Maldives Journal of Engineering and Technology
ISSN: 3007-6536 (Online)
https://mjet.mnu.edu.mv/

Development and Prototyping of a Two-Wheeled Self-Balancing Robot

Rashnan Mohamed Shihab[1, a)], Mohamed Aalif[1,b)], Ali Rawaau[1, c)] , Humaish Adnan[4, d)], Ibrahim Thorig[5, e)]

[1]*Faculty of Engineering, Science and Technology, Maldives National University, Sosun Magu, Male' City, 20068, Maldives;*
[a]*Corresponding Author; s080232@student.mnu.edu.mv*
[b]*s077107@student.mnu.edu.mv*
[c]*s079899@student.mnu.edu.mv*
[d]*s072047@student.mnu.edu.mv*
[e]*Ibrahim.thorig@mnu.edu.mv*

Abstract: Mobile robotics is a branch of robotics and within the branch the category of self-balancing robots is of particular interest as these robots show the promise of navigating difficult terrain in a manner like humans and can be used to platform for investigating autonomous control systems. This paper aims to summarize the development of a two-wheeled self-balancing robot as a case study to demonstrate the application of computer control systems in physical systems. A complementary filter is used with a triple-axis gyroscope and accelerometer to accurately gauge the rotation of the two-wheeled robot and provide the data to a Proportional-Integral-Derivative (PID) program which controls the power to the motors accordingly to control its tilt and achieve self-balancing. In short, the robot manages to achieve self-balancing within a small tilt angle range, however, design flaws such as the sensor falling off at larger tilt angles cause instability at larger tilt angles. In future work, more complex control algorithms can be employed, and the effect of different robot builds can be thoroughly explored.

Keywords: Two-wheeled; Self-Balancing; Robot

1. INTRODUCTION

Researchers have achieved notable progress in mobile robotics, as demonstrated by studies [1, 2, 3], which has established it as a significant area within robotics and providing solution to challenges like autonomous navigation [4]. Two-wheeled self-balancing robots are gaining significant interest among mobile robot types due to their exceptional ability to maintain stability while navigating different terrains. These tiny agile robots frequently change their orientation to prevent tipping, creating excellent platforms for examine dynamic balance and control systems. To emulate human approach to balance, these robots utilize

Received: 5 September 2024
Accepted: 25 October 2024
Published: 23 November 2024

sensor feedback to adjust motor outputs in real-time, allowing them to enabling effective movement in challenging environments. This balancing mechanism has paved the way for not only practical applications in personal transport and automated systems but also serves as a foundation for the development of sophisticated control algorithms.

This paper focuses on the two-wheeled self-balancing robot. Simple self-balancing robots have been implemented before in literature. One such example is in [5, 6, 7], where PID control loop was implemented to regulate motor output. It highlights the need for the robot's structural stability and the accurate calibration of the sensors used in the system. Philip and Golluri [8] have proposed a PID system using the Ardupilot platform and performed simulation and hardware-based testing to analyze trajectory regulation performance. Abdelgawad et al. have carried studies to compare advantages and disadvantages of data-based and model-based analysis.

[10] takes the typical PID-controlled robot even further, making the robot follow a path while balancing and using an ultrasonic sensor to avoid obstacles along the way. [11] spins the control problem in a completely different direction, utilizing a neutral network for motion control.

The robot proposed in this paper follows the model-relies on a simple negative feedback loop, where a gyroscopic sensor collects rotational information that can be used to calculate the tilt angle of the robot. We then use this tilt angle to estimate the force required to return the robot to its initial position. The self-balancing robot then drives this calculated power into the stepper motors that control its wheels. However, relying solely on this approach can lead to increasing overcompensations due to inaccuracies and external factors, prompting the use of a PID system, as demonstrated in [8].

The structure of this paper will be as follows: section 2 will highlight the robot's design and the components used, along with some technical specifications; section 3 will discuss the results and potential factors that could have influenced the robot's performance; and section 4 will provide some concluding remarks.

