iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** IV **Month of publication:** April 2026

DOI: https://doi.org/10.22214/ijraset.2026.79346

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue IV Apr 2026- Available at www.ijraset.com

Modular Self-Balancing Robotic Platform for Multi-Application Use

Nandhana M J[1], Noyal K Raju[2], Ashik V Rajendran[3], Cherian Eldho[4], Dr. Reenu George[5], Dr. Dinto Mathew[6]
[1, 2, 3, 4]*Department of Electrical and Electronics Engineering, Mar Athanasius College of Engineering, Kothamangalam*
[5, 6]*Assistant Professor, Department of Electrical and Electronics Engineering, Mar Athanasius College of Engineering, Kothamangalam*

Abstract: ***This paper presents the hardware implementation of a modular two-wheeled self-balancing robotic platform designed to support multiple real-world applications through a swappable sensor and module architecture. The system employs a dual-controller design where an ESP32 microcontroller executes a cascaded PID control algorithm for real-time balance stabilisation, using tilt angle data acquired from an MPU6050 Inertial Measurement Unit. DC motors fitted with incremental encoders provide closed-loop speed feedback, enabling precise wheel actuation. A Raspberry Pi serves as the high-level controller, handling computer vision tasks independently from the balancing loop, ensuring that processing overhead does not affect stability. A line-following module is demonstrated as a practical application, implemented using OpenCV on the Raspberry Pi, which relays directional commands to the ESP32 via I2C communication. The physical platform follows a three-tier modular chassis design fabricated from mild steel that allows application-specific hardware to be integrated or removed without modifying the core control system. Experimental results confirm stable balancing and reliable autonomous line-following behaviour, validating both the control strategy and the modular hardware concept. The platform offers a cost-effective and reconfigurable foundation for applications spanning smart agriculture, service robotics, and autonomous navigation.***

Keywords: Self-balancing robot, Inverted pendulum, ESP32, MPU6050, PID control, Cascaded control, Line following, OpenCV, Modular robotics, Dual-controller architecture.

I. INTRODUCTION

The growing interest in autonomous mobile robotics has driven significant research into dynamic stabilisation, embedded control systems, and real-time feedback mechanisms. Among the various robotic configurations explored in literature, two-wheeled self-balancing robots have emerged as a particularly valuable study platform due to their inherent instability, which demands active and continuous control to maintain an upright position. This characteristic makes them an ideal physical representation of classical control problems such as the inverted pendulum, offering a tangible and engaging medium for understanding feedback control theory in practice.

Despite this potential, most commercially available or research-grade self-balancing platforms remain either prohibitively expensive or architecturally rigid, limiting their accessibility in resource-constrained academic environments. Low-cost implementations that retain meaningful control complexity are relatively rare, and fewer still demonstrate a clean separation between real-time stabilisation and higher-level task execution within an affordable embedded design. This gap motivates the development of a platform that prioritises cost-effectiveness and architectural clarity without sacrificing control performance.

This paper presents the design and hardware implementation of a two-wheeled self-balancing robot built around a dual-controller architecture. An ESP32 microcontroller handles all time-critical balance operations through a cascaded PID control algorithm, processing orientation data from an MPU6050 Inertial Measurement Unit in real time. A Raspberry Pi operates alongside it as a higher-level controller, managing computer vision tasks such as line following via OpenCV without placing any computational burden on the balancing loop. DC motors with incremental encoders complete the actuation system, providing closed-loop speed feedback for stable and responsive wheel control. The entire system is built using widely available, affordable components, keeping the platform reproducible and practical for educational and research settings.

In this context, the platform directly supports SDG 4 on Quality Education by providing an accessible, hands-on environment for learning embedded control systems and robotics. Its low-cost construction also aligns with SDG 9, which advocates for inclusive technological innovation, demonstrating that meaningful control systems research need not depend on expensive proprietary hardware. The remainder of this paper is organised as follows. Section II reviews relevant prior work on self-balancing robot design and control strategies.

