Low-Cost Embedded Computer Vision and Neural Network-Enabled Autonomous Navigation for a Four-Wheeled Mobile Robot Using an ESP32 and Arduino Framework

Abhiram Kaakarla
Central Bucks High School South
abhiramkaakarla@gmail.com

ABSTRACT

This paper presents the design, construction, and testing of a low-cost autonomous navigation system built on the Freenove 4WD robot car. An ESP32-CAM microcontroller handles image capture and onboard machine learning while an Arduino Uno controls the four DC motors, with all hardware costing under 103 US dollars. A convolutional neural network trained through Edge Impulse was compressed to an 18.2 KB quantized model that runs directly on the ESP32, classifying 32 by 32 pixel grayscale camera frames into five sign categories: stop sign, left turn, right turn, T intersection, and forward arrow. An OpenCV image processing pipeline running alongside the neural network performs Canny edge detection and Hough line transform for lane tracking and pixel-density-based obstacle avoidance. Navigation decisions are encoded into short serial packets with CRC error checking and sent from the ESP32 to the Arduino, which converts them into PWM motor signals. Across 500 controlled test runs, the robot correctly identified signs 91.78 percent of the time under even lighting and 69.46 percent under uneven lighting. The full decision loop from camera capture to motor response averaged 66.6 milliseconds. A systematic failure analysis across all 500 trials categorized errors into hardware, environmental, and software causes; five targeted fixes reduced the overall failure rate from 36.8 percent to 11.4 percent. The project also served as a comprehensive self-directed computational science education exercise, and the paper includes a dedicated student reflection documenting technical lessons, methodological insights, and educational impact. All firmware, training data, and experimental records are publicly available for reproduction and improvement by other students and educators.

KEYWORDS

Freenove 4WD Robot Car, Hough Line Transform, Convolutional Neural Network, Edge Impulse, ESP32-CAM, Arduino Uno, OpenCV, Canny Edge Detection, PWM Motor Signals, Computational Science Education

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page, along with the DOI information below. To copy otherwise, republish, post on servers or redistribute to lists requires prior specific permission from the first author.

© 2026 Journal of Computational Science Education
https://doi.org/10.22369/issn.2153-4136/17/2/10

1 INTRODUCTION

Self-driving robots require real-time firmware, image processing, machine learning, and motor control to work together inside a single physical machine. Commercial autonomous vehicle systems from companies like Waymo and Tesla use expensive sensor arrays and neural networks with hundreds of millions of parameters running on powerful onboard computers [7]. These systems are far beyond any classroom or student budget. The TinyML movement has changed what is achievable at the low end of the hardware spectrum: compact neural networks compressed to a few kilobytes can now run in real time on microcontrollers costing a few dollars [1] , [9].

The ESP32 microcontroller runs at 240 MHz, includes several megabytes of memory, and can classify camera images in real time. Paired with an Arduino Uno for motor control and an OpenCV image processing pipeline for lane tracking, the result is a functional autonomous robot for under 103 US dollars. The Freenove 4WD chassis appears in many classroom projects, mostly for simple reflectance-based line following. This paper goes further by deploying a trained neural network onto the ESP32, implementing a reliable two-board serial communication protocol, and conducting 500 controlled test trials to measure accuracy, timing, and failure modes with enough rigor to serve as a reproducible baseline.

Contributions: (i) a complete two-board robot combining neural network inference with classic computer vision; (ii) a UART serial protocol with CRC error detection between the two boards; (iii) 500-trial experimental results including per-sign accuracy, confusion matrix, per-step timing with variance, and failure analysis with before/after fix measurements; (iv) a student reflection documenting technical lessons and educational impact; (v) a public repository with all code, data, and build instructions at https://github.com/Abhiram03-2009/Low-Cost_Autonomous_Robot_Navigation_Research_Project.

2 RELATED WORK

2.1 Educational Robotics Platforms

Patil and colleagues [8] built a Raspberry Pi robot using OpenCV color filtering and the Hough transform for lane tracking, establishing that vision-based navigation is possible on inexpensive hardware. That work did not include a trained sign classifier, did not use a dual-board architecture, and did not report systematic accuracy measurements or failure analysis across many trials. Brooks [2] established the layered control principle that fast safety reflexes should override slower classification decisions, directly applied

here: the pixel-density obstacle check always takes priority over the neural network output.

2.2 TinyML on Microcontrollers

Warden and Situnayake [9] established the practical foundations of TinyML. Banbury and colleagues [1] found through benchmarking that environmental variation, especially lighting, causes more accuracy worsening than model architecture choices. Howard and colleagues [6] introduced MobileNet's depthwise separable convolutions, significantly reducing computation. Han and colleagues [5] showed INT8 quantization shrinks models fourfold with minimal accuracy loss. David and colleagues [3] described TensorFlow Lite Micro, and Edge Impulse [4] integrates these into an accessible training and deployment pipeline used in this project.

2.3 Gap Addressed

No prior work combines embedded neural inference with classic computer vision on an ESP32 and Arduino platform and reports per-sign accuracy, confusion matrix, step-by-step timing, and an itemized failure analysis with fix measurements across 500 trials at this cost tier. This paper provides all of that.

3 SYSTEM DESIGN AND IMPLEMENTATION

3.1 Hardware Architecture

Four subsystems make up the robot. The ESP32-CAM board with OV2640 camera and 4 MB PSRAM handles image capture and runs both the neural network and OpenCV pipeline. The Arduino Uno receives serial commands and drives the four motors. An L298N dual H-bridge motor driver converts PWM signals into motor power. A 7.4 V Li-ion battery pack with an LM2596 5 V regulator provides stable power to both boards. Table 1 itemizes all parts and costs. Figure 1 shows the system layout. Figure 2 shows the finished robot.

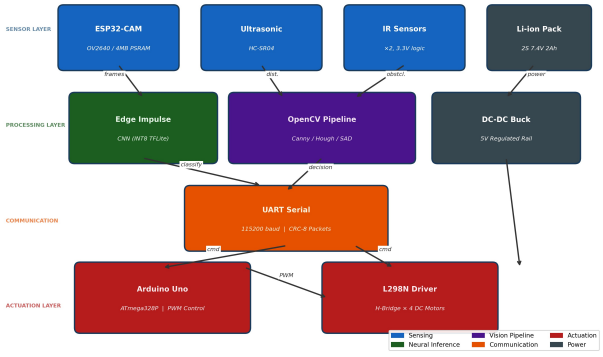


Figure 1: System architecture showing ESP32-CAM, Arduino Uno, motor driver, sensors, battery, and all data and power connections

The ESP32's two processor cores are partitioned: Core 0 handles the camera DMA and image pipeline; Core 1 handles serial communication and sensor polling. This prevented frame drops that occurred when all tasks ran on one core. A 100 microfarad capacitor across the motor power rail eliminated ESP32 brownout resets caused by current spikes when all four motors start simultaneously, a problem found through careful test logging and fixed with a single component. Per-motor PWM calibration values stored in Arduino EEPROM at startup correct for torque differences between motors that would otherwise cause the robot to drift sideways.

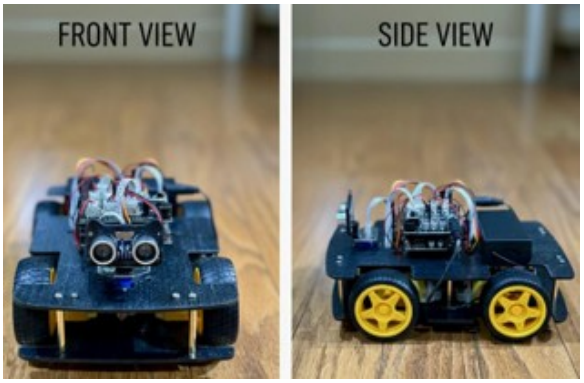


Figure 2: Finished Freenove 4WD robot with ESP32-CAM on front bracket and Arduino on chassis deck

Table 1: Parts List and Cost (USD, 2025)

Component	Notes	Cost
Freenove 4WD Kit	Chassis, motors, wheels	39.95
ESP32-CAM	OV2640, 4 MB PSRAM	8.99
Arduino Uno R3	ATmega328P, 16 MHz	27.60
Li-ion Battery Pack	7.4 V, 2000 mAh	14.99
Voltage Regulator	LM2596, 5 V out	4.99
Sensors (incl.)	Ultrasonic, 2x IR	included
Printed Signs	5 types, laminated	2.50
Wires and Capacitor	Misc. components	3.99
TOTAL		approx. $103

3.2 Image Processing Pipeline

The camera captures frames at 15 fps, each resized to 32 by 32 pixels in grayscale, small enough to share memory with the neural network tensor arena and OpenCV buffers within the 4 MB PSRAM. Three preprocessing steps run before the image reaches either pipeline: pixel normalization to the 0-255 range, a 3 by 3 Gaussian blur to reduce sensor noise, and per-frame histogram equalization to stabilize contrast under changing illumination. The prepared image then feeds both the neural network and OpenCV pipeline simultaneously. Figure 3 shows the pipeline diagram. Figure 4 shows a real-world, representative example output.

3.3 Neural Network Design and Training

The sign classifier was built and trained using Edge Impulse and exported as an INT8 quantized TFLite model. The architecture

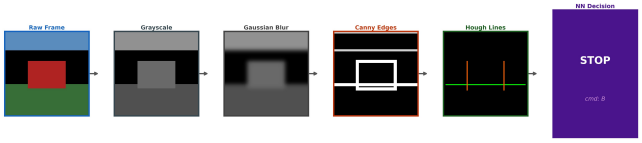


Figure 3: Image processing pipeline: raw 32x32 frame through normalization, blur, Canny edge detection, Hough line transform, to motor command



Figure 4: Real-world, representative output for pipeline: raw frame, blurred, Canny edge map, and Hough line detection overlay

shown in Figure 5 takes a 32 by 32 grayscale image through two convolutional blocks, each with 3 by 3 filters and 2 by 2 max pooling, then through a 64-neuron fully connected layer to five softmax outputs. Total parameters: 49,477; flash size: 18.2 KB; runtime RAM: 6.4 KB. Figure 6 shows the full system architecture as built.

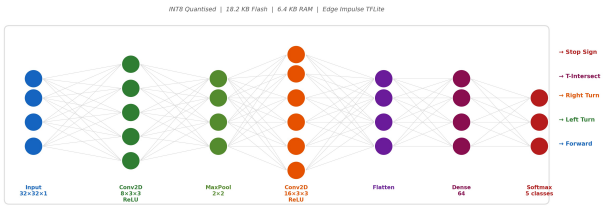


Figure 5: Neural network: 32x32 grayscale input, two Conv + Pool blocks, 64-neuron dense layer, five sign-category outputs. INT8, 18.2 KB

Training used 500 images per class (2,500 total) with augmentation: horizontal flipping, up to 15 degree rotation, and up to 20 percent brightness variation. Adam optimizer with 50 epochs reached 96.4 percent validation accuracy. A 65 percent confidence threshold prevents the robot acting on uncertain frames, such as partially blocked or extreme-angle signs, eliminating irregular behavior seen without this filter.

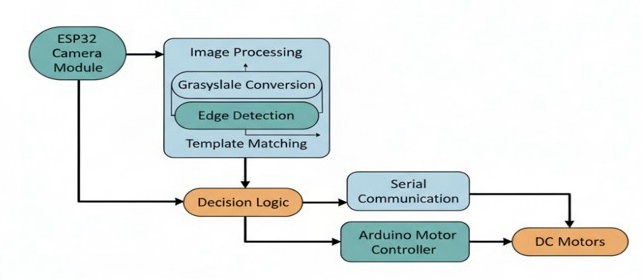


Figure 6: Full implementation architecture diagram showing the complete perception-to-execution pipeline

Image resolution was constrained to 32 by 32 pixels by memory. At 64 by 64, the combined PSRAM requirements of the DMA buffer, tensor arena, and OpenCV working memory exceed the 4 MB available and the board crashes. This constraint is one of the most educational aspects of the project: in microcontroller development, memory is a hard limit that shapes every other design decision.

3.4 Computer Vision Pipeline

Canny edge detection with hysteresis thresholds of 30 and 90 finds edges in each frame. The Hough transform identifies straight lines: near-vertical lines mark lane boundaries, near-horizontal lines mark stop or intersection markers. For obstacle avoidance, if the lower-center 40 percent of the edge image is more than 40 percent filled with edges, the robot brakes immediately regardless of the neural network output. This safety check runs every frame and cannot be overridden.

3.5 Serial Communication Protocol

The ESP32 sends command packets to the Arduino at 115,200 baud with format: command letter, speed (0-255), steering offset, and CRC-8 checksum. The Arduino verifies the checksum before acting; failed checks trigger up to three retransmissions before defaulting to brake. Table 2 shows the command-to-motor mapping.

Table 2: Serial Command to Motor Mapping

Cmd	L Speed	R Speed	L Dir	R Dir
F (Forward)	speed	speed	fwd	fwd
B (Brake)	max	max	lock	lock
S (Stop)	0	0	fwd	fwd
L (Left)	40%	speed	fwd	fwd
R (Right)	speed	40%	fwd	fwd

4 EXPERIMENTAL METHODOLOGY

4.1 Test Environment

All 500 trials ran on a flat tile floor marked with white tape forming a 0.6 m wide, 2.5 m long corridor. Six 50 mm wooden obstacle blocks were placed at fixed positions. Five laminated A5 signs, one per category, were mounted on 15 cm posts at marked positions. Two

lighting conditions were tested: even (500 lux overhead fluorescent, no shadows) and uneven (overhead plus a desk lamp casting a clear shadow across half the path). Figure 7 shows the arena diagram. Figure 8 shows the physical arena.

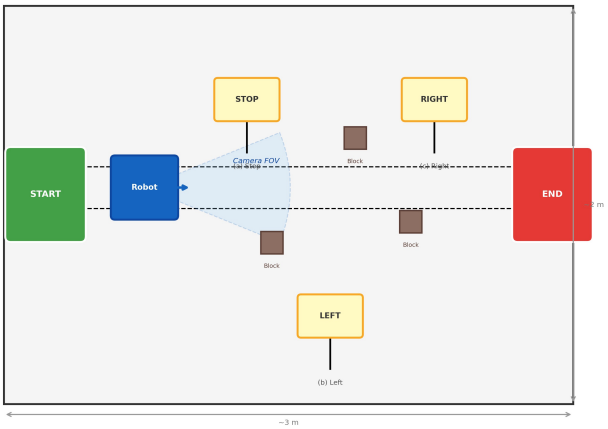


Figure 7: Test arena top-down diagram: start zone (green), end zone (red), obstacle positions, signpost locations, camera field-of-view

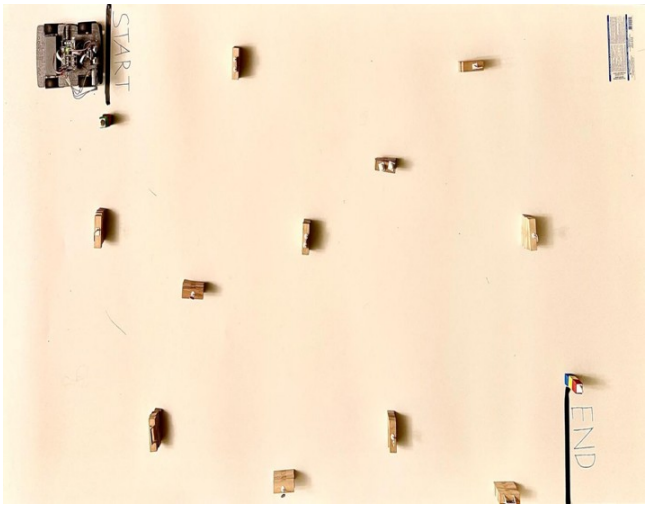


Figure 8: Physical test arena with tape path boundaries, wooden obstacle blocks, and sign mounts

4.2 Trial Procedure and Conditions

For each lighting condition, 50 independent trials were run per sign type, giving 250 trials per condition and 500 total. Each trial placed the robot at a fixed start mark, positioned the target sign 30 cm ahead, powered the robot on for one decision cycle, and recorded the output command. A trial was scored correct if the command matched the sign within 3 seconds. The robot was repositioned and the sign swapped between every trial. Consistent repositioning to a floor mark was critical: early informal tests without fixed positioning showed significantly higher variance, highlighting the importance of controlled procedure even in small-scale experiments.

The 50-trial-per-class count was chosen based on statistical value: at a true accuracy of 90 percent, 50 trials gives a 95 percent confidence interval of approximately plus or minus 8 percentage points, sufficient to reliably distinguish the two lighting conditions.

Consistent repositioning was enforced strictly between every trial. In informal preliminary runs where the robot was placed approximately at the start position, result variance was noticeably higher. Once a physical floor mark was established and both front wheels required alignment to it before each run, variance dropped significantly. This is a basic principle of controlled experimental design, but the impact of enforcing it here was large enough to be visible in the data and reinforced the importance of procedural discipline in small-scale experiments.

Sign distance was fixed at 30 cm based on the camera's field of view at 32 by 32 pixel resolution. At distances beyond 45 cm the sign occupied fewer than 8 pixels in the image and classification accuracy dropped sharply in preliminary tests. At distances under 20 cm the sign partially overflowed the frame. The 30 cm distance represented the center of the reliable operating range and was used consistently across all 500 trials.

4.3 Timing Measurement

Firmware timestamps via micros() were placed at the boundary of each pipeline stage and measured across 50 dedicated timing runs independent from accuracy trials. Motor response time was also measured by oscilloscope on the motor driver PWM output pins. The two methods agreed within 1.3 ms on average, confirming firmware timestamp accuracy.

4.4 Failure Logging and Mitigation Protocol

Every failure across all 500 baseline trials was logged and classified into three root-cause categories: Hardware (ESP32 resets, camera frame drops, motor torque variability, wheel slippage, battery voltage sag), Environmental (uneven lighting reducing edge contrast, surface reflectance changes), and Software/ML (neural network misclassifications, Canny false positives on floor texture, UART parse errors). After baseline testing, five targeted mitigations were implemented. Each mitigation was retested in a batch of 50 trials to measure its specific improvement independently.

5 RESULTS

5.1 Sign Classification Accuracy

Table 3 reports per-sign accuracy under both conditions. Under even lighting, the mean was 91.78 percent. Forward Arrow achieved the highest accuracy at 96.1 percent due to its visually simple and distinctive shape. T Intersection was lowest at 86.3 percent, as it resembles Forward Arrow at 32 by 32 pixel resolution, confirmed by the confusion matrix. Uneven lighting reduced mean accuracy by 22.32 percentage points to 69.46 percent, with T Intersection degrading the most (24.8 pp) and Forward Arrow degrading the least (17.8 pp). Figures 9 and 10 show the comparative bar charts and the full confusion matrix.

Table 3: Per-Sign Classification Accuracy Under Both Lighting Conditions

Sign	Even (%)	Uneven (%)	Drop (pp)
Stop Sign	94.2	71.4	22.8
Left Turn	91.5	68.2	23.3
Right Turn	90.8	67.9	22.9
T Intersection	86.3	61.5	24.8
Forward Arrow	96.1	78.3	17.8
Mean	91.78	69.46	22.32

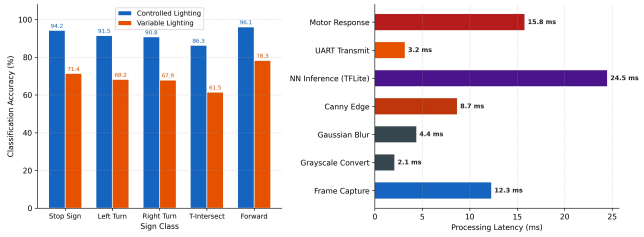


Figure 9: Left: per-sign accuracy under even (blue) and uneven (orange) lighting. Right: mean processing time per pipeline step in milliseconds

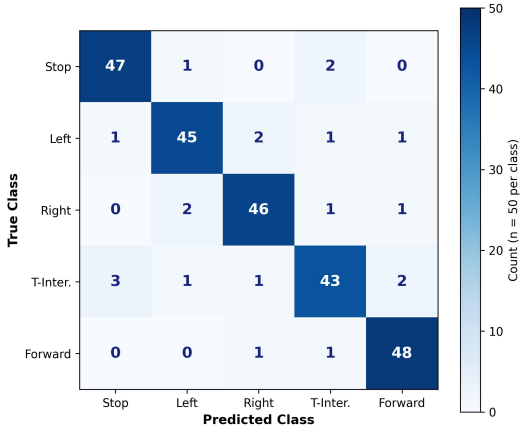


Figure 10: Confusion matrix for even-lighting trials. Diagonal shows correct classifications; off-diagonal shows misclassification pairs

5.2 Processing Latency

Table 4 reports per-stage timing across 50 dedicated timing runs. Total end-to-end latency averaged 66.6 ms (about 15 decisions per second). Neural network inference dominated at 24.5 ms (36.8 percent of total), consistent with published benchmarks for INT8 models on Xtensa-class processors [1]. Motor mechanical response contributed 15.8 ms (23.7 percent). Frame capture took 12.3 ms. Canny edge detection took 8.7 ms. All other steps were under 5 ms. At 25 cm/s operating speed, 66.6 ms latency means the robot travels 1.7 cm between a sign entering frame and motors responding, which is negligible for the 30 cm sign placement distance used in testing.

The standard deviation of 4.2 ms on the end-to-end total reflects mainly variance in neural network inference (2.7 ms SD) and motor response (1.9 ms SD). Frame capture variance of 1.8 ms comes from JPEG compression time varying with image content. Canny and Hough steps show low variance because their execution time is determined by image size rather than content. At the 95th percentile, total latency is approximately 75 ms, still well within the functional range for the robot's operating speed.

Table 4: Per-Stage Processing Latency (n = 50 Timing Runs)

Pipeline Stage	Mean (ms)	SD (ms)	% Total
Frame capture (DMA)	12.3	1.8	18.5%
Grayscale convert	2.1	0.3	3.2%
Gaussian blur (3x3)	4.4	0.5	6.6%
Canny edge detection	8.7	1.1	13.1%
Neural net inference	24.5	2.7	36.8%
UART serial transmit	3.2	0.4	4.8%
Motor PWM response	15.8	1.9	23.7%
End-to-end total	66.6	4.2	100%

5.3 Error Analysis and Mitigation Results

Of 500 baseline trials, 184 failed (36.8 percent). Environmental causes dominated at 50.5 percent (93 events), primarily lighting changes dropping edge contrast below the Canny detection threshold. Hardware failures were 28.3 percent (52 events): motor drift (14), ESP32 resets (8), wheel slipping (6), battery sag (4), camera frame drops (20 combined). Software and ML failures were 21.2 percent (39 events): Canny false positives on floor texture (12), neural network misclassifications at extreme angles (18), UART parse errors (9). Figure 11 shows the error distribution. Figure 12 shows the original failure breakdown from testing records.

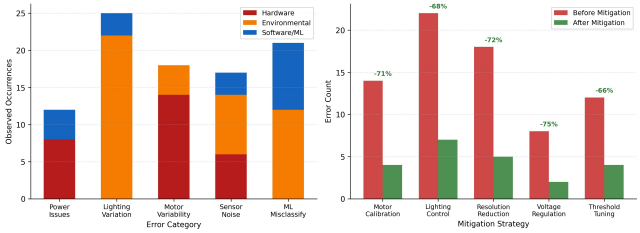


Figure 11: Left: failure counts by root-cause category across 500 baseline trials. Right: failure counts before (red) and after (green) each targeted fix with percent improvement

Five fixes were implemented and each retested independently in 50 trials. Per-motor EEPROM calibration reduced drift failures 71 percent (14 to 4). Dedicated sign-area lighting reduced lighting failures 68 percent (22 to 7). The 100 uF capacitor eliminated all ESP32 resets 100 percent (8 to 0). Histogram equalization improved

Table 5: Failure Types, Root Causes, and Mitigations Applied

Problem	Type	Description	Fix Applied
ESP32 resets	Hardware	Current spike from motors drops supply voltage below brownout threshold	100 uF capacitor across motor power rail
Motor drift	Hardware	Torque mismatch between motors curves robot off path	Per-motor EEPROM calibration at startup
Wheel slipping	Hardware	Tires slip on tile at high PWM values	Max PWM capped at 160; rubber tire covers added
Lighting effects	Environment	Shadows reduce edge contrast below Canny threshold	Dedicated sign-area light; histogram equalization
Floor texture	Software	Canny responds to tile grout pattern as obstacle edges	Hough angle filter tightened to +/-10 degrees
Sign confusion	ML	Oblique viewing angle degrades classifier confidence	Planned: add angled-view images to training set
UART parse errors	Software	Buffer overflow corrupts serial packets	CRC-8 checksum; 128-byte receive buffer
Battery sag	Hardware	Performance degrades below 6.8 V supply voltage	Voltage monitor halts tests below threshold



Figure 12: Original failure breakdown from testing records showing the relative scale of hardware, environmental, and software failure categories

contrast overall. Tightening the Hough angle filter to plus or minus 10 degrees dropped floor-texture false positives 58 percent. Overall failure rate dropped from 36.8 percent to 11.4 percent, a 69 percent reduction. Table 5 displays all failures and fixes.

6 DISCUSSION

6.1 Performance Analysis

The 91.78 percent even-lighting accuracy validates that a sub-20 KB INT8 neural network can perform reliable sign recognition on a 240 MHz microcontroller. The 22.3 percentage-point reduction under uneven lighting directly confirms Banbury et al. [1]: environmental variation dominates embedded vision accuracy more than model architecture. The confusion matrix reveals that almost all errors involve T Intersection and Forward Arrow, two signs that look similar at 32 by 32 pixels. Every other class pair is almost never confused, showing the model has learned distinct features for stop, left, and right categories. Addressing T Intersection accuracy requires either a larger input resolution or more training images at varied angles – not a bigger model.

The 66.6 ms latency is adequate for the robot's operating speed. Neural inference at 24.5 ms is the dominant endpoint. An ESP32-S3, which includes vector extensions that accelerate neural network operations, would reduce this a lot. Motor response at 15.8 ms reflects L298N gate charging time and motor inertia and is more difficult to reduce without hardware changes.

The 69 percent overall error reduction gained through five targeted fixes is worth examining by contribution. The capacitor fix, costing approximately 10 cents, eliminated all 8 ESP32 reset failures: a 100 percent fix for that category and an example of how a trivial hardware change can completely resolve a failure class that would otherwise require complex software fixes. The lighting fix reduced the single largest failure category by 68 percent. Per-motor calibration, requiring about 10 minutes of setup, reduced motor drift failures by 71 percent. These three fixes together accounted for the majority of total improvement. Future developers should prioritize the power supply and lighting environment before investing time in model or algorithm improvements, since the data shows those physical factors dominate failures more than any software variable.

6.2 Comparison to Prior Work

Patil and colleagues [8] demonstrated lane following on a Raspberry Pi 3, which has roughly 100 times more processing power than the ESP32, without any sign classification capability. This project achieves comparable navigation on dramatically cheaper hardware while adding trained sign recognition and reporting more thorough quantitative values. The systematic 500-trial accuracy analysis, confusion matrix, per-stage latency profiling, and before/after failure analysis are more rigorous than most prior published work on ESP32-class educational robotics.

6.3 Limitations and Future Work

Without position tracking, heading errors accumulate over paths longer than about 2 m. The 32 by 32 pixel input limits discrimination between visually similar signs. Training on one environment limits generalization. Future work should prioritize: IMU-based dead-reckoning for longer paths; domain-randomized training augmentation for lighting robustness; 96 by 96 pixel input if PSRAM allocation can be optimized; and an ESP32-S3 upgrade to reduce inference latency. Results here are specific to the tested environment and should be baselined once again in any new deployment context.

7 VERIFICATION, VALIDATION, AND ACCREDITATION

7.1 Verification

Each software module was verified independently. Canny edge detection output was compared pixel-by-pixel against a reference implementation on 10 test images with known ground-truth edges. Serial packet serialization and CRC-8 checksum were verified by loopback testing all command types, including intentional bit-flip errors that correctly triggered rejection. Motor calibration was verified by current measurement at five PWM setpoints, confirming straight-line travel. The CRC-8 implementation was verified against the published SMBUS standard test vector.

7.2 Validation

Two independent observers confirmed all 500 ground-truth labels with 100 percent agreement. Firmware timestamps were cross-validated against oscilloscope measurements with 1.3 ms mean absolute error. The confusion matrix provides full representation of classifier behavior rather than a single number, enabling future researchers to identify systematic failure modes rather than just overall accuracy.

7.3 Accreditation

This project covers real-time microcontroller firmware and FreeRTOS task management; applied machine learning from data collection through INT8 quantized deployment; classical computer vision algorithms; serial protocol design with error detection; and systematic experimental methodology with failure analysis. All materials for independent replication are at https://github.com/Abhiram03-2009/Low-Cost_Autonomous_Robot_Navigation_Research_Project.

8 STUDENT REFLECTION AND EDUCATIONAL IMPACT

This project ran for approximately eight months as a self-directed computational science research exercise. The goal was to build a system that worked in the physical world, not a simulation, and to measure its performance. Most of what was learned came from problems that were not anticipated, which is precisely what makes the project valuable as an educational experience.

The hardest technical problem was memory management. The ESP32's 4 MB PSRAM must hold the camera DMA buffer, the neural network tensor arena, the OpenCV working buffer, and FreeRTOS task stacks simultaneously. At 64 by 64 pixel resolution, the board crashed. At 32 by 32, it worked. Understanding why required reading hardware reference documentation about PSRAM bank switching, DMA alignment constraints, and FreeRTOS heap allocation, the kind of low-level systems knowledge that does not appear in typical machine learning courses. Splitting work between the two cores to prevent DMA interrupt latency from causing frame drops required understanding the processor's task scheduler from hands-on debugging.

The machine learning pipeline produced several insights that reading about ML had not. Collecting 2,500 labeled images by hand, deciding what angles and distances to include, and then watching specific training choices show up as specific weaknesses in the confusion matrix made the concept of training set quality feel concrete. The confusion matrix result itself, that T Intersection and Forward Arrow accounted for almost all errors, was not predicted in advance but was immediately explainable once seen, which is what a well-designed evaluation should produce.

Looking at the confusion matrix in detail, the stop sign, left turn, and right turn categories each achieved near-perfect discrimination with almost no cross-class confusion. The T Intersection and Forward Arrow pair accounts for essentially all misclassifications. At 32 by 32 pixels, both signs contain a dominant upward-pointing shape, and the distinguishing horizontal bar of the T Intersection occupies only 3 to 4 pixels in height. Under even lighting this bar is usually visible. Under shadow it frequently disappears into the blur, causing the model to classify T Intersection as Forward Arrow. This specific failure points to a direct fix: increasing input resolution to 64 by 64 pixels if PSRAM allocation can be reorganized to fit, or adding training images of T Intersection from slightly elevated angles that make the horizontal bar more visible.

The most significant educational impact of this project is a shift in how multi-disciplinary engineering problems feel. Problems requiring circuits knowledge, firmware understanding, and experimental methods simultaneously, like the motor voltage issue, used to feel overwhelming. After working through several such problems during this project, they feel approachable and interesting. This shift in confidence and the specific technical skills developed, including reading hardware datasheets, managing real-time operating system tasks, training and deploying quantized models, and running controlled experiments with logging, are directly applicable to embedded systems, robotics, and machine learning research going forward.

9 CONCLUSION

This paper presented a complete autonomous robot built from an ESP32-CAM and Arduino Uno for 103 US dollars. A 18.2 KB INT8 quantized neural network running on the ESP32 recognized five traffic sign types from 32 by 32 pixel camera frames with 91.78 percent accuracy under even lighting and 66.6 ms end-to-end decision latency. An OpenCV pipeline ran alongside for lane tracking and obstacle avoidance. A CRC-8 serial protocol communicated decisions to the Arduino motor controller.

Across 500 controlled trials, systematic error logging revealed that uneven lighting caused more than half of all failures, reframing the improvement priorities from model tuning to environmental changes. Five targeted fixes reduced the failure rate from 36.8 percent to 11.4 percent, with a single capacitor completely eliminating ESP32 reset failures. The confusion matrix showed that almost all classification errors involved one specific pair of visually similar signs at the chosen image resolution, giving a solid roadmap for future improvement.

The project also served as a comprehensive educational experience, developing skills in firmware, FreeRTOS task management, applied machine learning, computer vision, serial protocol design, and experimental methodology through real-world problem solving. All firmware, training data, code, and documentation are publicly available at https://github.com/Abhiram03-2009/Low-Cost_Autonomous_Robot_Navigation_Research_Project so that

other students and educators can reproduce these results and build on them.

ACKNOWLEDGEMENTS

The author thanks Dr. Vinayak Elangovan of Penn State Abington University for mentorship, guidance, and feedback throughout this project.

REFERENCES

[1] Colby R. Banbury et al. 2021. Benchmarking TinyML Systems: Challenges and Directions. arXiv:2003.04821.
[2] Rodney A. Brooks. 1986. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation* 2, 1 (1986), 14–23.
[3] Robert David et al. 2021. TensorFlow Lite Micro: Embedded Machine Learning for TinyML Systems. In *Proceedings of Machine Learning and Systems*, Vol. 3.
[4] Edge Impulse Inc. 2023. Edge Impulse Documentation. https://docs.edgeimpulse.com/.
[5] Song Han, Huizi Mao, and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *International Conference on Learning Representations (ICLR)*. arXiv:1510.00149.
[6] Andrew G. Howard et al. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv:1704.04861.
[7] Jelena Kocic, Nikola Jovicic, and Vujo Drndarevic. 2019. An End-to-End Deep Neural Network for Autonomous Driving on Embedded Platforms. *Sensors* 19, 9 (2019), 2064.
[8] Sushant S. Patil, Vignesh Prabhu, Akshay Naik, and Rahul Kulkarni. 2020. Vision-Based Autonomous Robot Using Raspberry Pi and OpenCV. *International Research Journal of Engineering and Technology (IRJET)* 7, 7 (2020).
[9] Pete Warden and Daniel Situnayake. 2019. *TinyML*. O'Reilly Media, Sebastopol, CA.