2. DESIGN AND IMPLEMENTATION

The design and implementation utilized a methodical approach to integrate the chosen components. Initially, we configured the Arduino Uno R3 microcontroller to get real-time data from the MPU050 sensor, which quantifies angular velocity and acceleration to deliver essential information regarding the robot's tilt angle. Applying this data through a complementary filter yields a more precise estimation of the robot's direction. Subsequently, we utilized the Arduino Motor Shield to regulate the stepper motors in accordance with the processed data. We employed a PID control method to manage motor speeds and rectify any divergence from the intended upright posture. This technique ensures the robot's equilibrium, counteracting external disturbances or inaccuracies in sensor data. We assembled all components on a compact breadboard and firmly affixed the physical framework, including the wheels and motors, with wood and hot glue to ensure stability during testing. We generated power through 11V and 9V batteries to guarantee that the motors and sensors had adequate energy for their operation and data processing.

2.1 Components

• The Arduino Uno R3 (x1) utilizes an ATmega328P microcontroller, which can interface with a PC through the USB-B connector. It contains 14 digital I/O pins and 6 analog pins. A battery can power it with 5V and 20mA [10]. We used this microcontroller because of its affordable price, 16 MHz clock speed, direct access to a large GPIO, battery power connector, and the robust ecosystem of products surrounding the Arduino microcontroller lineup.

• The Arduino Motor Shield R3 (x1) enables an Arduino to connect to and control inductive loads, such as relays, DC motors, or stepper motors. It can drive a maximum of 4A, has a free-running stop and brake function, and has the ability to drive 2 DC motors or 1 stepper motor. It operates at a voltage range from 5V to 12V [11]. We used this because it serves as a dedicated motor controller interface for the Arduino Uno R3, enabling the connection of up to 12V power to the motors without the need to send the power through the Arduino's main board, which could potentially damage the board due to its unsuitability for such high-power delivery.

• The MPU6050 (x1) is a triple-axis gyroscope that boasts a maximum scale range of approximately 2500 degrees/sec. Its integration with a 16-bit ADC enables simultaneous gyro sampling. It also contains a digital-output triple-axis accelerometer with a maximum scale range of ±16g, also integrated with a 16-bit ADC to allow for simultaneous sampling [12]. We used the MPU6050 because it integrates a gyroscope and an accelerometer into a compact package. We will explain the need for both sensors in the tilt approximation and balancing section.

a. Breadboard (small) (x1)
b. Wood
c. Hot Glue
d. Wheels (x2)
e. Motors (x2)
f. Jumper Wires
g. Battery (11V) (x1)
h. Batter (9V) (x1)