Section III describes the overall system architecture and hardware components. Section IV details the control system implementation. Section V presents the line-following application module. Section VI discusses experimental results, Section VII outlines future directions, and Section VIII concludes the paper.

II. RELATED WORK

A. PID and Classical Control Approaches

Classical PID-based control remains the most widely adopted strategy for two-wheeled self-balancing robots owing to its straightforward implementation and reliable performance. Nikita et al. developed a self-balancing robot using an Arduino Uno with an MPU6050 IMU, employing sensor fusion between the onboard accelerometer and gyroscope for tilt estimation, with MATLAB-Proteus co-simulation used for PID parameter tuning.

Their work is among the closest in hardware configuration to the system presented here, though it does not address the separation of balancing and application-level processing. Sarathy et al. implemented a PD controller on the NI myRIO platform, demonstrating stable balancing performance under varying loads, but the proprietary nature of the hardware significantly limits reproducibility in low-resource settings. Iwendi et al. proposed a PD-PI control scheme combined with Kalman filtering for sensor fusion and obstacle-aware navigation, achieving improved noise rejection compared to standard PID implementations. Abdelgawad et al. offered a direct experimental comparison between model-based PID control and data-based fuzzy logic control on a low-cost Arduino platform, concluding that while fuzzy logic reduces modelling requirements, PID controllers deliver faster and more accurate responses when the system is reasonably well characterised.

Kim and Ahn proposed a self-tuning outer-loop position controller paired with an inner-loop Linear Quadratic Regulator, achieving approximately 48% improvement in position-tracking performance, though the added complexity makes the approach less suitable for cost-constrained educational implementations.

B. Advanced and Hybrid Control Methods

Several studies have explored control strategies beyond classical PID to address limitations in robustness and adaptability. Khabazi and Shariyat demonstrated a fully simulation-driven design workflow integrating MATLAB, SolidWorks, Gazebo, and ROS2, enabling autonomous navigation with virtual LiDAR and SLAM-based mapping. While this establishes a powerful validation methodology, the computational and software infrastructure required places it outside the reach of most resource-limited settings. Chen et al. introduced a composite two-wheeled robot incorporating a reaction wheel as a secondary balancing mechanism, controlled by an Adaptive Double-Fuzzy Anti-Integral Saturation PID controller. This design achieves static balance and stable traversal of slopes up to 29°, significantly outperforming conventional two-wheeled platforms, though the mechanical and algorithmic complexity involved represents a considerable increase in both cost and development effort.

C. Application-Oriented Platforms

A smaller body of work has focused on extending self-balancing or mobile robot platforms toward practical task execution. Diallo et al. developed a Raspberry Pi-based autonomous tour guide robot using ultrasonic sensors for wall-following navigation and OpenCV for marker recognition, achieving identification accuracy of up to 98% under adequate lighting conditions. Their use of a Raspberry Pi for vision processing on a mobile platform directly informs the architectural approach taken in this work. Shukla et al. demonstrated a self-balancing robot for agricultural crop monitoring, integrating soil and environmental sensors with machine learning-based fruit ripeness classification, highlighting the potential of mobile balancing platforms in field applications.

D. Research Gap

While the works reviewed above collectively advance the understanding of balancing control, sensor fusion, and task-specific robotics, a common limitation is the lack of attention to architectural separation between real-time control and application-level processing within a genuinely low-cost hardware framework. High-performing systems tend to rely on expensive or specialised hardware, while low-cost implementations rarely demonstrate clean dual-controller designs capable of running vision-based tasks without compromising balancing stability. The platform presented in this paper addresses this gap by combining an ESP32 for dedicated real-time PID control with a Raspberry Pi for independent vision processing, achieving a clear functional separation at minimal cost.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue IV Apr 2026- Available at www.ijraset.com

III. SYSTEM ARCHITECTURE AND HARDWARE DESIGN

A. Overview

The platform is built around a dual-controller architecture that deliberately separates time-critical balance control from higher-level application processing. This separation ensures that computational tasks such as image processing never interfere with the fast control loop responsible for maintaining stability. An ESP32 microcontroller serves as the low-level real-time controller, while a Raspberry Pi operates as the supervisory high-level processor. Both controllers communicate over an I2C interface, with the Raspberry Pi issuing directional commands and the ESP32 executing them within the constraints of the active balancing loop (Fig. 1). The overall system is divided into four subsystems — power, sensing, control, and actuation — each described in detail below.

B. Power Subsystem

The entire platform is powered by a single 2000 mAh Lithium Polymer battery. A buck converter steps the battery voltage down to a stable 5 V rail to power the Raspberry Pi, while the motors are driven at 12 V directly from the battery through the motor driver. The ESP32 draws its operating voltage from the Raspberry Pi's USB output, simplifying the power distribution layout. This arrangement keeps the power circuit compact while ensuring each subsystem receives a voltage level appropriate to its operating requirements.

C. Sensing Subsystem

The MPU6050 Inertial Measurement Unit forms the core of the sensing subsystem. It integrates a three-axis accelerometer and a three-axis gyroscope on a single chip, communicating with the ESP32 over the I2C bus. The accelerometer provides a steady reference to gravity for computing the static tilt angle, while the gyroscope captures rapid angular velocity changes that the accelerometer alone cannot track reliably at high frequencies. Together, these two measurements are fused by the MPU6050's onboard Digital Motion Processor (DMP) to produce stable, real-time orientation estimates. A USB webcam mounted on the upper tier of the chassis feeds video data directly to the Raspberry Pi for line detection processing.

D. Control Subsystem

The ESP32 acts as the dedicated real-time controller, running the PID balancing algorithm at a fixed sampling rate. It receives orientation data from the MPU6050 DMP, computes the control output, and sends motor commands to the driver at each control cycle. The Raspberry Pi runs the OpenCV-based line-following algorithm, processing webcam frames to determine the robot's lateral deviation from a line. Based on this, it sends corrective directional instructions to the ESP32 over the I2C bus. The ESP32 then blends these commands into the balancing loop, adjusting motor speeds accordingly while continuing to maintain upright stability.

E. Actuation Subsystem

Two DC motors fitted with incremental encoders drive the left and right wheels independently. The encoders provide closed-loop speed feedback to the ESP32, allowing accurate wheel velocity control. The BTS7960 motor driver serves as the interface between the ESP32 and the motors. Compared to commonly used drivers such as the L298N, the BTS7960 offers a significantly higher continuous current rating of up to 43 A, lower on-resistance, and better thermal performance — making it well suited for driving motors under the dynamic load conditions typical of a self-balancing platform. PWM signals from the ESP32 control the driver, which in turn regulates motor speed and direction.

F. Mechanical Design

The chassis is fabricated from mild steel and follows a three-tier vertical layout. The bottom tier houses the motors and wheel assemblies. The middle tier carries the ESP32, BTS7960 motor driver, and power management components. The upper tier provides a mounting platform for the Raspberry Pi, webcam, and battery pack. This vertical arrangement keeps the centre of mass elevated above the wheel axis, which is a necessary condition for the inverted pendulum dynamics the PID controller is designed to stabilise as seen in Fig 1.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue IV Apr 2026- Available at www.ijraset.com

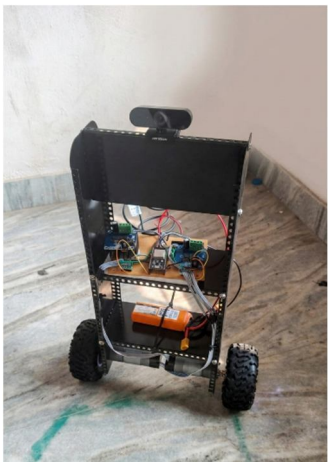


Fig. 1 Hardware implementation of the robot

IV. CONTROL SYSTEM IMPLEMENTATION

A. Tilt Angle Estimation

Accurate and low-latency tilt estimation is fundamental to the performance of any self-balancing platform. Raw accelerometer data alone is susceptible to vibration noise from the motors, while gyroscope data drifts over time due to integration error. Rather than implementing a software-based fusion filter on the ESP32, this system leverages the MPU6050's onboard Digital Motion Processor (DMP). The DMP performs sensor fusion internally, combining accelerometer and gyroscope readings to produce stable quaternion-based orientation estimates. These are then converted to Euler angles on the ESP32 to extract the pitch angle relevant to balancing. This approach offloads the computationally intensive fusion calculation from the ESP32, freeing its processing capacity entirely for the control loop and motor management tasks.

B. Cascaded PID Control Architecture

The balancing system is implemented as a cascaded PID structure consisting of two interlinked control loops. The inner loop is responsible for tilt stabilisation. It receives the current pitch angle from the DMP and compares it against a setpoint angle, which under static balancing conditions is the upright equilibrium angle of the robot. The error between the measured and desired tilt is fed into the inner PID controller, which computes a corrective motor output according to:

$$u(t) = K_p e(t) + K_i \int e(\tau)\, d\tau + K_d\, de(t)/dt \quad (1)$$

where e(t) is the tilt error, and Kp, Ki, Kd are the proportional, integral, and derivative gains respectively. The proportional term generates an immediate corrective response proportional to the current tilt, the integral term eliminates steady-state offset that accumulates over time, and the derivative term dampens oscillations by anticipating the rate at which the error is changing.

The outer loop handles velocity regulation. It monitors wheel encoder feedback to estimate the robot's actual forward or backward velocity and computes a velocity error relative to a target speed. The output of the outer loop dynamically adjusts the tilt setpoint passed to the inner loop — for instance, commanding a small forward lean to accelerate, or a slight backward lean to decelerate. This cascaded arrangement allows the robot to simultaneously maintain balance and execute controlled translational movement (Fig. 2).

C. PID Gain Tuning

The PID gains for both loops were determined through systematic manual tuning on the physical hardware. Tuning began with the inner tilt loop, with Ki and Kd set to zero. The proportional gain Kp was increased incrementally until the robot could sustain balance with visible but bounded oscillations. The derivative gain Kd was then introduced to suppress these oscillations and improve damping. Finally, a small integral gain Ki was added to correct for residual lean caused by uneven weight distribution. Once the inner loop was stable, the outer velocity loop gains were tuned following the same sequence. The final simulation gain values used are listed in Table I.

TABLE I
PID GAIN VALUES USED IN SIMULATION

	Parameter	Loop	Value
1	Kp	Inner(tilt)	70
2	Ki	Inner(tilt)	0.5
3	Kd	Inner(tilt)	1.2
4	Kp	Outer(tilt)	0.5
5	Ki	Iouter(tilt)	0.05
6	Kd	Outer(tilt)	0.1
7	Sampling Time		0.01

D. Motor Control and Encoder Feedback

The PID output is converted to a PWM duty cycle and a direction signal sent to the BTS7960 motor driver. The driver independently controls the left and right motors, allowing differential speed commands that enable turning while balancing. Incremental encoders mounted on each motor shaft generate pulse trains proportional to wheel rotation speed. The ESP32 reads these pulses using hardware interrupt pins to ensure no encoder counts are missed even at high motor speeds. The encoder feedback is used by the outer velocity loop to prevent the robot from drifting indefinitely in one direction during stationary balancing.

E. Communication with Raspberry Pi

High-level navigation commands from the Raspberry Pi are transmitted to the ESP32 over the I2C bus, with the Raspberry Pi acting as the master and the ESP32 as the slave device. Commands are encoded as single-byte instructions representing directional actions such as forward, reverse, turn left, and turn right. Upon receiving a command, the ESP32 modifies the outer loop velocity setpoint accordingly, while the inner tilt loop continues operating uninterrupted at its full sampling rate. This decoupled communication model ensures that any processing delay on the Raspberry Pi side does not stall or destabilise the balancing control loop.

V. LINE FOLLOWING MODULE

A. Overview

The line following module serves as the primary application demonstration of the platform, validating the practical utility of the dual-controller architecture under real task conditions. The module runs entirely on the Raspberry Pi, processing visual input from the USB webcam to determine the robot's lateral position relative to a black line on a white surface. Navigation corrections derived from this processing are communicated to the ESP32 over I2C, which incorporates them into the active balancing loop without interrupting tilt stabilisation.

B. Image Acquisition and Preprocessing

Frames are captured continuously from the USB webcam using OpenCV's VideoCapture interface. Each captured frame is first converted from its native BGR colour space to grayscale, reducing the three-channel image to a single intensity channel. A binary threshold is then applied to the grayscale frame, producing a two-tone image where pixels below a defined intensity value — corresponding to the dark line — are set to zero, and all remaining pixels representing the white surface are set to 255. The threshold value is selected to provide reliable contrast separation under the expected indoor lighting conditions.

C. Line Detection and Centroid Calculation

Following thresholding, the binary image is cropped to a horizontal region of interest near the bottom of the frame, focusing detection on the portion of the line directly ahead of the robot. Connected contours within this region are identified using OpenCV's contour detection function. The largest contour by area is selected as the detected line segment, and its centroid is computed using image moments:

$$c_x = M_{10} / M_{00} \quad (2)$$

where M_{00} is the zeroth moment representing the contour area, and M_{10} is the first spatial moment along the horizontal axis. The resulting centroid coordinate cx represents the horizontal position of the line within the frame.

D. Deviation Calculation and Command Generation

The lateral deviation of the robot from the line is computed as:

$$e_{line} = c_x - W/2 \quad (3)$$

where W is the frame width in pixels. A positive value of eline indicates the line lies to the right of centre, while a negative value indicates it lies to the left. Based on the magnitude and sign of this error, the Raspberry Pi generates one of three directional commands — turn left, move forward, or turn right — and transmits the corresponding instruction byte to the ESP32 over I2C. A dead band is applied around zero error to prevent unnecessary corrections during straight sections, reducing oscillatory steering behaviour.

E. Integration with the Balancing Loop

Upon receiving a directional command, the ESP32 adjusts the velocity setpoint of the outer PID loop to steer the robot in the instructed direction. A turn-left command increases the speed setpoint of the right motor relative to the left, and a turn-right command does the opposite, producing a differential drive steering response. Throughout this process the inner tilt stabilisation loop continues operating at its full sampling rate, completely unaffected by the vision processing occurring on the Raspberry Pi. Even if a particular video frame takes longer than usual to process, the robot remains balanced because the ESP32 never waits on the Raspberry Pi to complete its control cycle.

VI. RESULTS AND DISCUSSION

A. Simulation Validation

Prior to hardware assembly, the control strategy and mechanical design were validated through a series of simulation experiments. The cascaded PID controller was implemented in CoppeliaSim Edu on a two-wheeled rigid body model, with simulated IMU tilt feedback and wheel velocity feedback driving the inner and outer control loops respectively. The simulation confirmed that the cascaded control architecture produces stable balancing behaviour and responsive heading control with the selected gain values, and that inner loop tilt deviations remained within acceptable bounds during forward and reverse motion.

Motor behaviour was independently verified through two separate simulation campaigns. Stepper motor driver logic and step sequencing were validated in Proteus 8, confirming correct torque availability under expected load conditions. DC motor encoder pulse generation and closed-loop velocity response were verified in Tinkercad, where the observed encoder count rate confirmed an effective resolution of 200 pulses per revolution and a motor time constant of approximately 0.25 seconds — sufficient for the response speed requirements of the balancing application.

B. Hardware Implementation

Following simulation validation, the physical platform was assembled according to the CAD design. The ESP32 and BTS7960 motor driver are housed on the middle tier alongside the power regulation circuit, while the Raspberry Pi, USB webcam, and LiPo battery pack occupy the upper tier. The MPU6050 IMU is mounted centrally on the middle tier to minimise vibration interference from the motors (Fig. 3).

C. Balancing Performance

The assembled platform demonstrated stable upright balancing upon successful PID gain tuning on the physical hardware. The system responded effectively to manual disturbances, recovering to the upright equilibrium position within the settling time reported in Table II. The BTS7960 motor driver performed reliably throughout testing with no thermal issues observed under continuous operation. The encoder feedback contributed meaningfully to velocity regulation, preventing the steady drift typically observed in open-loop balancing implementations.

D. Line Following Performance

The line following module was tested on a track consisting of a black line approximately 3 cm wide on a white surface under standard indoor lighting. The Raspberry Pi successfully processed webcam frames in real time, detecting the line centroid and generating corrective steering commands at a rate sufficient to maintain tracking during forward motion. The dead band applied around zero lateral error effectively reduced unnecessary steering corrections on straight sections, resulting in smoother forward motion. Key performance metrics are summarised in Table II.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue IV Apr 2026- Available at www.ijraset.com

TABLE II
HARDWARE PERFORMANCE METRICS

	Metric	Value
1	Balancing settling time after disturbance	7s
2	Maximum recoverable tilt angle	10 degrees

E. Discussion

The results confirm that the dual-controller architecture achieves its primary design objective — maintaining stable balance while executing a vision-based navigation task simultaneously. The separation of the tilt control loop on the ESP32 from the OpenCV processing on the Raspberry Pi proved effective in practice, as no balancing instability was observed during active line following despite the inherent variability in frame processing time. The use of the MPU6050 DMP for onboard sensor fusion simplified the ESP32 firmware considerably while delivering reliable tilt estimates. The overall component cost of the platform remains low, with all hardware sourced from widely available suppliers, supporting the goal of producing a reproducible and accessible platform for control systems education and applied robotics research.

VII. FUTURE SCOPE

A. Advanced Control Strategies

The current implementation relies on manually tuned PID gains, which while effective, require repeated iteration when the platform's physical configuration changes. A logical next step is the implementation of an auto-tuning or adaptive control strategy that adjusts gains in real time based on observed system response. Model-based approaches such as Linear Quadratic Regulation or model predictive control could also be explored as alternatives offering more systematic performance guarantees, particularly under varying load conditions.

B. Enhanced Navigation and Autonomy

The line following module demonstrated in this work represents a basic form of structured navigation. Future development will look toward more generalised autonomous navigation capabilities, including obstacle detection and avoidance using ultrasonic or infrared sensors, and map-based path planning using simultaneous localisation and mapping techniques. Integrating a depth-sensing camera would further expand the robot's ability to operate in unstructured environments where predefined line tracks are unavailable.

C. Additional Application Modules

An agricultural sensing module incorporating soil moisture and pH sensors could enable the robot to autonomously traverse field rows and log environmental data for precision farming applications. A human interaction module featuring audio output and speech recognition could adapt the platform for use as an indoor guide or information delivery robot. Each of these extensions can be developed and tested independently without modifying the core balancing system.

D. Multi-Agent Coordination

A particularly promising direction is the deployment of multiple platforms operating collaboratively as a swarm. The ESP32's built-in Wi-Fi and Bluetooth capabilities provide a natural foundation for wireless inter-agent communication without requiring additional hardware. Swarm-based task allocation, formation control, and cooperative mapping represent research challenges well suited to this platform given its low cost and reproducibility.

E. Improved Mechanical Design

Future iterations could explore aluminium or carbon fibre composite construction to reduce overall platform mass, improving the power-to-weight ratio and extending battery life. A 3D printed modular mounting system for the upper tier would further simplify the process of attaching and detaching application-specific hardware, reducing reconfiguration time in research and educational settings.

VIII. CONCLUSION

This paper presented the design and hardware implementation of a low-cost two-wheeled self-balancing robotic platform built around a dual-controller architecture. The system addressed a practical gap in existing literature by demonstrating a clean and deliberate separation between real-time balance control and higher-level vision-based task execution within an affordable embedded hardware framework. An ESP32 microcontroller was dedicated entirely to cascaded PID-based tilt stabilisation, processing orientation data from the MPU6050 DMP at a fixed sampling rate, while a Raspberry Pi handled OpenCV-based line following independently without imposing any computational burden on the balancing loop. DC motors with incremental encoders provided closed-loop wheel speed feedback, and the BTS7960 motor driver delivered reliable high-current actuation throughout operation.

Simulation-based validation in CoppeliaSim, Proteus 8, and Tinkercad confirmed the viability of the control strategy and motor selection prior to hardware assembly. Physical testing demonstrated stable balancing performance and successful autonomous line following on a structured track, validating both the control implementation and the dual-controller communication model over I2C. The mild steel three-tier chassis provided a rigid and well-organised mechanical platform accommodating all subsystems within a compact footprint.

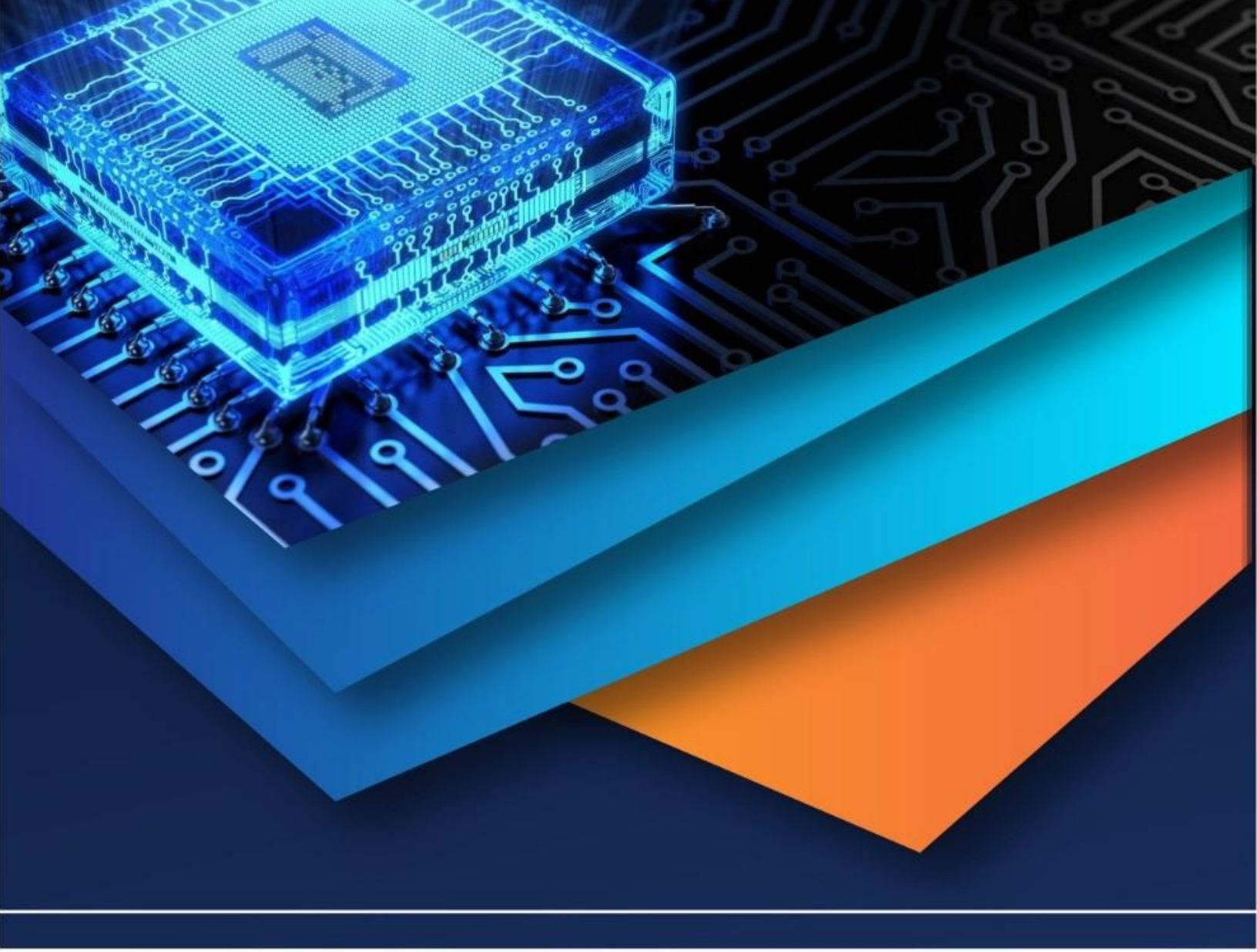
Beyond its technical contributions, the platform demonstrates that meaningful control systems complexity — including sensor fusion, cascaded feedback control, closed-loop motor regulation, and computer vision — can be realised at low cost using widely available components. This makes it a genuinely practical tool for undergraduate robotics and control systems education, directly contributing to the goals of SDG 4 on Quality Education and SDG 9 on inclusive technological innovation.

IX. ACKNOWLEDGMENT

The authors would like to thank the Department of Electrical and Electronics Engineering, Mar Athanasius College of Engineering, Kothamangalam, for providing the resources and facilities necessary to carry out this work.

REFERENCES

[1] A. Abdelgawad, T. Shohdy, and A. Nada, “Model- and data-based control of self-balancing robots: Practical educational approach with LabVIEW and Arduino,” arXiv preprint arXiv:2405.03561, 2024.
[2] H. Khabazi and M. Shariyat, “Control and simulation of autonomous navigation of a self-balancing robot,” in Proc. 12th RSI Int. Conf. Robotics and Mechatronics (ICRoM), Tehran, Iran, pp. 226–232, 2024.
[3] J. Chen, N. He, Z. Xu, M. Dou, and L. He, “Design and implementation of a novel two-wheeled composite self-balancing robot for stationary operations in unknown terrain,” IEEE Access, vol. 13, pp. 86032–86045, 2025.
[4] S. Sarathy, M. H. M. M., S. Kalaivani, and A. S., “Implementation of efficient self balancing robot,” in Proc. Int. Conf. Recent Trends in Electrical, Control and Communication (RTECC), Chennai, India, 2018.
[5] N. T. and P. K. T., “PID controller based two wheeled self balancing robot,” in Proc. Fifth Int. Conf. Trends in Electronics and Informatics (ICOEI), Bangalore, India, 2021.
[6] C. Iwendi, J. H. Anajemba, M. A. Alqarni, A. S. Alfakeeh, Z. Zhang, and A. K. Bashir, “Robust navigational control of a two-wheeled self-balancing robot in a sensed environment,” IEEE Access, 2019.
[7] A. D. Diallo, S. Gobee, and V. Durairajah, “Autonomous tour guide robot using embedded system control,” in Proc. IEEE Int. Symp. Robotics and Intelligent Sensors (IRIS), vol. 76, Kuala Lumpur, Malaysia, pp. 126–133, 2015.
[8] R. Shukla, A. R., and J. Ramprabhakar, “Automatic self balancing robot for crop monitoring,” Research Article, 2023.
[9] S.-K. Kim and C. K. Ahn, “Self-tuning position-tracking controller for two-wheeled mobile balancing robots,” IEEE Trans. Circuits Syst. II: Express Briefs, vol. 66, pp. 2019–2026, June 2019.
[10] H. P. Santoso, H. Maghfiroh, J. S. Saputro, and M. M. M. Dinata, “Balancing robot navigation with virtual map and virtual sensor,” in Proc. Int. Conf. Radar, Antenna, Microwave, Electronics, and Telecommunications (ICRAMET), Surakarta, Indonesia, pp. 218–223, 2020.
[11] S. R. Garces et al., “Hybrid self-balancing and object tracking robot using artificial intelligence and machine vision,” in Proc. 2nd Novel Intelligent and Leading Emerging Sciences Conf. (NILES), pp. 264–269, 2020.
[12] N. T and P. K T, “PID Controller Based Two Wheeled Self Balancing Robot,” in 2021 5th International Conference on Trends in Electronics and Informatics (ICOEI), Tirunelveli, India, 2021, pp. 1–4.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY