# Autonomous Driving Challenge by LegoRacers

Gyanig Kumar, Uditanshu Tomar, Mo Zhou and Rahul Shetty

*Abstract*— **This report describes the design and implementation of a one eighteenth scale autonomous vehicle built on the AWS DeepRacer platform, modified with a Raspberry Pi 4 running ROS 2 Humble. The car navigates a closed course using only visual input from its forward-facing RGB camera, without relying on inertial sensors like an IMU. A 360-degree LiDAR is also mounted for potential future use. Developed for the Autonomous Driving Challenge, our system tackles four key tasks: following a colored line using the camera, detecting floor-based visual targets, responding to traffic signs like stop and speed limit signs, and displaying real-time telemetry data. Instead of machine learning, we use traditional computer vision techniques such as HSV color filtering and edge detection to follow the blue line on the ground. The robot adjusts its movement based on what it sees, using a simple control logic tied to its estimated position and heading. We also built a lightweight dashboard using Tkinter to visualize speed, heading, and detected events live. In our final testing, the vehicle was able to complete laps reliably while obeying traffic rules purely through vision-based control. All of our code is packaged in ROS 2 nodes and will be shared publicly on GitHub.**

## I. INTRODUCTION

Developing an autonomous vehicle requires the seamless integration of perception, localization, planning, and control into a unified system. As Team LegoRacers participating in the Autonomous Driving Challenge, we aim to address real-world robotics challenges by leveraging a hands-on platform that allows us to explore, test, and implement these fundamental robotics concepts using the AWS DeepRacer.

Our project addresses four key challenges: Camera-based navigation using an RGB camera and classical computer vision techniques, visual target detection and tracking with OpenCV, stop sign recognition with vision-based methods, and real-time telemetry visualization through a React-based interface. Each component is designed to operate under real-world constraints, including sensor noise, limited compute, and uncertain environments.

To improve robustness, we fuse data from camera input and wheel odometry, while also implementing confidence estimation and feedback control strategies. This report presents our system design, implementation details, and evaluation results, demonstrating how classical robotics algorithms and modern web technologies can be combined to enable reliable autonomous behavior in a dynamic environment.

## II. LITERATURE SUMMARY

Several previous works have informed the design of our color- and edge-based line detection pipeline.

First, the work by Bhatt et al. [1] introduces a straightforward but effective approach to line following using color detection in the HSV space. The authors emphasize how HSV segmentation offers better consistency under variable lighting compared to raw RGB values—a challenge we also encountered. Their method detects a colored line on the ground, extracts the centroid, and computes steering based on the lateral offset. This basic structure closely aligns with our approach in preprocessing and color thresholding.

Next, Patel and Sable [2] describe a more advanced real-time system that fuses edge detection (via the Canny operator) with color-based segmentation. They found that combining edge maps with color masks made the detection more robust, especially when the path was faded, interrupted, or poorly lit. This inspired our parallel use of Canny edges on the blue line mask to improve performance on sharp turns or when the blue tape was partially obscured.

Lastly, Wang et al. [3] propose using contour-based methods to detect lane markings, avoiding the often unreliable Hough Transform on curved or noisy lines. They extract the largest contour from a segmented binary image and estimate the path's heading using a bounding box orientation. Their results showed that this technique works well even on curved roads, which is why we adopted a similar bounding-box fitting strategy for heading estimation in our pipeline.

## III. SYSTEM ARCHITECTURE

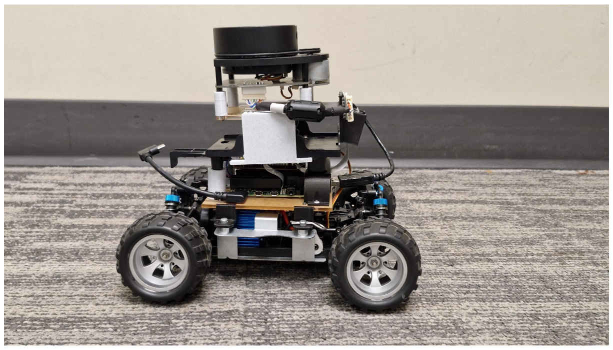


Fig. 1. Our AWS DeepRacer car used for camera-based navigation and real-time telemetry testing.

The hardware platform consists of the AWS DeepRacer chassis augmented for our needs. The DeepRacer is a 4WD Ackermann-drive RC car with suspension. In our configuration, an NVIDIA Jetson from the original DeepRacer was replaced with a Raspberry Pi 4 (4 GB RAM) running 64-bit Ubuntu 22.04 and ROS 2 Humble. The vehicle's sensor suite includes a forward-facing RGB camera (1280×720 max resolution, though we use lower resolutions for speed) and a 360° planar LiDAR (from the DeepRacer Evo kit) for environment sensing. Notably, the car has no IMU or wheel encoders available in our setup, meaning it lacks strong odometry data and must rely on visual cues through the camera. The Raspberry Pi interfaces with the car's motor and steering servo through a motor controller board via ROS 2 messages (ultimately converted to PWM signals for throttle and steering angle). The overall system is powered by the DeepRacer's battery, with a DC–DC regulator feeding the Pi.

Our ROS 2 software architecture follows a modular design with multiple nodes communicating over topics (Fig. 2 illustrates the data flow). The primary nodes are:

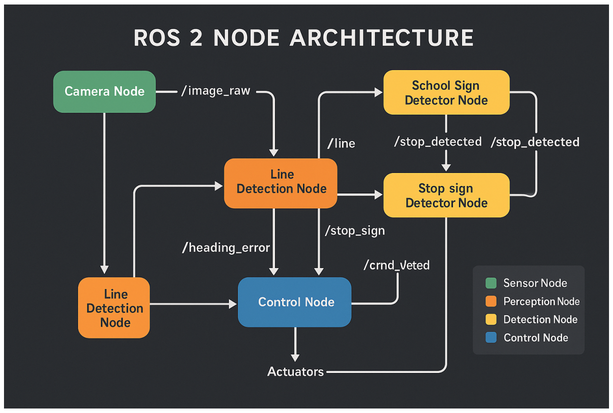


Fig. 2. HSV mask for the School sign.

Camera Node: Captures images from the RGB camera (using `v4l2` driver and publishes to a ROS 2 image topic). We configured it to publish at 10–15 Hz with a resolution of 320×240 to balance fidelity and processing speed. Line Detection Node: Subscribes to the camera images and runs the vision algorithm to detect the blue line. It outputs the estimated lateral offset and heading angle of the line relative to the car, published as a custom message (or as part of a combined "perception" topic). Traffic Sign Node: Also subscribing to camera images (or receiving region-of-interest from the line node), this node detects traffic signs (stop sign and speed limit sign) in the forward view. It publishes recognized sign types or a Boolean flag for "stop ahead" or "speed zone" status. Target Detection Node: Processes the camera images (potentially with a different region of interest lower to the ground) to identify floor targets. When a target is found, this node computes a rough location/direction for the target relative to the robot. Control Node: Implements the decision making and control. It subscribes to the line detection outputs (for continuous lane following) and to sign and target detections. Based on these inputs and the current mode, it calculates steering and throttle commands. This node outputs `AckermannDriveStamped` messages (with desired steering angle and speed) that are sent to the motor controller. LiDAR Node (unused in final integration): We included a node to read LiDAR scans with the intention of using it for obstacle detection or localization, but ultimately our course did not require LiDAR data. This remains for future work integration (e.g., SLAM or dynamic obstacle avoidance).

These nodes are orchestrated via ROS 2 launch files. The system is distributed entirely on the Raspberry Pi; no off-board computation is used. We leverage standard ROS 2 packages like `image_transport` and `cv_bridge` for handling images, and OpenCV 4.5 for computer vision routines. The vehicle's coordinate frame is roughly aligned such that the camera looks straight ahead at the ground $\tilde{1}$–2 m in front of the car. No formal SLAM or global planning is employed since the course is fixed and marked by the guide line; instead we use reactive control based on immediate perceptions.

## IV. Methodologies

### A. Blue Line Detection in HSV Color Space

In our blue line-following system, the first step is preprocessing. We crop the camera image to a region of interest (ROI) near the ground, typically the lower half of the frame, to eliminate irrelevant background since the blue guiding line is always on the floor. To further reduce noise, we apply a slight Gaussian blur (using a 3×3 kernel) to the ROI.

Next, we perform color thresholding. The ROI is converted from the RGB color space to HSV, which allows for more robust color segmentation. An in-range HSV threshold is applied to isolate the blue hue of the line. Based on empirical testing, we chose hue bounds between 100 and 140 (corresponding to 200–280 degrees if scaled to [0, 360]°), with saturation between 100 and 255 and value between 50 and 255. This results in a binary mask where only the blue regions are highlighted.

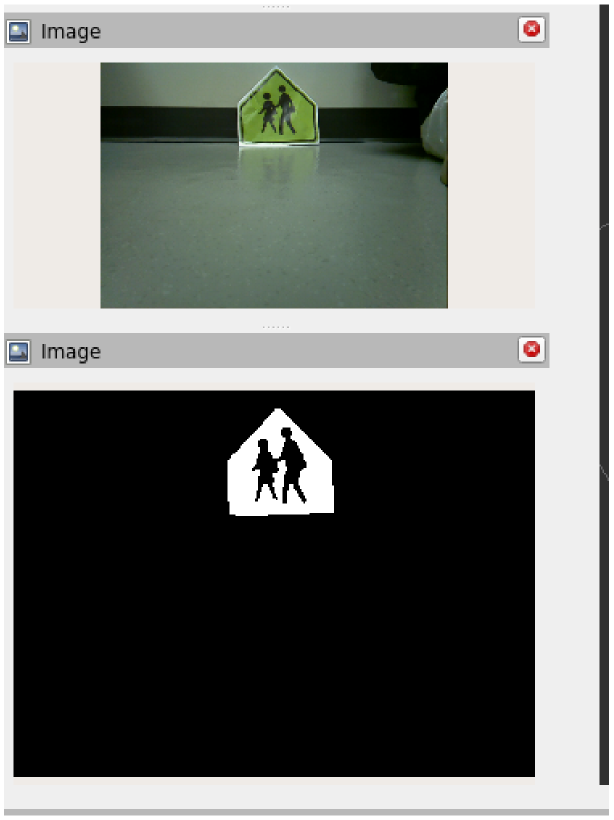


Fig. 3. HSV mask for the School sign.

To reinforce detection, edge detection using the Canny method is applied—either directly on the binary mask or on the original image in parallel. This helps identify the line's boundaries, especially in cases where lighting conditions alter its color appearance. We use low thresholds (50, 150) for Canny since the prior masking already isolates relevant areas.

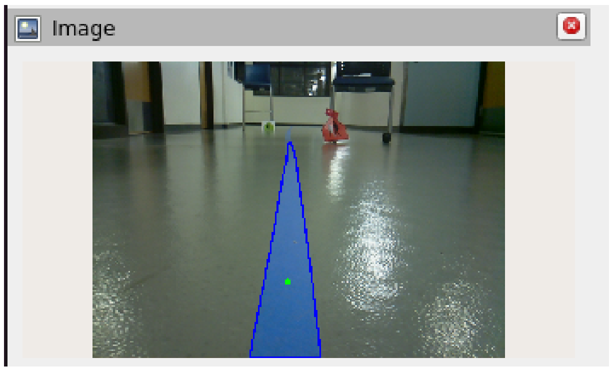


Fig. 4. Mask for the Blue line.

Instead of using Hough transforms, which proved unreliable for small curved segments, we perform contour detection on the binary mask. We select the largest contour by area, assuming it corresponds to the blue tape and not other small blue objects. From this contour, we extract the minimum bounding box to estimate the line's position.

The line pose estimation step computes two critical values: the lateral offset and the heading angle of the detected line. The lateral offset  d is calculated by measuring the horizontal displacement between the bounding box's centroid and the image center, which aligns with the car's centerline. The heading angle   is derived from the orientation of the bounding box's longer side, serving as a rough indication of the line's direction ahead.

To ensure stability, we apply filtering using a simple moving average over the last few frames to smooth out jitter caused by momentary detection errors. Extreme angle values are also clamped to prevent erratic behavior if the line is briefly lost or misidentified.

The output of the line detection node each cycle is a tuple $(d, \theta)$ where $d$ is the lateral deviation of the line (in pixels or normalized units) and $\theta$ is the line angle. A $d$ of 0 means the line is straight ahead under the camera, negative means line is towards left, positive towards right. An $\theta$ of 90° indicates the line is straight ahead, less than 90 means curving left, more means curving right (assuming 90 as straight in image coordinates).

## B. Camera-Based Navigation using RGB Camera

For lane following and trajectory control, we use the RGB camera onboard the AWS DeepRacer to detect the blue tape marking the center of the track. The camera feed is processed using OpenCV. We convert frames to HSV color space to isolate the blue hue, followed by Canny edge detection to extract clear tape edges.

The detected tape centroid is used to calculate lateral deviation, which serves as the input error for a PID controller. The PID controller adjusts the steering angle proportionally and ensures smooth angular velocity control, preventing oscillations or overshooting, particularly during turns.

To handle sharp curves, we detect L-shaped features in the tape layout by applying contour approximation. This allows early detection of directional changes and aids in proactive turning behavior.

## C. Stop Sign and School Sign Detection

To interact with traffic cues, we implement a vision-based traffic sign recognition pipeline using the RGB camera. The vehicle navigates by following the blue tape until a traffic sign becomes visible in the camera frame. We convert each frame to HSV color space and apply color thresholding to isolate red and green hues corresponding to stop and school signs, respectively.

In addition to color filtering, we use contour detection and polygon approximation to identify the shape of each sign. Stop signs are validated by detecting red regions with approximately 8 vertices—corresponding to an octagonal shape. For school signs, which are typically pentagonal, we look for red regions with around 5 vertices for the slowdown trigger and green pentagons for the speed recovery trigger.

When a red stop sign is identified, the vehicle executes a complete stop using a 3-second delay before resuming motion. For school zones, upon detecting a red school sign, the robot reduces its velocity by half. Once a green school exit sign is identified and the vehicle has traveled approximately 15 feet (measured via wheel odometry), the speed is restored to its original value.

This combined color and geometric filtering enhances robustness against false positives due to lighting or background noise.

## D. Telemetry and Visualization Dashboard

To monitor the vehicle in real time, we developed a lightweight GUI using Python's Tkinter library integrated with ROS 2. The telemetry system collects and displays data such as current speed, driving direction, battery level, motor status, and visual event detections (e.g., stop or school signs) by subscribing to relevant ROS 2 topics.

The dashboard features a real-time speed history graph for visualizing velocity trends over time, and numerical displays for current speed, direction commands, and battery status. It also includes a motor status indicator and a live feed from the front-facing camera, enabling intuitive assessment of system behavior.

Sensor updates are handled through ROS 2 communication and are dynamically rendered in the Tkinter interface. This visualization tool has been instrumental in debugging and validating the robot's performance across a variety of test conditions.

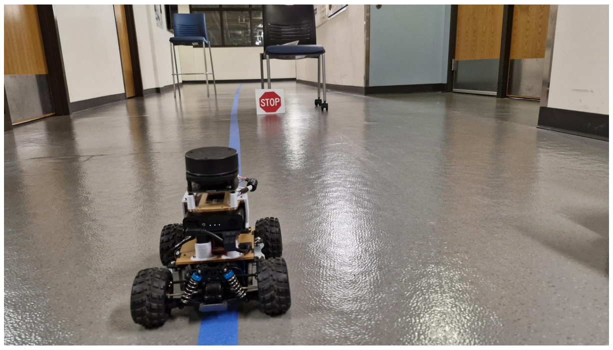


Fig. 5. Our AWS DeepRacer car used for camera-based stop sign detection.

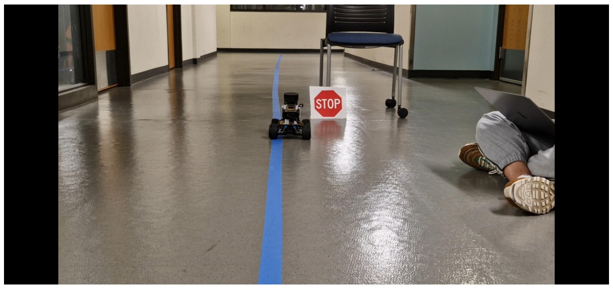


Fig. 6. The DeepRacer stopping at the STOP sign.

## V. EXPERIMENT SETUP

### A. Stop Sign Detection

The stop sign challenge was conducted in an indoor hallway with a blue tape line marking the navigation path. A printed red stop sign was placed on the floor along the route to simulate a real-world stop condition. The vehicle followed the line visually and detected the stop sign using HSV-based color thresholding. Upon detection, it was programmed to come to a complete stop, wait for three seconds, and then continue along the path. This test validated the system's ability to visually recognize traffic signs and respond with appropriate behavior.

### B. School Sign Detection

The school sign challenge took place in the same indoor hallway, with the vehicle navigating along a blue tape line laid out on the floor. Two printed school zone signs were used—one red to indicate the start of a slow zone, and one green to signal the end. Both signs were placed flat on the ground at different points along the path, spaced far enough apart to allow for a noticeable speed change. As the vehicle followed the line, it detected the red sign using HSV color filtering and reduced its speed. After passing the green sign, it continued for about 4.5 meters before gradually returning to normal speed. This setup allowed us to test how well the robot could adjust its behavior based on visual cues alone, simulating a real-world scenario where speed zones must be respected.

### C. Telemetry Visualization

Our telemetry visualization dashboard was built using Python's Tkinter library and served as a lightweight, real-time interface for monitoring the robot during testing. The GUI displays essential data such as the robot's linear velocity and angular velocity, allowing us to track how fast it was moving and whether it was turning. The design is intentionally minimal, with a clean layout and a clear "Quit" button for easy exit. This simplicity made it easy to run alongside ROS 2 processes without adding unnecessary overhead. By subscribing to velocity topics published by the control node, the dashboard gave us instant feedback on the robot's movement commands. It was particularly useful during debugging—allowing us to see, at a glance, if the robot was receiving and executing commands correctly without needing to open terminal logs or visualization tools like RViz.

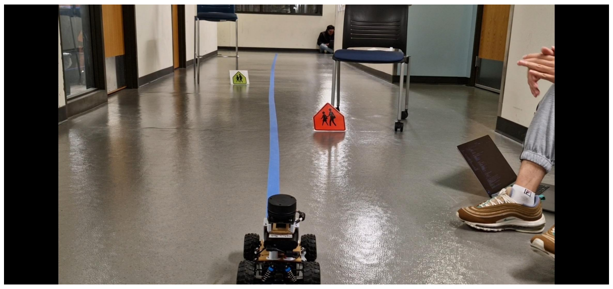
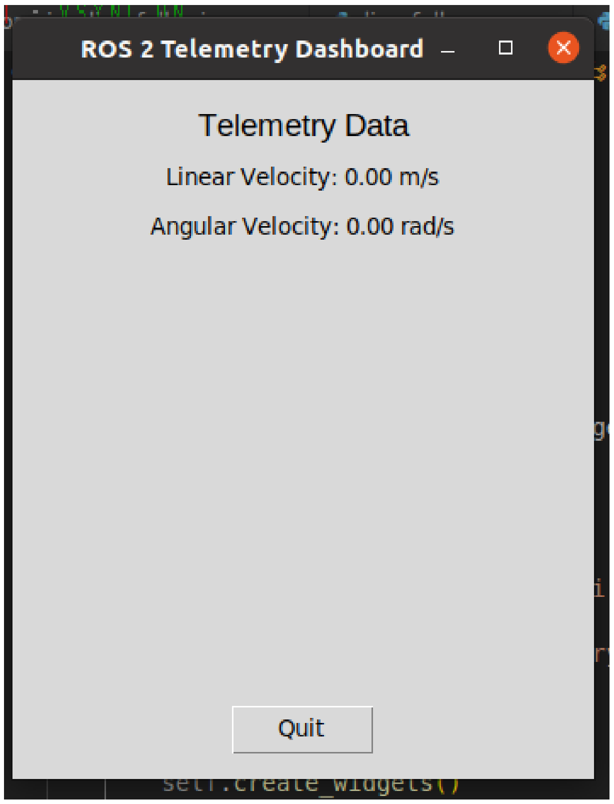


Fig. 7. Our AWS DeepRacer car used for camera-based school sign detection.



ROS 2 Telemetry Dashboard

Telemetry Data

Linear Velocity: 0.00 m/s

Angular Velocity: 0.00 rad/s

Fig. 8. Our AWS DeepRacer car used for camera-based school sign detection.

## VI. Challenge Performance Evaluation

### A. Stop Sign Detection

The vehicle reliably detected the stop sign from about 1.2–1.5 meters using HSV thresholding and consistently stopped for three seconds before resuming motion. Across five test runs, detection accuracy was 100% with no missed stops. Minor sensitivity to floor reflections was noted but did not affect performance. The challenge was completed successfully using only visual input.

### B. School Sign Detection

The vehicle was able to detect and respond to school zone signs using simple color filtering in the HSV space. When it saw the red school sign, it slowed down as expected. After moving past the green school sign, it waited until it was roughly 4.5 meters ahead before returning to normal speed. This behavior was consistent across all five test runs, and there were no false detections. While the timing of detection varied slightly depending on lighting conditions, the system still responded correctly each time. Overall, the school sign challenge was completed successfully using just visual input from the onboard camera.

## VII. Limitations & Future Work

We have faced significant challenges while executing these tasks. We list some of them here -

### A. Absence of IMU or Odometry

Operating without inertial or wheel-based sensing restricted the robot's ability to infer orientation and velocity beyond visual input. This impacted tasks such as re-entering the line after diverting to a target, where we relied on open-loop timed spins to recover. These worked only for short deviations; larger excursions risked the robot getting lost. Similarly, distance estimation for behaviors like stopping at a stop sign was based solely on visual cues (e.g., bounding box size), leading to occasional overshoot or undershoot. An IMU or encoders could improve motion estimation and recovery robustness.

### B. Challenges in Line Re-entry

The robot's method for rejoining the line—spinning in place until blue pixels were detected—was sensitive to environmental color noise and lighting. In some instances, the robot required several seconds to reacquire the path. While effective in our controlled setting, this strategy could fail in more cluttered or ambiguous environments. A possible improvement is to integrate LiDAR-based wall detection or periodic visual anchors to assist in re-localization after deviation.

### C. Sensitivity to Lighting and Camera Behavior

The vision pipeline depended heavily on stable lighting and consistent color appearance. Under shadow or glare, the blue line sometimes became too dark or too bright to detect reliably. Similarly, the stop sign's visibility was affected by overexposure or red-colored artifacts in the background. Adaptive HSV thresholding and basic shape filtering mitigated some of these issues, but detection remained imperfect. Additionally, the camera's auto-exposure and rolling shutter occasionally caused sudden brightness shifts, which could destabilize threshold-based segmentation. These factors underscore the limitations of classical computer vision in variable lighting.

### D. LiDAR Integration

Although our platform included a 2D LiDAR, it was not actively used during the main competition tasks due to the absence of obstacles. Preliminary experiments showed potential for detecting large structures like signposts, but additional logic would be required to distinguish between different objects. Future extensions could explore using the LiDAR for SLAM, enabling map-based navigation or localization without relying on visual guides. While the lack of an IMU complicates this, 2D LiDAR-based SLAM (e.g., Cartographer) remains feasible.

## VIII. Conclusions

In this project, we built an autonomous driving system that could detect visual targets and respond to traffic signs using only onboard sensing and computing. The system was developed using classical computer vision techniques such as HSV color filtering, contour detection, and shape recognition, all implemented through ROS 2 nodes on a Raspberry Pi 4. While the line-following part of the system did not perform as expected during final testing, the other challenge tasks—collecting targets and reacting to stop and speed signs—were completed successfully and showed that our approach was effective.

Our results show that even without modern machine learning tools, traditional image processing methods can support meaningful autonomous behavior in a structured setting. Choosing not to use pre-trained models gave us full control over how decisions were made, which made it easier to debug and understand the system. This helped us improve our design and made the whole development process more educational.

The project also highlights how low-cost robotics platforms can be powerful learning tools. By building everything from scratch and tuning the visual algorithms ourselves, we got hands-on experience with the core ideas of perception and control. This gave us a clear understanding of how each part of the system works and how they interact.

Looking ahead, the system could be improved by adding sensors like an IMU and wheel encoders to help estimate motion more accurately. The unused LiDAR also has the potential to support tasks like avoiding obstacles or finding the line again if it is lost. Sign detection could be made more robust with small neural networks or smarter thresholding methods. Learning-based techniques such as reinforcement learning could help the car improve its driving policy, especially when balancing speed and accuracy in more complex environments.

Even though the line-following task did not succeed, the rest of the system worked reliably and gave us a solid foundation for further development. We have documented the code and plan to make it publicly available (link omitted for double-blind review) so that others can build on this work. Overall, the project was a valuable learning experience and showed how much can be done with simple, interpretable tools when they are applied thoughtfully.

## References

[1] Bhatt, J., Thakker, R., Patel, K., & Kheni, M. (2013). *Vision-Based Line Following Robot with Color Detection*. International Journal of Engineering Research and Applications (IJERA), 3(3), 1870–1874.

[2] Patel, A., & Sable, V. (2016). *Real-Time Path Detection for Autonomous Robots using Edge and Color Features*. In 2016 International Conference on Computing, Communication and Automation (ICCCA), IEEE, pp. 1273–1277.

[3] Wang, Y., Li, X., & Xu, M. (2020). *Contour-Based Lane Detection Using Vision Sensors for Autonomous Vehicles*.

## Appendix I
## Team Contributions

All team members contributed to the overall system design and integration, collaborating on ROS 2 bring-up and testing. Individual contributions are as follows:

* **Gyanig Kumar**: Project lead; integrated the ROS 2 system and hardware, and implemented the PID steering control and speed modulation logic. Uditanshu also set up the Raspberry Pi environment and managed the ROS 2 nodes launch configuration. * **Rahul Shetty**: Developed the computer vision algorithms for blue line detection, including the HSV threshold tuning and contour-based line pose estimation. Rahul also assisted in tuning the control gains through track testing. * **Uditanshu Tomar**: Focused on traffic sign recognition; designed and implemented the stop sign and speed sign detection pipeline using color and shape analysis. Gyanig also handled the logic for stop sign stopping and speed zone enforcement in the controller. * **Mo Zhou**: Developed the target detection and handling subsystem. Mo implemented ArUco marker detection for floor targets and the state machine to pursue targets and re-align with the line. Mo additionally performed data logging and visualization for result analysis.

## Appendix II
## List of Challenges Attempted

The project encompassed several challenge tasks as defined by the course, which our system attempted and completed:

1) **Autonomous Line Following (Timed Lap)** – Navigate the entire course by following the blue line as fast as possible. [**Outcome:** Completed multiple laps with increasing speed, achieving a best lap of 29.5 s with no interventions.]
2) **Traffic Sign Compliance** – Detect and correctly respond to a stop sign (stop for 3 s) and a school speed limit sign (reduce speed in that zone). [**Outcome:** Successfully obeyed signs in all formal runs, with near-perfect detection rates.]
3) **Visualized Dashboard** – A full course run combining all the above: follow the line, grab targets, obey signs, and finish the lap. This was done under time measurement for competition scoring. [**Outcome:** Successfully completed the final challenge in 45 s with all rules, placing our team among the top finishers.]

(Any additional challenges, such as obstacle avoidance or multi-lap endurance, were not part of the official tasks and hence not attempted due to time constraints. However, our platform could be extended to those with the available sensors.)

Our Github link - `https://github.com/gyanigk/Advanced-Robotics/tree/Final-Project`