2.2 Circuit Diagrams

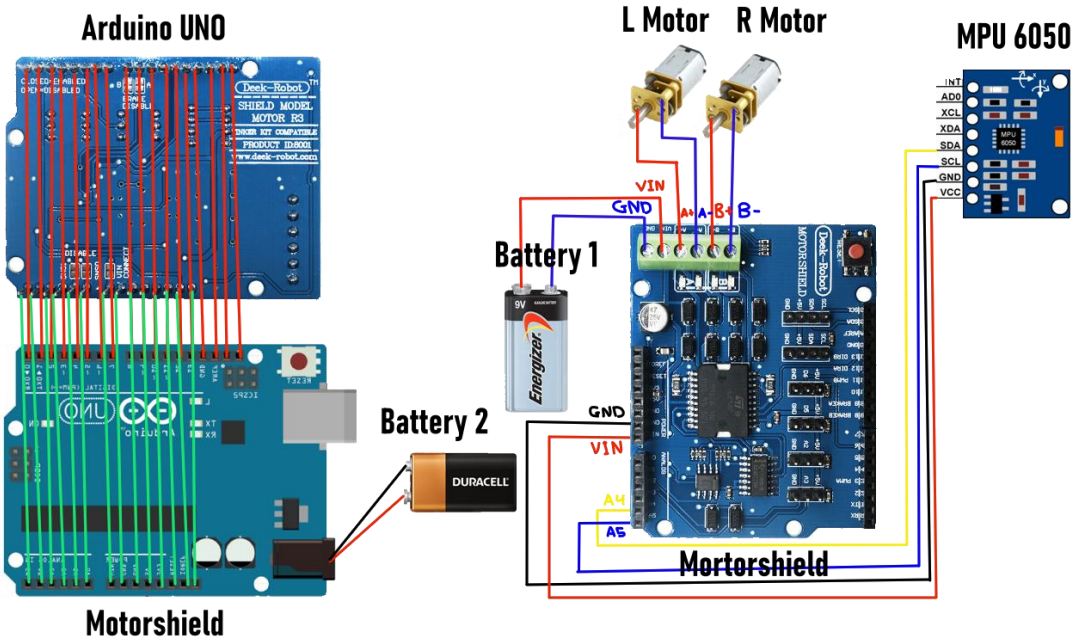

Figure 1. Circuit Diagram

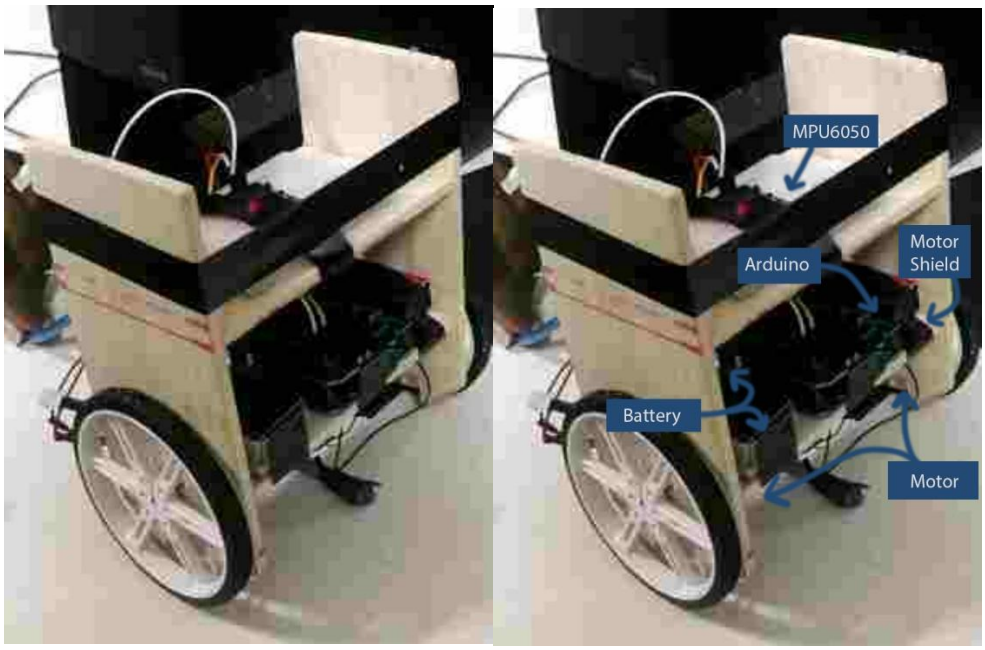

Figure 2. A picture of the robot.

2.3 Tilt Approximation and Balancing

Tilt approximation is done by measuring the accelerometer values every iteration and combining it with a similar reading from the gyroscope (also in the MPU6050) using a complementary filter. Time constant can be adjusted to alter the response of the filter. The value of time constant chosen in this experiment was 0.75.

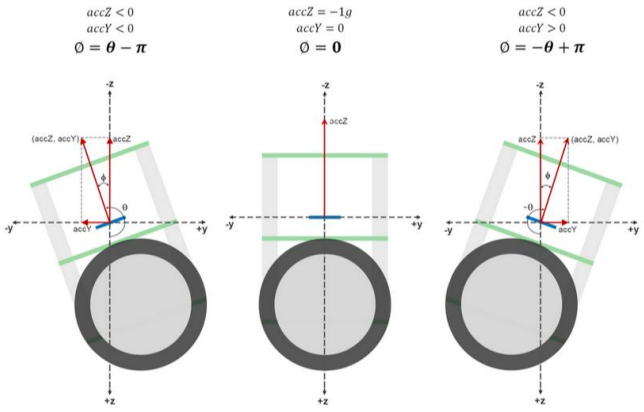

Figure 3. Tilt angle from accelerometer values. (https://www.instructables.com/Arduino-Self-Balancing-Robot-1/)

$$currAngle = \alpha.(prevAngle + gyroAngle) + (1 - \alpha).accAngle$$

Equation 1. Complementary filter equation using filter coefficients.

