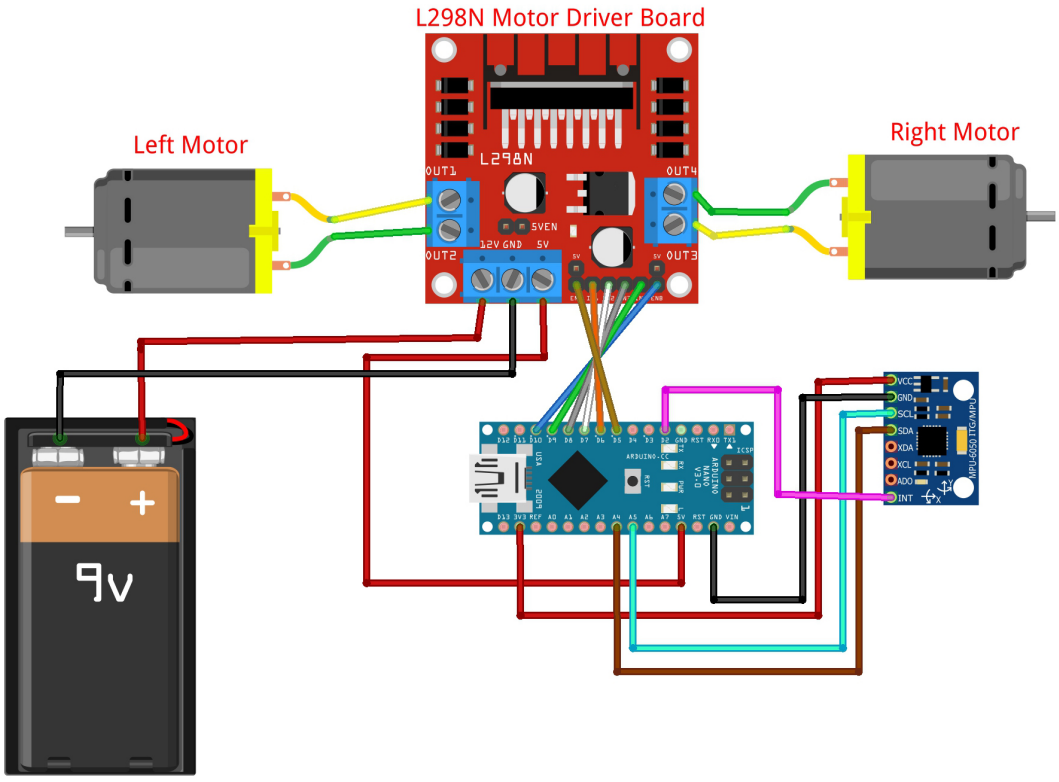

Fig. 3.2 Circuit Diagram of the Self-Balancing Bot

3.4 Software

Arduino UNO is the bridge that connects the hardware with the software part. in which we implement the following libraries "wire.h", "MPU6050.h", and "PID_v1.h".

Chapter 4

Results

4.1 Calculation

We can simplify the model by considering it as a pendulum that hangs upside down from a moving cart.

4.1.1 Kinematic Analysis

The kinematic analysis of the two-wheeled self-balancing robot yields the following equations:

$$v_1 = r\dot{\alpha}_1, \quad v_2 = r\dot{\alpha}_2,$$

$$v_0 = \frac{v_1 + v_2}{2}, \quad (3)$$

$$\dot{\theta} = \frac{|v_1 - v_2|}{L},$$

where $L = 2l$.

The kinetic energies for the wheels are given by:

$$T_1 = \frac{1}{2}mv_1^2 + \frac{1}{2}J_1\left(\frac{v_1}{r}\right)^2,$$

$$T_2 = \frac{1}{2}mv_2^2 + \frac{1}{2}J_2\left(\frac{v_2}{r}\right)^2,$$

where $J_1 = J_2 = mr^2$. We can determine J_3 as:

$$J_3 = \frac{4}{3}m_1h^2.$$

The total kinetic energy of the two-wheeled robot is:

$$T = T_1 + T_2 + T_3,$$

After simplification, we get:

$$T = mr^2(\dot{\alpha}_1^2 + \dot{\alpha}_2^2) + \frac{m}{8r^2}(\dot{\alpha}_1 + \dot{\alpha}_2)^2 + \frac{1}{2}m_1rh(\dot{\alpha}_1 + \dot{\alpha}_2)\dot{\beta}\cos\beta + \frac{7}{6}m_1h^2\dot{\beta}^2.$$

Taking the ground as a reference surface, the total potential energy of the robot is:

$$V = 2mgr + m_1g(h\cos\beta + r).$$

4.2 PID Controller Operation

The control algorithm utilizes a Proportional-Derivative (PD) control approach for stabilizing the inverted pendulum. PID tuning is employed to optimize PID parameters for achieving the desired performance, and state estimation techniques like complimentary filters are used for accurate state estimation. The selected model for the self-balancing bot incorporates a compact design optimized for stability, efficiency, and adaptability. the decided model of the self-balancing bot, featuring a compact and sleek design. which plays a vital role in the bot's actuation mechanism, providing the necessary torque and control for

precise movements.

$$\text{pid} = K_p \times \text{error} + K_i \times \text{integral} + K_d \times \text{derivative} \quad (21)$$

$$\text{integral} += \text{error} \times (\text{millis}() - \text{lasttime}) \quad (22)$$

The derivative term, on the other hand, captures the rate of change of the error and helps in reducing overshoot and oscillations:

$$\text{derivative} = \frac{\text{error} - \text{lasterror}}{\text{millis}() - \text{lasttime}} \quad (23)$$

After computing the PID output, it is essential to limit the output to a specific range to prevent the robot from moving too fast or too slow:

$$\text{pidoutput} = \text{constrain}(\text{pidoutput}, -255, 255) \quad (24)$$

Finally, the last error and last time variables are updated with the current error and current time, respectively, to be used in the subsequent calculations:

$$\text{lasterror} = \text{error} \quad (25)$$

$$\text{lasttime} = \text{millis}() \quad (26)$$

4.3 PID Tuning

We begin the PID tuning process by adjusting the proportional (P) gain. The gain is increased gradually until the robot starts to oscillate around its upright position. Once oscillation is observed, the gain is then reduced until the oscillation subsides. Subsequently, we introduce the derivative (D) gain to further mitigate overshoot or oscillations. The D

gain is increased until the robot begins to overshoot, after which it is decreased to eliminate the overshoot. This iterative process is repeated until we achieve optimal performance for the robot. The final tuned values we obtained are

$$K_p = 15 \; and \; K_d = 170$$

.

Chapter 5

Discussion

5.1 Adversities

The journey of making a Self Balancing Robot led to various adversities that were challenging and fun to overcome.

5.1.1 Challenges

Achieving a stable design where the bot can balance itself was a major challenge due to which we had to restructure the whole robot multiple times. This consumed a lot of time and effort with no results.

Tuning the PID controller to ensure smooth and stable operation was very challenging as incorrect tuning was leading to oscillations and instability.

Motors and sensors require different voltage levels, necessitating proper voltage regulation and management. MPU6050 was running on 3.3V of voltage while the same voltage was very low to run two DC gear motors. Therefore, one must study the desired voltage level of operations required by the various components.

5.1.2 Solutions

To tackle the mechanical design challenge of the robot, careful consideration of the center of gravity, weight distribution, and physical structure. Ensuring that the weight is evenly distributed is crucial for the bot's stability. Any imbalance can cause the bot to tip over. All the components were arranged symmetrically so the center of mass remains in the middle and the heaviest component, i.e., battery in our case was placed at the bottommost possible position.

We continuously tested and iterated the code to identify and rectify design flaws, sensor inaccuracies, and software bugs. Running the code multiple times only led to fixation in values of proportionality constant and integral constant.

5.2 Conclusion

The self-balancing robot project is a very interesting project which is implemented successfully using various components such as Arduino UNO, MPU6050 sensor, L293d motor driver module, 9-volt battery, and other minor components. Special care has been taken to make the hardware symmetric. MPU6050 senses the angle of inclination and the data is then fetched to the PID algorithm. The algorithm then outputs a value that determines the speed of the motor. After numerous experiments and tuning, we finally got the desired value of various constants to achieve a stable and smooth self-balancing of the robot.

5.3 Future Work

The bot has many future scopes and many features can be added to it. Starting with the basic ones which can be easily implemented is making it a line-follower self-balancing robot. Thus, it can move by following a line and will maintain its speed and balance accordingly. Secondly, what one can do is add Bluetooth module and create a remote-controlled robot. The robot can also be controlled by hand motion by adding motion sensors.

Another intermediate work on the self-balancing robot is to make it stand on an inclined plane and thereby test its balance and speed in the inclined plane. This might be more difficult and may require high-precision components.

The currently present features will help in designing more productive system that could make human life easier.

References

[1] https://www.instructables.com/Self-Balancing-Robot/

[2] https://circuitdigest.com/microcontroller-projects/arduino-based-self-balancing-robot

[3] https://youtu.be/aUbBUd-hBLI?si=Qk169xWsyPqX6s8A

[4] https://youtu.be/EEnSDy4PouE?si=DomFP8MHOAjtBUim