Mono-Wheel Robot

* In fulfillment of the requirements for the ROCO507Z module

1st Yasas Chandrasena *10940925*

2nd Puranjana Wijayarathna *10942647*

3rd Sudeera Milakshan *10948185*

4th Tanvibahen Lakhani *10937718*

5th Himeth Sanjula *10749155*

6th Anjana Sreekumar *10916281*

7th Sangeetha Dhavamani *10912930*

***Abstract*—This project presents the development of a mono-wheel robot capable of self-balancing and step climbing. The robot demonstrates advanced functionalities through the integration of computer vision and sensor fusion techniques. The robot leverages the latest version of YOLO for object detection and combines data from an ESP32-S3 camera module and a HC-SR04 ultrasonic sensor to estimate object heights using a Kalman filter. Additionally, the implementation of object tracking and surface detection has been tested successfully on the robot's camera feed, although these features are not yet integrated into the final system. This innovative design showcases the potential for single-wheel robotic systems in applications requiring compact, agile, and intelligent mobility solutions.**

***Index Terms*—mono-wheel, self-balancing, YOLO, sensor fusion, vision based height detection**

I. Introduction

The design and development of autonomous robots have seen significant advancements, particularly in creating compact and efficient systems capable of performing complex tasks. This project introduces a single-wheel robot designed to achieve self-balancing and step-climbing capabilities, addressing the challenges of stability and mobility in constrained environments. The robot integrates advanced computer vision techniques, leveraging the latest YOLO framework for object detection, and employs sensor fusion for precise height estimation using an ESP32-S3 camera module and an ultrasonic sensor in conjunction with a Kalman filter. In addition to these core functionalities, object tracking and surface detection have been implemented and successfully tested on the robot's camera feed, although these features are not yet integrated into the final system. The development process involved four stages of design modifications, refined through trial and error, to optimize its performance and address technical challenges. Single-wheel robots hold potential for various applications, including last-mile delivery, reconnaissance in narrow or uneven terrains, inspection tasks in industrial settings, and personal mobility devices. By combining innovative design and robust functionality, this project demonstrates the versatility and potential of single-wheel robotic systems in addressing real-world challenges. The system was developed taking inspiration from the work of D. B. Tank, R. S. Jo, and H. S. Jo in their research titled "Dynamic Modelling and Control of Self-Balancing One-Legged Robot" [1].

II. Design Process

A. Mechanical Conceptual Design (Yasas)

To achieve self-balancing, the mechanism must be capable of balancing along two axes (x and y). The limbs of the mechanism are utilized to balance along the x axis. By actuating the limb, the center of gravity shifts along the x-axis, allowing self-balancing along the x axis. The reaction wheel balances the robot along the y-axis by adding counter-momentum to the system 1.

The actuation of the limb mechanism not only supports balancing along the x axis but also provides kinetic and potential energy for the robot to execute a jump, enabling it to ascend a step.

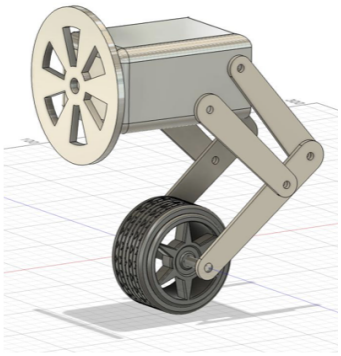

Fig. 1. Conceptual Design

B. Mechanical Detailed Design (Yasas)

1) Version 1: This first version of the detailed mechanical design encompasses all the necessary linkages and components of the conceptual design 2. However, there are several drawbacks to this mechanism. To jump the robot, a significant amount of current will be required. Furthermore, the impact

introduced by the jump will affect the self-balancing mechanism of the robot, since there is no control mechanism to control the forces of the robot when it is in the air.

Video: *Mechanical Design Version 1*

2) Version 2: To simplify the climbing process, a flexible wheel has been introduced (refer to Figure 3). This flexibility enables the wheel to adapt its shape in response to the step, facilitating the ascent. Since the jump is no longer expected, the connection linkages between the wheel and the robot's head have been simplified. However, the motion of the center of gravity along the x-axis is achieved through the combined motions of the servo motor positioned on the head and the driver motor. Nevertheless, the weight distribution of the system is not evenly distributed due to the weight of the servo motor. Furthermore, it is challenging to control the movement of the center of gravity along the x-axis due to the combined motions of two motors. In contrast to version 1, the disturbance caused by the reaction wheel still exists, as the reaction wheel is an integral component of the design.

Video: *Mechanical Design Version 2*

3) Version 3: To simplify the actuation of the center of gravity movement along the x-axis, the degree of freedom was reduced in the linkage between the head of the robot and the wheel (refer to Figure 4). Nevertheless, since the reaction wheel balances the robot along the y-axis, the disturbance caused by the reaction wheel still exists.

Video: *Mechanical Design Version 3*

4) Version 4: In this version, a planer mechanism is introduced as the balancing mechanism along the y-axis (refer to Figure 5), which will not introduce a significant disturbance to the system. Consequently, it will enable efficient system balancing. The balancing along the x-axis is achieved using the same method as version 3. In addition, the flexible wheel will support climbing the steps. Moreover, a servo motor controls the planer mechanism, while a geared DC motor drives the wheel.

Video: *Mechanical Design Version 4*

C. SolidWorks Simulation (Yasas)

The dynamic profiles were analyzed using DS SolidWorks Motion Analysis. The calculated weight of the robot from the CAD model is 205.69 grams. The robot was driven on a straight path, a curved path with a radius of 150 centimeters and a step with the height of 3 centimeters during the simulation (refer to Figure 6).

Video: *SolidWorks Simulation*

Based on the figure 7, the required torque for the servo motor is approximately 5 Nmm. Based on figure 8, the required torque for the drive motor ranges from 0 Nmm to 135 Nmm. Figures 9 to 12 provide a clear visualization of the torque requirements for the drive motor.

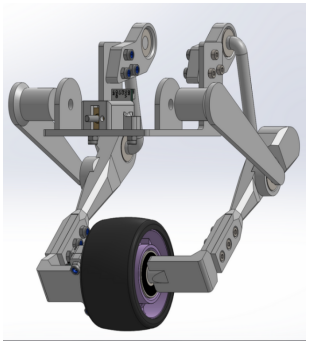

Fig. 2. Mechanical Design v1

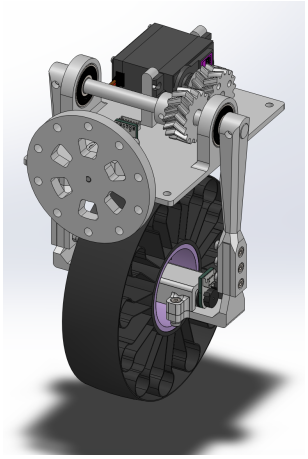

Fig. 3. Mechanical Design v2

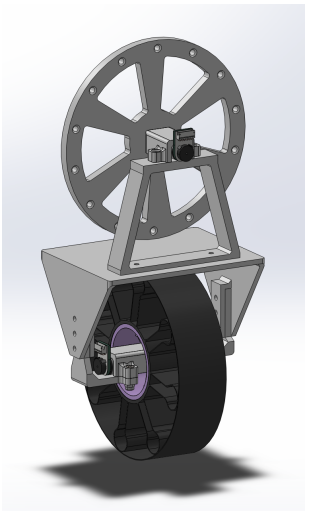

Fig. 4. Mechanical Design v3

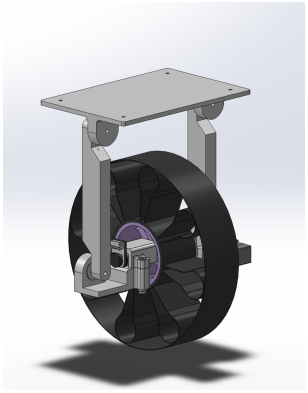

Fig. 5. Mechanical Design v4

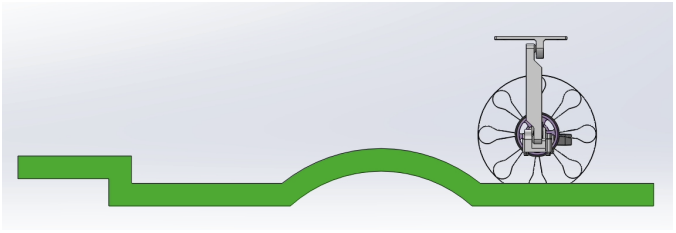

Fig. 6. SolidWorks Simulation

III. Implementation

A. Electronics Implementation (Tanvi & Himeth)

The electronic implementation of the mono-wheel robot integrated a variety of components, sensors, and controllers to achieve real-time balancing, navigation, and obstacle detection. The ESP32 microcontroller was chosen as the central processing unit due to its ability to manage critical functionalities such as balancing algorithms, sensor data processing, and motor control for stability and motion. Initially, the project utilized the Arduino Nano for this role; however, its limited computational power and inability to handle real-time balancing and sensor data fusion prompted a transition to the ESP32. This microcontroller, with its superior processing power and

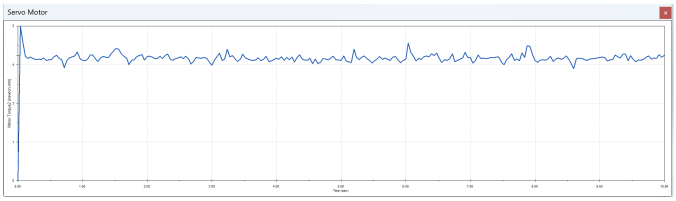

Fig. 7. Servo Motor Torque Requirement

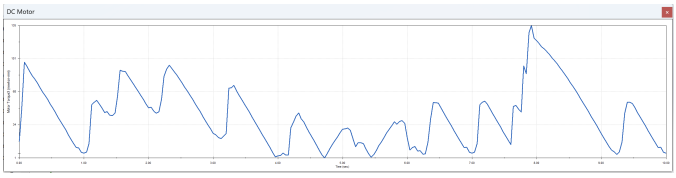

Fig. 8. DC Motor Torque Requirement

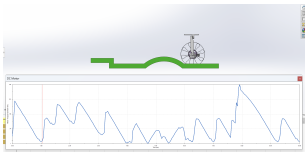

Fig. 9. DC Motor Torque Variation 1

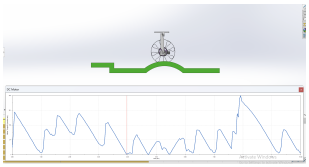

Fig. 10. DC Motor Torque Variation 2

Fig. 11. DC Motor Torque Variation 3

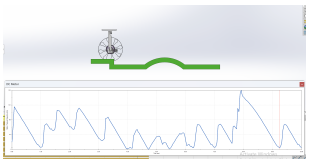

Fig. 12. DC Motor Torque Variation 4

integrated Wi-Fi and Bluetooth, proved to be a more robust solution for the robot's demanding requirements. To optimize system performance, a XIAO ESP32S3 microcontroller was included to handle proximity detection, specifically interfacing with the HC-SR04 ultrasonic sensor and the XIAO camera. This division of tasks improved the efficiency of the robot's operations. The motor control system incorporated an L293D motor driver, which powered the single motor responsible for the robot's motion. A logic-level converter ensured compatibility between the 3.3V logic of the ESP32 and the 5V requirements of other components, such as the ultrasonic sensor. Additionally, the MG90S servo motor provided stabilization and orientation adjustments, adding versatility to the robot's functionality.

1) Sensors (Tanvi & Himeth): The robot relied on an array of sensors to ensure balance, navigation, and environmental awareness. The MPU6050 IMU sensor played a key role in maintaining stability by providing real-time acceleration and gyroscopic data to determine the robot's orientation. During the project, raw data from the MPU6050 was collected and logged into CSV files for analysis. This data facilitated the refinement of stabilization algorithms, including complementary filtering and PID control, ensuring precise and effective balancing. The HC-SR04 ultrasonic sensor was utilized for proximity detection, measuring distances to nearby obstacles. This sensor worked in conjunction with the XIAO ESP32S3 microcontroller, which processed the data to avoid collisions and assist in navigation. Additionally, the XIAO camera captured visual data for advanced proximity sensing and object recognition. The integrated sensors enabled the robot to intelligently perceive and navigate its environment with enhanced balance and obstacle detection.

2) Electronic Circuit (Tanvi & Himeth): The electronic circuit design integrated all sensors, actuators, and controllers into a unified system. The ESP32 functioned as the central hub, interfacing with the MPU6050 IMU sensor via I2C for balance control, the L293D motor driver for motor operation, and the MG90S servo motor through a PWM pin for orientation adjustments. The XIAO ESP32S3 microcontroller, with its enhanced processing capabilities, was responsible for processing data from the HC-SR04 ultrasonic sensor and the XIAO camera, alleviating the computational load on the ESP32. A printed circuit board (PCB) was used to consolidate and organize the electronic components into a compact and durable assembly. The PCB design provided dedicated slots and pathways for connections between components, reducing wiring clutter and minimizing the risk of disconnections during operation. Soldering was employed to securely attach components to the PCB, ensuring reliable electrical connections and durability during prolonged usage. This approach enhanced signal integrity and reduced electrical noise, ensuring stable and reliable performance. To address voltage differences, a logic-level converter facilitated communication between the 3.3V ESP32 and 5V components like the ultrasonic sensor. The power distribution system was designed to deliver consistent voltage levels, with 3.3V and 5V supplies for sensors and controllers, and a high-current voltage source for the motor. The circuit layout minimized electrical noise and ensured reliable communication using I2C and PWM protocols, supporting seamless integration and efficient operation of the robot.

B. Firmware Implementation (Sudheera)

After developing the monowheel robot mechanism, designed through 3D modeling, conducting individual tests on each component, the primary PID-based control system was implemented. To enable the monowheel robot to self-balance, it is imperative to maintain both its vertical and horizontal stability. The implementation process comprised two primary components:

1) Developing a PID control system for the DC motor to ensure vertical stability.
2) Developing a PID control system for the servo motor to ensure horizontal stability.

The MPU6050 sensor was employed to measure the acceleration and gyroscope data along the X, Y, and Z axes during the robot's movements. This sensor generates gyro and

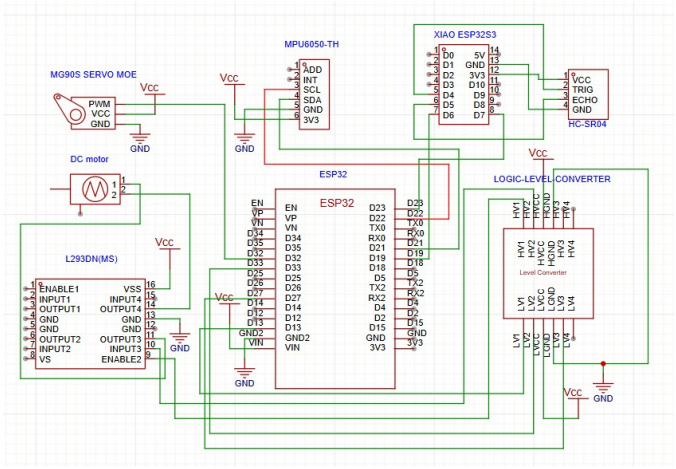

Fig. 13. Circuit Diagram

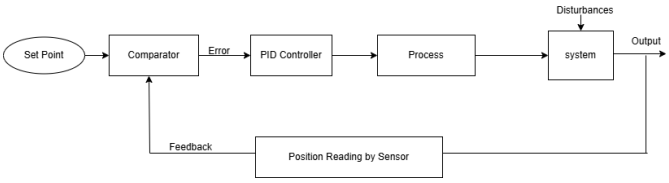

Fig. 14. Closed Loop PID diagram

acceleration data based on its position. Utilizing these data, the algorithm employs mathematical modeling to determine the robot's angle. The PID-based closed-loop diagram employed to achieve vertical balance for this robot is presented in Figure 14.

Initially, the monowheel robot undergoes calibration and establishes a reference angle as the setpoint, initialized at a zero-degree angle. During this calibration phase, the robot requires external support to maintain balance until the setpoint is securely defined. Subsequently, the system employs continuous feedback data from the sensor to monitor the robot's vertical axis. The robot continuously compares the setpoint with the actual current angle of the vertical axis, thereby defining the error as the difference between these two values. Utilizing a PID controller, the system maintains a steady effort to minimize the error value to zero. To accomplish this, the PID controller determines the direction and speed of the DC motor based on the error signal. The DC motor is directly coupled to the wheel, which plays a pivotal role in the robot's balancing mechanism. When the robot encounters external disturbances that cause it to move forward without the wheel rotating, the algorithm instructs the wheel to reverse its motion to restore balance along the vertical axis. Conversely, if the robot moves backward due to external forces, the algorithm commands the wheel to move forward to maintain equilibrium. The robot effectively counteracts external disturbances by generating forces through the wheel's reaction, thereby ensuring its vertical stability.

To develop a proportional-integral-derivative (PID) control system for the servo motor to ensure horizontal stability, a PID-based closed-loop control structure has been implemented. Initially, the robot undergoes calibration to establish the set point. The initial angle is designated as the reference angle, which is conventionally set to zero degrees. While the robot is in motion, the horizontal axis may shift. Without any control, such movement poses a challenge to maintaining the robot's balance solely through the vertical self-balancing method. To address this, a horizontal self-balancing technique has been implemented. This control system continuously strives to maintain the robot's horizontal angle at zero degrees. It receives continuous feedback on the horizontal angle from the sensor, compares it to the reference angle, and calculates the corresponding error. To ensure horizontal balance, a servo motor is mounted on the lower left side of the limb. When the robot's horizontal axis rotates counterclockwise due to external disturbances, the PID system computes the error and promptly commands the servo motor to rotate clockwise to restore balance. Conversely, when the horizontal axis rotates clockwise due to external disturbances, the servo motor responds by rotating counterclockwise. The servo motor's rotation is restricted to 35 degrees in both directions. This limitation is essential to prevent potential collisions with the wheels. Moreover, 35 degrees of rotation is sufficient to effectively maintain the robot's horizontal balance.

C. Computer Vision Implementation

1) Camera Sensor Integration (Puranjana): The integration of the camera module with the ESP32-S3 microcontroller in this system presents an efficient approach to facilitate real-time video streaming and supplementary sensor-based tasks. The hardware encompasses pivotal parameters such as pin assignments for data transmission, synchronization signals and clock input. This configuration also defines the resolution, pixel format, and memory allocation, utilizing PSRAM when feasible for enhanced image processing and additional frame buffering.

Post-initialization, adjustments are made to optimize image output for the specific sensor model. These adjustments include image flipping, brightness enhancement, and saturation adjustment to ensure quality and usability. For video streaming, the pixel format is configured as JPEG to reduce bandwidth, and the frame size is dynamically adjusted based on available resources to ensure smooth operation.

A lightweight web server is implemented to stream the camera feed over a Wi-Fi network. The ESP32-S3 manages the connection and provides a local IP address for client access. The system is designed with periodic distance measurements using external sensors. This modular and resource-efficient integration demonstrates the ESP32-S3's suitability for IoT applications requiring real-time imaging and multi-tasking capabilities.

2) Object Detection (Puranjana): The video stream is retrieved from a temporary server through a Flask application and processed using the YOLOv11m model developed by Ultralytics for real-time object detection. The YOLOv11m

model identifies objects in the stream, and for each detection, the bounding box coordinates (x, y, width, height) are extracted. These coordinates are utilized to draw visual bounding boxes around the detected objects on the video frames using the OpenCV library. Furthermore, the detection labels are overlaid onto the bounding boxes using OpenCV, facilitating clear identification of each detected object and enhancing the visualization.

Video Link: *YOLO Object Detection*

3) Object Height Detection: Sensor Fusion (Puranjana): To accurately perform height detection, the inferences from the YOLO model are filtered to retain detections with the highest confidence. A significant challenge in height detection using visual data, such as a camera feed, is the variability introduced by changes in the distance between the camera sensor and the object, or by the movement of the robot. This issue is addressed by integrating the visual detections with distance measurements obtained from the proximity sensor, employing sensor fusion techniques. This combination ensures more robust and reliable height detection by compensating for the variations in object distance and maintaining consistent measurements.

The sensor fusion process integrates visual data from the camera feed with proximity measurements to accurately estimate object height, with a significant emphasis on the utilization of the Kalman Filter to enhance the reliability of the results. Initially, pixel differences representing the visual height of an object are mapped to corresponding distances in centimeters using an interpolation method. Proximity sensors provide the distance between the sensor and the object, but these measurements can be noisy and prone to inaccuracies. The Kalman Filter is then employed to refine these distance estimates by predicting the state of the system (distance and velocity) and updating it based on new measurements. This filter effectively smoothens the data, reduces noise, and provides more stable and precise distance calculations. By combining the Kalman Filter with the visual and sensor data, the system ensures a robust height estimation process that mitigates errors caused by changes in object distance or movement, resulting in a more accurate and dependable sensor fusion outcome (The video links are provided in the experiments for this section).

4) Object Tracking (Sangeetha): This implementation of object tracking is designed for detecting object heights in the mono- wheel robot system to avoid redundant detections of the same object. The system integrates YOLOv8 Nano, a lightweight object detection model, with a custom tracking algorithm to ensure unique object identification across consecutive frames in real-time. The YOLOv8 model, loaded via the Ultralytics library, performs high-speed inference to detect objects in a video stream, providing bounding box coordinates, confidence scores, and class IDs. Detected objects are passed to the ObjectTracker, which maintains unique object identities by assigning each new detection a unique ID and tracking them based on their centroids. This ensures objects are identified consistently even as they move or reappear in the field of view. The system processes alternate video frames, resized to 640x480 resolution, to optimize performance and computational efficiency, making it suitable for resource-constrained systems like robotic platforms.

The tracking logic relies on calculating Euclidean distances between centroids of detected bounding boxes and previously tracked objects. By setting a maximum distance threshold (max_distance) and disappearance tolerance (max_disappeared), the tracker matches detections to existing objects or registers new ones. It uses an OrderedDict to maintain a mapping of object IDs to bounding boxes and class labels. To visualize the results, bounding boxes and class names are overlaid on the video stream using OpenCV, allowing real-time monitoring of the tracking process. The system's modular design, which combines the accuracy of YOLOv8 with a robust centroid-based tracking algorithm, ensures efficient detection and height measurement of objects in the field of view, avoiding duplicate detections and improving reliability for use in the momo-wheel robot. Robust error handling and efficient resource cleanup make the implementation practical for long-term deployment.

Video Links: *Object Tracking*

5) Surface Detection (Anjana): The dataset for surface detection was sourced from the publicly available Texture Design TD01 Natural Ground Textures dataset [2]. It comprises high-resolution images of 17 unique ground surface classes, including rocks, sand, grass, and leaves. The images were divided into training (80%) and testing (20%) subsets. The implementation was on Google Colab, utilizing its GPU support for accelerated training. Python was used with essential libraries like PyTorch, torchvision, and scikit-learn. The dataset was loaded, saved locally, and resized to 331x331 pixels. It was normalized and augmented with transformations to increase diversity and robustness. A PyTorch Dataset class was created for efficient handling. Each image was paired with its corresponding label.

The custom-designed CNN optimizes multi-class surface detection. It comprises three convolutional layers with ReLU activation and max-pooling layers, followed by fully connected layers for classification. Dropout regularization prevents overfitting. The Adam optimizer, with a learning rate scheduler that reduces the rate after 25 epochs, ensures stable convergence. Training and validation losses are monitored, and performance is quantified by accuracy, confusion matrices, and classification reports. The trained model, integrated with the robot, classifies surface textures in real-time by capturing frames from the camera feed and preprocessing them to match the CNN's input requirements.

Video Links: *Surface Detection*

D. Communication Network Implementation(Yasas and Puranjana)

A communication network based on HTTP facilitates the exchange of data between the robot controller and the computer responsible for processing computer vision algorithms. The primary core of the ESP32 houses an HTTP server, which manages the video stream captured by the camera.

Simultaneously, a dedicated server operating on the secondary core of the ESP32 processes GET requests responsible for retrieving decision data based on the processed video data from the computer for subsequent transmission to the robot controller. The HTTP server implemented on the computer handles POST requests responsible for retrieving ultrasonic data from the robot controller (refer to Figure 15).

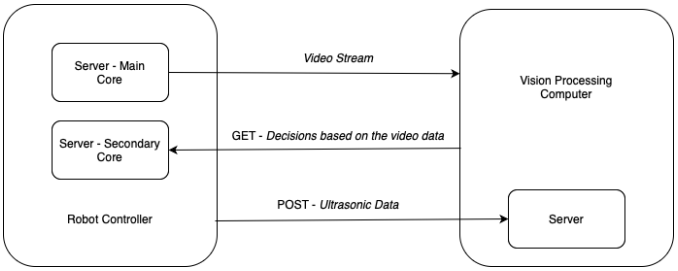

Fig. 15. Communication Network

IV. Experiments

1) Mechanical Design (Yasas): The initial robot design (Fig. 2) was not implemented due to its significant drawbacks. The second version (Fig. 3) was successfully implemented and tested for its motions. However, the servo motor mounted on the robot's head exhibited unexpected behavior. An unintended motion around the pivot point of the robot's head was detected due to the actuation of the servo motor. This issue was resolved by integrating the actuation of both the drive motor and the servo motor. Nevertheless, the weight of the servo motor caused instability when the robot's head moves closer to the wheel. Furthermore, the weight of the servo motor added additional stress to the drive motor. The flexible wheel performed as anticipated. Nevertheless, its rigidity hindered its ability to adapt its shape while moving over an object. The third version (Fig. 4) was implemented to address the previously identified issues. The wheel's flexibility was enhanced by reducing the internal links and increasing the wall thickness. This reduction in the degree of freedom facilitated the stabilization along the x-axis. Nevertheless, the additional distance caused by the reaction wheel obstructed the robot's efficient self-balancing capabilities. The planer mechanism introduced in the fourth version (Fig. 5) effectively addresses the y-axis balancing issue by eliminating the disturbances previously caused by the reaction wheel. The servo motor, which controls the planer mechanism, enables precise movements of the center of gravity along the y-axis. To reduce the stress on the electronics involved in balancing, the wheel width was increased. This enhancement resulted in a more stable balancing mechanism. The modification of the wheel's internal link shape and the subsequent increase in the number of its internal links contributed to reducing the stress on the drive motor. Simultaneously, it preserved the wheel's ability to adapt to varying object shapes while ascending a step.

2) Electronics (Tanvi & Himeth): During the early stages of development, Arduino Nano was used as the main controller. However, it faced significant challenges in handling complex tasks such as real-time sensor data processing and balancing. Its limited processing power and lack of wireless capabilities prompted the transition to the ESP32. The ESP32's dual-core architecture, high processing speed, and integrated Wi-Fi capabilities allowed for real-time computations, making it a robust choice for the robot's demanding requirements. For the motor driver also, the L298N was initially used, but it failed to provide sufficient torque because it could not supply adequate current to the motors. To address this limitation, the L9110S-H motor driver was tested as an alternative, offering improved current delivery. However, it was observed that the L9110S-H could not simultaneously control the motor's rotational direction and speed, which was a critical requirement for the robot's functionality. After further testing and evaluation, the L293D motor driver was ultimately selected. This driver met all operational needs, allowing precise control of both motor speed and direction while delivering adequate torque for smooth and efficient performance.

3) Firmware (Sudeera & Yasas): After implementing a PID-based self-balancing algorithm, challenges arose in efficiently working it. The DC motor mounted on the wheel needs initial torque to balance the monowheel robot, even without external disturbances. During design, it was assumed the motor's gear wheel could provide sufficient torque, but experimental results showed it was insufficient. Upon measuring the vertical axis angle, the PID controller decides the motor's rotational speed and direction. The motor reacts smoothly based on the analog output value. However, the motor doesn't react until it surpasses a certain level due to gear wheel reaction. To address this, the analog output range was changed from 0–255 to 60–255. This ensures a more stable and controlled response from the motor by adding 60 to the analog output value.

To achieve proper self-balancing, the PID controller's tuning required testing and observations. Adjusting the PID parameters showed changes in the robot's output, like oscillations, overshoot, and response time. The optimal values were found when the robot had minimal oscillations, overshoot, and the fastest response time. Initially, the Arduino Nano was planned for this algorithm, but testing showed running two PID loops was challenging and caused crashes due to its single core and 16 MHz frequency. The x-axis balance compared to its set value is shown in Figure 18. The ESP32 WROOM-32, with dual cores and a 240 MHz frequency, was chosen instead. The ESP32's much greater computational power made it a better choice.

In addition to the hardware modification, the PID calculations were implemented using a timer interrupt to enhance the accuracy of the output. The calculation is executed precisely 100 times per second with equal intervals. Moreover, it has been identified that the noise generated by the IMU for angle measurements adds a considerable unwanted stress to the system. A Kalman filter was introduced in order to filter the noise. As a result of the Kalman filter, the noise flow was reduced from 6 degrees to 0.3 degrees as mentioned in figure 16 and 17.

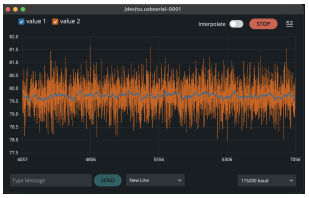

Fig. 16. Kalman Filter for Pitch

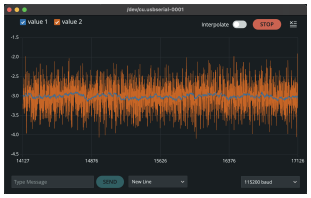

Fig. 17. Kalman Filter for Roll

Fig. 18. Balancing with Arduino Nano

Fig. 19. Balancing with ESP32

The balance achieved due to the modification of the controlled, time-interrupt implementation, and Kalman filter implementation is illustrated in Figure 19 compared to its set value. The error of approximately 15 degrees is reduced to approximately 0.6 degrees.

4) Computer Vision: Object detection and Height Calculation (Puranjana): The initial implementation of sensor fusion for height detection resulted in significant errors, with the actual object height of 10 cm being measured within a height range of 4 cm to 18 cm at a measurement distance of 25 cm to 50 cm. Notably, 25 cm was identified as the optimal starting point where the object was fully captured within the sensor's frame. To address this issue, the covariance and measurement noise parameters in the Kalman filter were adjusted through iterative testing. Initially set at 10 and 4, these parameters were refined to a value of 5 for both, which minimized the error margin to a range 2cm for the object's height. When the device was positioned at a fixed distance, it consistently provided accurate height measurements across various objects. Additionally, it was observed that the longer an object remained within the frame, the recorded values converged more closely to the actual height, demonstrating the effectiveness of the Kalman filter in refining measurements.

Video Links: *Single object Height Detection* , *Multiple object Height Detection*

5) Computer Vision: Object Tracking (Sangeetha): This object tracking algorithm initially combined the YOLOv8 Small (YOLOv8s) model with the Segment Anything Model (SAM) for accurate object detection and segmentation. However, this approach faced challenges due to dimensional mismatches between SAM-generated masks and input frames, causing errors during video processing. Resizing the masks resolved the issue but increased computational overhead, leading to crashes and delays. To address these issues, optimizations like skipping frames, using smaller frame resolutions, and reusing SAM masks were implemented. Despite these adjustments, the system's performance was insufficient for real-time applications, especially on a constrained robotic platform. Consequently, SAM was excluded, and the focus shifted to real-time object tracking using lightweight models.

The YOLOv8 Nano (YOLOv8n.pt) model was used for real-time object detection with a custom centroid-based tracking algorithm. This improved speed and reduced crashes, but tracking accuracy was affected by fast-moving objects. Multiple bounding boxes persisted in previous positions, leading to confusion. Optimizations were made to the tracking algorithm, including reducing the max_disappeared parameter and lowering the max_distance threshold. Frame skipping was introduced to balance tracking accuracy and computational efficiency. These adjustments enhanced the system's ability to predict object motion and maintain consistent tracking, even for fast-moving objects. However, the height detection component remains to be integrated, marking the next stage of development.

6) Computer Vision: Surface Detection (Anjana): Initially, the model showed high accuracy after the first training round but overfit due to the small dataset size. The training and validation accuracy graphs fluctuated, indicating poor generalization and unstable learning. The limited dataset size and diversity caused high-test accuracy, reflecting memorization rather than meaningful learning. To mitigate overfitting, data augmentation techniques increased dataset diversity, dropout layers reduced neuron reliance, and a dynamic learning rate scheduler stabilized training. Validation metrics were closely monitored, and early stopping prevented over-training. The model achieved a training accuracy of 92.0% and a validation accuracy of 88.5%, with a final test accuracy of 88.89% and a test loss of 0.239. The confusion matrix showed most classes accurately identified, with high precision, recall, and F1-scores. However, classes with limited data had slightly lower performance.

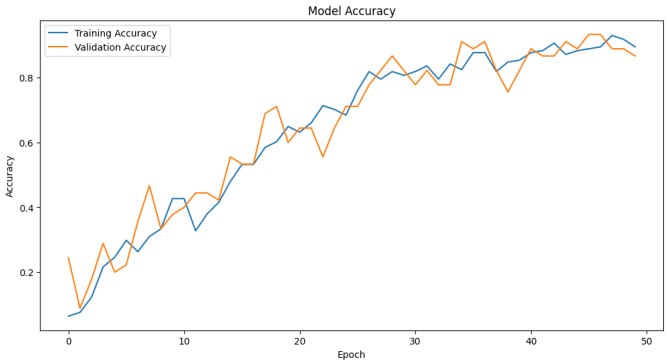

Fig. 20. Accuracy graph

The model demonstrated high accuracy on unseen dataset images but mixed results on unseen internet video footage, likely due to factors like lighting, resolution, and texture similarity. The real-time surface detection system was tested outdoors in a garden, where the robot's camera captured frames preprocessed and classified using CNN. Observations often revealed inaccurate predictions due to varying lighting

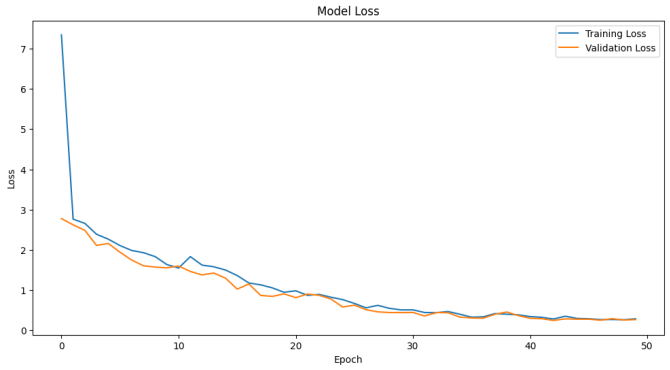

Fig. 21. Loss graph

conditions and environmental complexities, indicating the need for further improvement.

V. Discussion

The robot underwent extensive testing in diverse environments to assess its functionality. The planer mechanism appears to effectively balance the robot along the y-axis. However, the robot encounters challenges in balancing along the x-axis due to the dead band of the motor. Implementing a more responsive motor can effectively address this issue. The robot exhibits comparable performance in maintaining balance on both flat and inclined surfaces. The robot demonstrated successful step climbing capabilities up to a height of 1 cm without any difficulties. Nevertheless, as the height increases, the robot's ability to climb steps limited due to insufficient motor torque. Introducing a motor with enhanced torque will rectify this problem. Furthermore, testing the robot with a battery pack was not feasible due to its excessive weight. The battery's weight would increase the stress on the drive motor, leading to complications in self-balancing as the motor would lack torque.

Video: *Robot Testing*

The errors encountered in height detection during sensor fusion can arise from several factors. The primary source of error is the inconsistency in bounding boxes provided by YOLO. Even when the device remains stationary, the bounding box dimensions can fluctuate slightly, introducing noise into the system. Secondly, using a more precise ultrasonic sensor would improve height detection accuracy. Additionally, the ESP32-S3 camera utilized in the project has a fisheye lens, which can distort the pixel height measurements derived from the bounding box, adding further noise; replacing this with a higher-quality camera module and fish-eye lens correction in the software implementation would mitigate this issue. Finally, enhancing the Kalman filter's implementation could further refine the accuracy of the detection process.

VI. Conclusion

The robot exhibits the capability to ascend heights of 1 centimeter or less, which can be augmented with the utilization of more powerful drive motors. The planner mechanism effectively supports the self-balancing system, enabling the robot to maintain equilibrium autonomously. Nevertheless, there exists a deadband where the self-balancing functionality encounters limitations; this issue can be mitigated by incorporating more responsive motors. While the current wheel design is satisfactory, further testing with higher-powered motors may unveil opportunities for enhancement. Object detection is functioning at an acceptable level, but height detection through sensor fusion can benefit from the incorporation of more precise sensors and advanced algorithms to augment accuracy.

Acknowledgment

We would like to express our sincere gratitude to Dr. Mohammed Diab for his inspiring guidance and unwavering support throughout the ROCO507Z module. His ability to make the coursework both engaging and thought-provoking has been instrumental in motivating us and enriching our understanding of the subject.

We also extend our heartfelt thanks to Mr. James Rogers and Mr. George Seymour for their invaluable technical assistance and guidance during the development of the Monowheel Robot project. Their expertise and support were critical in helping us overcome challenges and successfully fulfill this requirement for our master's course.

References

[1] D. B. Tank, R. S. Jo, and H. S. Jo, "Dynamic Modelling and Control of Self-Balancing One-Legged Robot," in *2018 IEEE International Conference on Automatic Control and Intelligent Systems (I2CACIS)*, Shah Alam, Malaysia, 2018, pp. 19-23, doi: 10.1109/I2CACIS.2018.8603694. Keywords: Legged locomotion; Wheels; Mathematical model; Prototypes; Gears; One-legged robot; Euler-Lagrange mechanics; Self-balancing system; Limb balancing; Reaction wheel.

[2] Hugging Face *td01_natural-ground-textures* dataset. Available at https://huggingface.co/datasets/texturedesign/td01_natural-ground-textures.

Appendices

GitHub repository: *Monowheel Robot GitHub Repository*

Full Video: *Monowheel Robot Video*