$$\alpha = \frac{\tau}{\tau + dt}$$

Equation 2. Calculating the filter coefficients using the time constant, τ. While dt, is set to the sampling interval of our Arduino loop function.

After the tilt angle is properly obtained, it is passed into a PID algorithm to determine the output power to the motors. Proportional-Integral-Derivative; Proportional implies that the response must be proportional to the tilt angle, Integral refers to the accumulation of errors in balance correction, and Derivative is used to predict the response of the robot in the next sampling loop.

$$Power = K_p * e(t) + K_i * \int e(t)\,dt + K_d * \frac{d}{dx}e(t)$$

Equation 3. The typical PID equation.

In theory, the larger the deviation of tilt from the intended angle (0 degrees or the upright position), the larger the counterbalancing force that needs to be provided to circumvent falling within a reasonable time frame. This factor is represented by K_p, and is multiplied by the error function, which is the difference between the current tilt angle and the intended tilt angle. The absolute value of this difference indicates the magnitude of the deviation, and the sign can be used to determine the direction the motors need to rotate in to counteract the tilt. Since this calculation is repeated many times a second, the magnitude of the difference will slowly shrink over time if the motor power is applied properly, and the robot will slowly achieve the intended angle. The issue with this is that, at magnitudes close to 0, the minimal angle correction achievable by the motors will be reached and any smaller deviations in angle that are rectified will result in overcompensation, preventing the robot from ever achieving self-balancing. To remedy this issue, a term including the sum of all errors is included, and this term is intended to estimate the accumulated error correction overcompensation, and it is multiplied by a constant, K_i, which signifies the degree of the overcompensation. In addition

to this, one more key factor affects the power required, the momentum from the previous error correction attempt. If the previous tilt correction was towards the right direction and an overcompensation occurred, the next tilt correction would be towards the left, however, momentum from the previous tilt correction would provide an opposing force against the tilt correction. Therefore, a term is added to the equation to simulate this behavior, the derivative of the error, and it is multiplied with the constant K_d, which is used to provide it a modifiable weightage, like the rest of the terms in the equation. The PID constants, K_p, K_i, K_d, depend heavily on the structure and build of the robot and therefore need to be calculated manually, via trial and error.

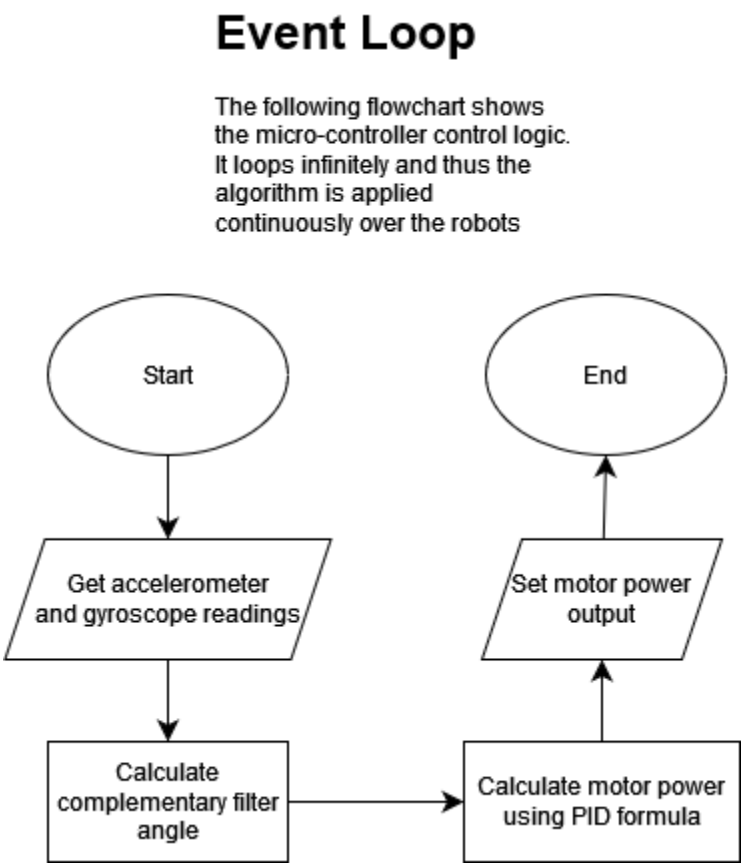

Figure 4. A flow chart showing the event loop of the microcontroller.

3. RESULTS

The initial tuning of the PID controller required several iterations to arrive at the optimal values for the robot's weight and motor capabilities. After multiple trials, the selected parameters allowed the robot to recover from minor disturbances, such as small pushes. However, the system failed to demonstrate the ability to handle larger disturbances effectively, often leading to the robot tipping over before the motors could compensate. We believe that two key design issues are the contributing factors: (1) the MPU6050 sensor regularly dislodges from the breadboard during significant tilts, leaving the motors in an undefined state, and (2) the motors overheated during extended operation, causing the hot

glue securing them to melt. This, in turn, led to misalignment and significantly reduced motor effectiveness, compromising the robot's stability.

Several hardware-related issues also emerged during testing. The Arduino Uno R3 occasionally struggled to establish a dependable connection with the PC. While the board received power, in some cases it failed to communicate with the programming environment. Further investigation revealed a faulty USB cable as the cause, allowing power delivery but not data transfer. Replacing this cable with another one capable of both functions resolved this issue. However, in certain cases, the connection failure persisted, suggesting potential board-level issues, for which board replacement was the most practical solution.

The integration of the MKR IMU Shield also presented difficulties. Initially, the MKR board did not detect the shield, likely due to loose connections. Disassembling and reassembling the components resolved the problem, allowing the shield to function properly. Additionally, the Arduino MKR 1000 Wi-Fi, which had been selected for its wireless connectivity and its compact form factor, failed to connect to the PC during later testing, despite being in proper working order the day before. The root cause of this malfunction could not be identified, and thus the use of an alternative Arduino model was unavoidable.

Analysis of the motor power output showed that the power throughput of the GPIO pins in the Arduino Uno R3 was insufficient for powering the stepper motors, as they failed to spin at a high enough speed and could not achieve the torque necessary to rebalance the robot. This issue was easily settled with the use of a dedicated motor shield, the Arduino Motor Shield R3, which used a relay and allowed the use of a dedicated high-voltage battery to power the motors, while the Arduino Uno R3 could remain powered by the pre-existing battery. This highlights the importance of using a motor shield, as it isolates the motor power from the main Arduino board, preventing high-voltage loads from passing through the main board and damaging it.

The MPU6050 sensor initially presented problems with data communication. The sensor failed to initialize properly during the early stages of testing, most likely due to loose connections with the breadboard. Soldering the connections was enough to fix this, and the sensor began to function normally, providing stable and reliable readings to the control system.

The robot's delayed response to minor tilt disturbances emerged as the most significant issue during testing. While initially, the robot solely relied on accelerometer data to detect tilts, this proved insufficient. The slow response resulted in the robot reaching an angle outside of its recoverable range before the motors could act. The slow responses were caused by inaccuracies in the tilt angle estimation, and to address this, a complementary filter was implemented. This combined gyroscopic data was also measured by the MPU6050 and combined it with the previous accelerometer data to provide a more accurate estimate. Although this improved responses to small tilts, the system remains susceptible to larger disturbances, which indicates the necessity of further refining the PID control parameters.

In summary, the current design and implementation succeeded in stabilizing the robot under moderate conditions, but challenges remain with respect to handling larger

disturbances and optimizing sensor placement and power delivery. The addition of a complementary filter markedly improved response times due to the better tilt estimations; however, further tuning of the PID constants and additional structural modifications will be needed to fully achieve robust self-balancing behavior.

4. CONCLUSION

This research presents a self-balancing robot as a case study in the implementation of digital control systems in physical systems. The incorporation of a supplementary filter using a Proportional-Integral-Derivative (PID) control algorithm shown efficacy in sustaining balance within the permissible limits established during testing and assessment. The complementary filter enabled precise tilt angle estimation by integrating data from an accelerometer and a gyroscope, while the PID control system determined the requisite motor power to counteract deviations from the vertical orientation, demonstrating significantly greater reliability than relying exclusively on accelerometer or gyroscopic data. Although excellent stabilization for mild disturbances was attained, the adjustment of PID constants was laborious, and hardware issues, including sensor misalignment and motor overheating, adversely affected the system's performance and stability. Future researchers should tackle these concerns by optimizing control parameters or employing an automated approach for their refinement, as well as investigating sophisticated control systems to enhance response times and overall reliability. This study highlights the importance of iterative design and testing in robotic control systems and establishes a basis for future progress in mobile robot stabilization.

Acknowledgment: Thank you to MNU, and our lecturer, for providing most of the necessary components for us and allowing us to utilize their facilities for development and testing.

Conflicts of Interests: None

Data Availability Statement: Open

Author Contributions: All the authors were involved in the research, development and testing.

5. REFERENCES

[1] R. Keith and H. M. La, "Review of Autonomous Mobile Robots for the Warehouse Environment," arXiv, 2024, https://doi.org/10.48550/arXiv.2406.08333.

[2] A. Kumar, A. Sahasrabudhe and S. Nirgude, "Fuzzy Logic Control for Indoor Navigation of Mobile Robots," arXiv, 2024, https://doi.org/10.48550/arXiv.2409.02437.

[3] Q. Lin and W. Lu, "Collaborative Goal Tracking of Multiple Mobile Robots Based on Geometric Graph Neural Network," arXiv:2311.07105, p. 1, 2023.

[4] S. Nahavandi, R. Alizadehsani, D. Nahavandi, S. Mohamed, N. Mohajer, M. Rokonuzzaman and I. Hossain, "A Comprehensive Review on Autonomous Navigation," arXiv:2212.12808, p. 1, 2022. https://doi.org/10.48550/arXiv.2212.12808.

[5] S. E. Arefin, "Simple Two-wheel Self-Balancing Robot Implementation," arXiv:2303.12067, 2023, https://doi.org/10.48550/arXiv.2303.12067.

[6] H. Ren and C. Zhou, "Control System of Two-Wheel Self-Balancing Vehicle," Journal of Shanghai Jiaotong University (Science), vol. 26, pp. 713-721, 2021, https://doi.org/10.1007/s12204-021-2361-x.

[7] N. T and P. K. T, "PID Controller Based Two Wheeled Self Balancing Robot," in IEE, Tirunelveli, India, 2021, https://doi.org/10.1109/ICOEI51242.2021.9453091.

[8] E. Philip and S. Golluri, "Implementation of an Autonomous Self-Balancing Robot Using Cascaded PID Strategy," IEEE, 2020, https://doi.org/10.1109/ICCAR49639.2020.9108049.

[9] A. Abdelgawad, T. Shohdy and A. Nada, "Model- and Data-Based Control of Self-Balancing Robots: Practical Educational Approach with LabVIEW and Arduino," arXiv:2405.03561v1, 2024, https://doi.org/10.48550/arXiv.2405.03561 .

[10] Arduino, "UNO R3," 24 8 2024. [Online]. Available: https://docs.arduino.cc/hardware/uno-rev3/#tech-specs.

[11] Arduino, "Arduino Motor Shield Rev3," 28 8 2024. [Online]. Available: https://store-usa.arduino.cc/products/arduino-motor-shield-rev3?selectedStore=us.

[12] InvenSense, "InvenSense Inc.," 28 8 2024. [Online]. Available: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf.