

Decoupled Perception-Actuation Architecture for Low-Cost Level 2 ADAS Prototyping Using YOLO and ESP32

Jaydeep Ravibhai Ahir

Department of Computer Engineering

Tolani Foundation Gandhidham Polytechnic

Gujarat Technological University, Gujarat, India

ahirjaydeep0018@gmail.com

Abstract—Advanced Driver Assistance Systems (ADAS) are transforming modern transportation, but their proprietary and expensive nature limits accessibility for students and independent researchers. This paper presents the development of a low-cost, functional Level 2 autonomous vehicle prototype that bridges the gap between advanced perception software and reproducible hardware. Our architecture decouples high-level computer vision from low-level physical actuation. An external computational unit processes wireless visual telemetry using OpenCV for nearest-pair lane tracking and a fine-tuned YOLO model for dynamic object and traffic sign recognition. Navigation commands are transmitted via UDP to an ESP32 microcontroller, which executes precise motor control using a Proportional-Integral-Derivative (PID) algorithm. The ESP32 has built-in hardware reflexes like ultrasonic emergency braking and an infrared-triggered corrective steering reflex for lane correction to make sure safety no matter how slow the network is. Experimental results demonstrate a 90 percent object detection success rate under optimal lighting and highly reliable obstacle avoidance, while also highlighting the environmental limitations of single-camera visual telemetry. This project proves that complex autonomous driving logic can be effectively simulated and studied on an affordable, small-scale platform.

Index Terms—Autonomous Vehicles, ADAS, Computer Vision, YOLO, ESP32, PID Control, Edge Robotics

I. INTRODUCTION

Self driving vehicles and ADAS (Advanced Driver Assistance Systems) have rapidly increased from theory concepts to commercially deployed technologies, fundamentally transforming how vehicles move. Modern transportation increasingly relies on computer vision and automated safety systems to detect obstacles. These systems are central to improving road safety, reducing human error.

There is large gap between commercial ADAS and accessible platforms for education, experimentation. Commercial ADAS are typically proprietary, highly integrated and expensive, limiting use to large companies. This complexity and cost create a barrier for students who wish to explore ADAS. There is a critical need for affordable prototypes that can accurately demonstrate core ADAS functionalities. While

established open-source academic platforms like MIT RACE-CAR [5] and DonkeyCar [6] provide excellent foundations for autonomous research, they often require relatively expensive onboard microcomputers like the Raspberry Pi or Jetson Nano to handle perception locally.

“AI Car” that serves as a functional level 2 ADAS prototype. Our system integrates computer vision algorithms, implemented using OpenCV [2] and the YOLO [1] object detection, with lightweight and responsive embedded hardware on an ESP32 microcontroller, ultrasonic sensor and IR sensor. This combination enables real time obstacle detection, autonomous braking and lane following on a small scale vehicle platform, advanced perception software and cost-effective, easily reproducible hardware.

The remainder of this paper is organized as follows. Section II details the physical hardware architecture, includes ESP32 integration, motor control systems and sensors required for the prototype. Section III details the computer vision, OpenCV preprocessing, deployment of YOLO object detection model finetuned for signs recognition. Section IV explains the system integration, showing how data is translated into real time physical command. Section V presents result of experimentation getting accuracies and responsiveness. Section VI concludes the paper and discusses future enhancements

II. PHYSICAL HARDWARE ARCHITECTURE

The hardware of the AI Car was built to be lightweight and fast enough to handle real-time driving commands. Instead of using expensive proprietary hardware, the system is divided into three practical subsystems: central processing, environmental sensors, and visual telemetry.

A. Central Control Unit and Actuation

The brain of the vehicle is an ESP32-WROOM-32 [3] microcontroller module. This specific chip was selected because its dual-core processor and native Wi-Fi capabilities easily handle high speed, real time data transfer. The ESP32

functions as the central hub, receiving instructions from the external computer and translating them into physical actions. It connects directly to a motor driver circuit to direct the motors, that make precise wheel speed and steering adjustments required for autonomous braking and lane correction.

B. Environmental Sensor Array

To give the vehicle immediate spatial awareness, a multi-sensor array is wired directly into the chassis. Forward collision detection relies on a front-mounted ultrasonic sensor. This sensor continuously bounces high frequency sound waves off physical obstacles in the car's path, acting as a hardware level fail safe to trigger emergency braking if an object gets too close.

For lane keeping, downward facing IR sensors are mounted on the undercarriage. These sensors measure the reflectance of the surface below them to distinguish between the road and bright lane markings. By constantly feeding this reflectance data back to the ESP32, the system ensures the car physically corrects its steering before drifting out of its lane.

C. Wireless Visual Telemetry

Running complex AI models requires heavy processing, the computation is entirely on external machine. The vehicle uses a smartphone camera running the IP Webcam application as an independent visual unit. The smartphone captures high resolution video and wirelessly streams it over the local network to the main computer. This approach provides low latency video feed without need to mount heavy, expensive camera processing hardware onto the car itself.

III. COMPUTER VISION

The biggest challenge was giving the vehicle the ability to visually interpret its surroundings. The ESP32 is only used for physical actuation; all visual processing is handled externally. We engineered a dual pipeline software architecture that processes the raw wireless video feed and extracts actionable driving data using computer vision techniques.

A. Video Stream Processing and Lane Tracking

The smartphone captures the environment using the IP Webcam application, and kornia_rs—a Rust-backed computer vision library—is utilized for zero-copy JPEG decoding to ingest this wireless feed into the external computational system, drastically reducing frame-capture latency. After all this, utilize OpenCV to capture and preprocess frames in real time. To ensure lane tracking algorithm remains robust under any lighting conditions, apply Contrast-Limited Adaptive Histogram Equalization (CLAHE) and Gaussian blurring to filter out visual noise and harsh glare.

Developed a "Nearest Pair" tracking algorithm, rather than relying on a rigid, hard coded split to find lane lines. The system generates a binary mask of the road and identifies all continuous white contours. By isolating the pair of lines

whose midpoint is closest to the car's true center, the system accurately tracks the lane. To account for the smartphone's physical mounting position, applied a camera offset (Δ_{off}) of 55 pixels to shift the vehicle's "straight ahead" reference point. The optimal lane midpoint M is calculated using the left (L_x) and right (R_x) contour centroids:

$$M = \frac{L_x + R_x}{2} + \Delta_{off}$$

A lane memory system uses an Exponential Moving Average (EMA) to remember line positions, preventing tracking outer boundaries during sharp turns.

B. Object Detection and Driving Logic

While OpenCV handles the lane geometry, a robust deep learning model is required to recognize dynamic objects. Instead of relying on a generic, pre-trained model that wastes processing power on irrelevant everyday objects, implemented a YOLO object detection architecture.

The YOLO model was fine tuned specifically for driving scenarios to identify critical markers such as traffic lights, stop signs, and speed limit indicators. We trained this custom model by recording proprietary video datasets of the physical road signs on the test track. By strictly narrowing the model's focus, the neural network achieves exceptional performance and low latency, recording a 91.7% mAP@50, 90.2% Precision, and 87.7% Recall during validation. Once an object like a stop sign or a pedestrian is detected, the system calculates the bounding box's pixel area relative to the entire frame. When this area crosses a predefined proximity threshold, the software registers a stop condition, establishing the necessary condition to trigger the brake.

To translate the lane data in smooth physical movement, implemented an Proportional-Integral-Derivative (PID) [4] controller. The discrete steering angle $u(t)$ is computed using the standard control law:

$$u(t) = K_p e(t) + K_i \int e(t) dt + K_d \frac{de(t)}{dt}$$

To achieve stability without oscillation at a 10 Hz command rate, the controller was manually tuned to favor proportional and derivative action, utilizing constants of $K_p = 0.22$, $K_i = 0.0$, and $K_d = 0.05$.

IV. SYSTEM INTEGRATION AND CONTROL LOGIC

The final phase of the system involves translating these insights into physical motor commands. This requires seamless integration between the external computational unit and the ESP32 microcontroller. We developed a multi-threaded communication architecture combined with a dynamic state machine and hardware-level safety overrides.

A. Asynchronous Communication and Telemetry

The communication between the external computer and the ESP32 is handled through UDP (User Datagram Protocol), that ensure the car reacts instantly to AI's decisions. The Python based control software operates on multi thread architecture. Main thread process camera feed and AI inference, other fires motor commands.

A dedicated heartbeat thread sends a ping signal to the ESP32 at a frequency of 2 Hz, operating independently of the 10 Hz motor command thread. The ESP32 utilizes a watchdog timer set to a strict 2000 ms threshold; if it stops receiving this heartbeat due to a WiFi dropout or software crash, the microcontroller automatically cuts power to the motors.

B. Dynamic Drive State Machine

Control software uses dynamic state machine with four primary modes: DRIVE, STOP, COOLDOWN, and LOST. During DRIVE the PID controller continuously adjusts wheel speeds. When YOLO model detects stop sign, system goes to STOP state, applying brakes for set duration. To prevent the vehicle from becoming indefinitely stuck at a single stop sign, it then enters a COOLDOWN state. In this state, it temporarily ignores stop signs while still enforcing mandatory safety stops for pedestrians or red lights. If lane tracking algorithm loses sight of the road for a period, the car enters in LOST state and halts until the path is reacquired.

C. Hardware-Level Reflexes

The ESP32 is programmed with embedded hardware reflexes to ensure safety, operating entirely independent of the WIFI network. The microcontroller polls a front-mounted ultrasonic sensor every 60 ms. If an obstacle is confirmed within a critical threshold of 10 cm over two consecutive readings, the ESP32 instantly overrides all AI commands and executes a hard brake.

Implemented an Infrared (IR) sensor fusion system to act as a lane-keeping fail safe. Downward facing CA-023 IR sensors monitor the road surface. If the car's chassis begins to cross a white lane boundary, the ESP32 triggers an instant 150 ms "bounce" maneuver reversing the inner wheels while maximizing the outer wheels. This reflex pivots the car back onto road, ensuring safety even if the visual AI experiences latency

V. EXPERIMENTAL RESULTS

We constructed a custom oval test track featuring 1.5-meter straightaways and a 50 cm lane width to evaluate the system. The vehicle was tested across 18 controlled runs to quantitatively assess perception accuracy, network latency, and lane-keeping reliability under continuous motion.

A. Perception and Object Detection Accuracy

The YOLO model was trained on a custom dataset of approximately 1,200 proprietary images captured directly from the track environment. Operating at an average of 4 to 5 Frames Per Second (FPS) on the external computer, the visual perception loop was constrained by a 200 ms processing bottleneck. However, the decoupled UDP telemetry maintained a dedicated network transmission latency of just 25 to 30 ms, ensuring that once an inference was made, the physical command was dispatched near-instantly. Across the 18 formal trials, the perception system achieved a 77.7% perfect detection rate (14/18 runs), identifying all signs flawlessly. This 14% degradation from the validation mAP (91.7%) to live deployment accuracy (77.7%) highlights a classic machine learning distribution shift; the model, trained on a constrained dataset of 1,200 static images, experienced confidence drops when exposed to dynamic real-world motion blur and variance in ambient lighting.

After test revealed that the system is highly sensitive to environmental factors. Because the single smartphone camera as solo visual telemetry source, the mounting angle had to be compromise. Camera was forced to capture both road surface directly ahead for lane tracking and the elevated environment for the road signs. Harsh glare or poor lighting significantly impacted the model's confidence.

B. Hardware Responsiveness and Braking

The hardware level fail safes proved exceptionally reliable during testing. The ultrasonic sensor's emergency braking system successfully overrode the drive motors in almost every test case. In the straight line approaches toward physical barrier, the ESP32 consistently halted the car between 8 and 10 centimeters away from the obstacle. This minimal stopping distance validates the architectural decision to handle emergency actuation locally on the microcontroller. By bypassing the 25 to 30 ms WiFi UDP latency and the 4 to 5 FPS visual processing bottleneck, the hardware reflex guarantees immediate deceleration.

C. Navigation and Lane Tracking

Lane keeping performance varied depending on the complexity of the track geometry. On straight sections the "Nearest-Pair" algorithm and PID controller worked seamlessly, maintaining center alignment.

However, during the tight 50-cm wide curved sections of the oval track, the successful navigation rate dropped to approximately 75% to 80%. Observations indicate the failures on curves were rarely due to the PID mathematics, but were primarily caused by limitations in the OpenCV software environment constraints. As the vehicle enters a sharp turn, the inner lane line often leaves the limited field of view of the single camera. Sudden changes in lighting across the curve sometimes caused the OpenCV mask to drop remaining lane

line, forcing the vehicle to rely on the physical IR sensors for a "bounce" correction to avoid leaving the track completely.

VI. CONCLUSION AND FUTURE WORK

The development of AI Car successfully demonstrates complex ADAS (Advanced Driver Assistance Systems) can be reproduced on an accessible, small scale hardware platform. By decoupling high level visual perception from the low level physical actuation, created a responsive Level 2 lane tracking algorithm, and hardware level fail safes like the ultrasonic sensors brake and IR corrective steering reflex resulted in a robust system capable of real time navigation and collision avoidance.

Testing also revealed limitations of relying on an external computational unit and a single visual telemetry source. The next version of the project "V2" will focus heavily on to an edge computer architecture. By removing the external computer and Wi-Fi UDP link and replacing them with an onboard microcomputer, such as a Raspberry Pi, network latency can be eliminated and AI inferences can be processed directly on the chassis.

Upgrading the hardware environment will also allow for a drastically improved sensor array. We propose a multi camera system, dedicating one downward facing camera strictly for lane tracking and a separate forward-facing camera for road sign recognition. Additionally, intend to mount the front ultrasonic sensor on a sweeping servo motor, effectively creating a wider field of view rather than a single forward vector.

These hardware upgrades resolving current environmental and lighting bottlenecks, the software ecosystem can be expanded to handle more complex driving logic. Future goals for software include refining the steering mathematics for perfect reliability on sharp curves, implementing a trajectory planning module for automatic overtaking, and developing an autonomous parallel parking sequence.

ACKNOWLEDGMENT

The author wishes to thank their peers at Tolani Foundation Gandhidham Polytechnic for their collaborative assistance with the physical chassis wiring and sensor mounting during the hardware prototyping phase.

REFERENCES

- [1] Ultralytics, "Ultralytics YOLO," 2023. [Online]. Available: <https://github.com/ultralytics/ultralytics>.
- [2] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.
- [3] Espressif Systems, "ESP32-WROOM-32 Datasheet," 2023. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf.
- [4] K. J. Åström and T. Hägglund, *PID Controllers: Theory, Design, and Tuning*. Research Triangle Park, NC: Instrument Society of America, 1995.
- [5] Karaman, S., et al., "Project RACECAR: Rapid autonomous complex-environment competing ackermann-steering robot," *MIT Professional Education*, 2017.
- [6] DonkeyCar Community, "DonkeyCar: An open source DIY self driving platform for small scale cars," 2024. [Online]. Available: <https://www.donkeycar.com/>.