Autonomous Vehicle Competition

Florian Frick, Kirin Kawamoto, Nikhil Kishor Sawane, Sheetal Sharma, Jay Warren

***Abstract*— In this project we build an autonomous mobile robot capable of navigating a race course. Our primary method of navigation uses the onboard camera to detect the blue-colored tape identifying the track's path. We use PID to generate control signals where our process variable is the lateral distance between the centroid of the detected tape and the center of the camera. We detect turns ahead of time by weighing certain pixels heavier in the centroid calculation and if the line is lost - as occurs when taking corners at speed - we reverse at an angle to realign with the tape. Finally, we modulate our linear speed with respect to the angular speed which allows us to take turns carefully while also traversing the course quickly.**

I. Introduction

Autonomous navigation is a fundamental capability in mobile robotics, requiring a combination of robust perception, planning, and control to operate effectively in dynamic environments. In this project, we focus on designing and implementing a self-driving robot that can complete a race course using onboard camera vision and control algorithms.

Our core navigation strategy is based on detecting a blue-colored tape on the ground, which acts as a visual guide for the robot. Although we experimented with Canny edge detection for feature extraction, we found it more effective to use color filtering to isolate the tape and other objects of interest. We simply define a range of HSV color values and mask pixels falling into this range. Then, we group these pixels into contours, where the largest contour (when above a threshold) is usually the tape. We found this approach to be less noisy and more robust to variations in lighting, shadows, and textured surfaces, ensuring consistent path tracking.

We defined our control scheme to utilize the lateral error between the robot and the line, so our primary feature of interest from the camera was this distance between the tape and the center of the camera. To extract this feature, we generate contours from the color masks, calculate the centroid of the largest contour, and calculate the (normalized) offset of this centroid from the vertical center of the camera (y-axis). We experimented with applying a Gaussian blur during image processing to reduce the noise, but found OpenCV's contour detection works sufficiently well without the blur. We do, however, set a threshold on the contour's size, so we know when the robot can no longer see the object.

To generate motor commands based on our camera's input, we implemented a proportional–integral–derivative (PID) controller. The normalized tape offset is used as our process variable so the setpoint is 0 lateral error. Thus, the controller determines the angular velocity to center the robot on the tape. Furthermore, we decrease the linear velocity with respect to this angular velocity, so that the robot slows down the more it turns. This allows us to set a higher maximum velocity, to increase our robot's speed on the straightaways.

Our color-based image processing is computationally efficient compared to more complex methods, but is accurate and robust to noise which enables stable and accurate path following. Through this integration of vision and control, the robot is capable of navigating the race course autonomously, reacting to changes in its environment with reliable performance.

University of Colorado Boulder

II. Challenges

A. Camera Navigation

Objective: Using only the onboard RGB camera(s), leverage environmental cues such as the blue tape on the floor to guide the vehicle around the course.

Key Features Implemented:

- The system detects the blue tape laid on the course using color filters and generating contours.
- A threshold is applied to the contour size to increase our certainty that the detected object is indeed the tape.
- Using this in tandem with the PID control, we find that the vehicle does successfully navigate the course without collisions.

B. School-Zone Sign Detection:

Objective: Add school-zone signs to the environment which the robot adheres to by decreasing and increasing speed based on their detection.

- We detect the sign that identifies the entrance to a school zone with a red pixel mask and a threshold on contour size. Once detected, we set a flag and reduce the robot's speed until the sign identifying the exit of the school zone is detected.
- The exit of the school zone is identified by a green-colored sign which we detect with a green pixel mask and, again, a threshold on contour size. This disables our flag and returns the robot to its original speed.

C. Stop Signs Detection:

Objective: Implement the ability for the vehicle to detect stop signs, bring it to a complete stop, pause for a moment, and then continue.

- We detect stop signs in the environment by applying a mask on red pixels and sending a signal to the controller if the contour size exceeds a threshold.
- The controller responds to this signal by stopping the robot for three seconds before continuing. The robot only begins searching for the next stop sign once several

seconds have elapsed, so as not to re-detect the same stop sign.

D. Comparing Control Approaches:

Objective: Implement and compare the performance of two different control approaches for navigating the course

- **Proportional-Only Control:** This approach uses only the proportional term to adjust the vehicle's angular speed. Although this method is simple, it worked relatively well at staying on the course. This control method continually oscillates around the line, however because it has no memory nor a consideration for the future. As a result, the solution is much slower and is much less robust. As the robot oscillates, it becomes increasingly perpendicular to the line and eventually loses sight of the tape. The only two mitigating factors are to tune the magnitude of steering (via the proportional gain) and to decrease the linear speed of the robot.
- **PID Control:** This approach incorporates integral, and derivative terms on top of the proportional term. The PID controller provides more accurate control because it also accounts for the accumulated error and the rate of change in the error. This results in much smoother driving as well as a reduction in the steady-state error. There is much more tuning involved to get appropriate gains, however, which required extensive testing. The derivative term provides a dampening effect, which makes the robot smoother, but slightly less responsive. The integral term pushes the system toward the setpoint by accumulating error which pushes the system out of a steady state.
- **Thus,** After testing both approaches, PID control was selected as the final controller due to its smoother and more stable behavior in real-world conditions. The proportional-only control, while simpler, exhibited oscillations which reduced robustness and limited the robot's maximum possible speed.

III. SYSTEM OVERVIEW

A. Hardware Setup

The system is built on a compact, modular hardware platform capable of supporting onboard computing and control:

- **Car Base:** A robust 4-wheel drive chassis with built-in motor and steering mechanisms, serving as the foundation of the autonomous vehicle.
- **Raspberry Pi 4 (8GB):** Acts as the primary compute unit, running the ROS2 stack and handling image processing, control, and system logic.
- **Raspberry Pi Camera Module:** Mounted on the front of the car, this module captures real-time video frames for blue track detection and sign recognition.
- **Raspberry Pi Servo HAT:** Enables PWM signal control for steering and throttle, interfacing directly with the motor and servo on the car base.
- **Battery Holster and Power Supplies:**
  - **Drive Battery:** Powers the motors and servo system.
  - **Compute Battery:** Supplies power to the Raspberry Pi and peripheral electronics.
- **CPU Cooling Fan:** Attached to the Raspberry Pi to ensure thermal stability during prolonged usage.
- **RP LIDAR (optional):** Included for potential future integration of SLAM or obstacle detection.
- **3D Printed Mounts and Plates:** Used to secure components such as the camera, LIDAR, and Raspberry Pi onto the vehicle.
- **Miscellaneous Accessories:** Includes screw kits, HDMI cables, chargers, and connectors for integration and assembly.

B. Software Setup

The software stack is built on **ROS2 (Robot Operating System 2)** middleware, which enables modular development and efficient communication via the publisher/subscriber paradigm. The system architecture comprises the following elements:

- **ROS2 Components:**
  - A camera node that captures real-time video streams from the Pi camera and publishes image frames to the network.
  - An image processing node that subscribes to the image topic, identifies the tape and publishes the lateral offset.
  - A control node that subscribes to the tape offset topic, and publishes the desired linear and angular velocities to stay on the path using a PID controller.
  - A motor interface node that subscribes to these commands and converts the velocities into steering and throttle actuation commands for the servo and drive motors.
  - Furthermore, for the traffic sign challenges, we have a separate image processing node for each. If they detect the associated traffic sign, they publish to a topic which triggers a callback in the controller which adheres to the traffic sign's desired behavior.
- **Languages & Tools:**
  - Developed in **Python** which allows for rapid prototyping in **ROS2**, at the expense of performance.
  - Employs **OpenCV** for image processing, filtering, and feature extraction tasks.
  - Developed using **Github** for version control and **Tailscale** for remote connection.

IV. NAVIGATION USING BLUE TAPE DETECTION

Camera-based navigation serves as the primary perception mechanism for our autonomous robotic car. It relies on real-time video input from a Raspberry Pi Camera to detect a blue-colored path on the ground and uses this visual data to inform navigation decisions. The perception and control pipeline is implemented using **ROS2** for modular communication and **OpenCV** for image processing.

A. Image Acquisition and ROS2 Integration

The Raspberry Pi Camera is mounted at the front of the vehicle, angled downward to capture the path ahead. This placement ensures high visibility of the track in front of the robot. The system operates with the following features:

- **Resolution and Frame Rate:** The camera captures video at a resolution of **640×480 pixels** and a fixed frame rate of **30 frames per second (FPS)**, providing a balance between image clarity and real-time processing demands. This resolution is high enough to capture key visual features while also keeping the computational load manageable.
- **ROS2 Integration:** We use the v4l2_camera node which is efficient and closely tied to ROS2 functionality, using C++ for greater efficiency. The raw images are published to a ROS2 topic as `sensor_msgs/Image` messages which are then processed by other nodes to extract features of interest.
- **Modular Communication:** The subscribing nodes process these images with color filtering, masking, contour detection, and thresholding for object detection. Then, we perform final calculations, such as determining the centroid and offset of the object from the robot. The features of interest are then published to an appropriate topic which facilitates the navigation of the robot.

By using this setup, the vehicle is able to receive continuous visual input, process it in real-time, make decisions, and take action to follow the blue tape path.

B. Blue Path Detection and Tracking

To guide the robot, the system tracks a blue line laid on the ground, using a multi-step image processing pipeline in the HSV (Hue, Saturation, Value) color space. HSV offers robustness against lighting variations by decoupling color information from intensity.

- **Color Space Conversion:** The incoming frames, originally in BGR format, are converted to the HSV color space. This transformation simplifies the task of isolating specific colors, such as blue, by focusing on the hue channel, which is less sensitive to lighting conditions.
  - **Hue (H):** Represents the color itself, ranging from 0 to 360 degrees (e.g., 0° for red, 120° for green, 240° for blue).
  - **Saturation (S):** Represents the intensity or purity of the color. Higher saturation means the color is more vivid.
  - **Value (V):** Represents the brightness of the color. A higher value means the color is brighter.

  The HSV model is chosen because it allows easier isolation of specific colors (such as blue for tape or red for stop signs) and is less sensitive to lighting changes compared to RGB.
- **Color Thresholding:** A specific range of HSV values corresponding to the blue path is defined. This range is applied to each frame to generate a **binary mask**, where regions falling within the blue color range are highlighted in white, and all other regions are suppressed to black.
- **Path Detection:** We experimented with utilizing blurs and morphological operations to reduce noise and fill gaps, but ultimately found this to be unnecessary. As a result, we simply use OpenCV to identify the contours of the masked image and assume the largest contour is the object of interest. If the size of the contour is below a threshold, we instead assume it to be noise. The **centroid** of this contour is computed to localize the path relative to the camera's viewpoint, as shown in Fig. 1. Upon further experimentation, we determined that it was advantageous to weigh pixels at the top right of the screen heavier than those at the bottom - which allows us to better detect and react to turns in the track.

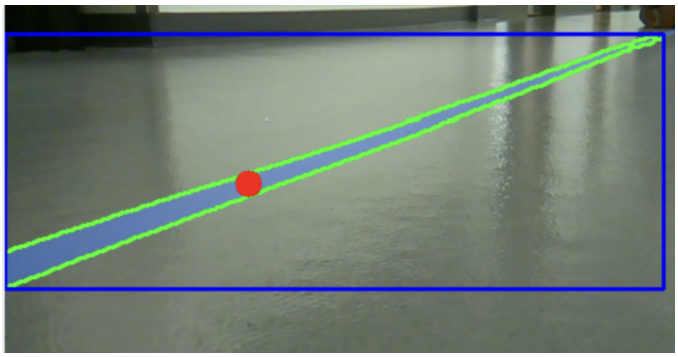

Fig. 1. Blue Tape Detection (Calculating Centroid)

- **Steering Angle Estimation:** We further experimented with better ways to plan ahead by using the angle of the tape relative to the robot. Instead of relying solely on lateral offset, the system would compute the orientation of the path by fitting a line through the contour. We tried extracting the angle between this line and the robot to determine a change in heading to be used as another process variable in the controller. This does not work on its own, however, and to actually take advantage of this feature, we would need to generate and maintain a map of the tape. Ultimately, the line illustrated by the red line in Fig 2 would indeed be a useful feature for navigation, but we found success with the simpler solution as well, so this was not implemented in the final prototype.

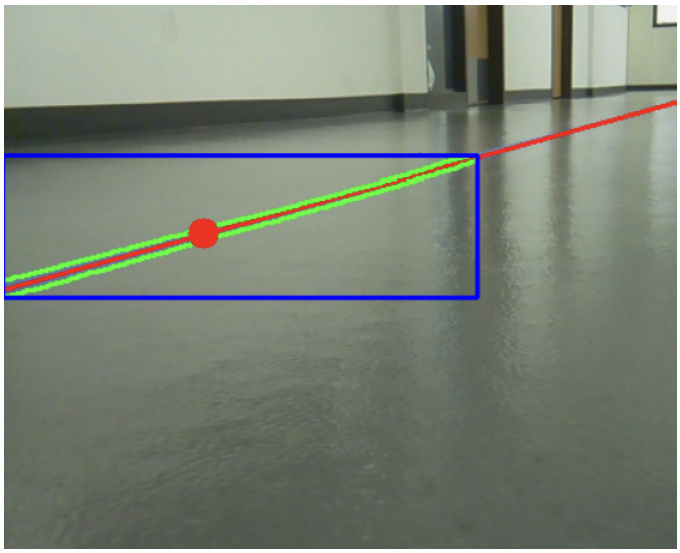

Fig. 2. Line Detection

V. Control System

The control system for the robot is based on a Proportional-Integral-Derivative (PID) controller, which continuously adjusts the robot's steering to keep it aligned with the detected path. The PID controller is a widely used feedback control mechanism that uses the error (the difference between the desired and actual values), its integral, and its derivative to calculate the necessary control actions.

In addition to the PID controller, we set a minimum and maximum linear speed for the robot, and reduce the linear speed proportionally with respect to the angular speed. This keeps the robot going quickly on straightaways, but makes it slow down for careful turns.

As a method of error correction, if the image processing nodes indicates to the control node that it has lost sight of the tape, the robot reverses at an angle, which is often used on turns to realign the robot with the track.

A. Comparison: PID vs Proportional-Only Control

To highlight the advantages of using a full PID controller, it is useful to compare it with a simpler Proportional-only (P-only) control approach.

- **Proportional-Only Control:**
  - In this approach, the correction is directly proportional to the current error using the proportional gain K_p.
  - It is simple to implement and often sufficient for very basic control tasks.
  - However, it has notable limitations:
    - **Steady-state errors:** The robot may never perfectly align with the path due to lack of integral compensation.
    - **Zig-zag behavior:** High K_p can result in oscillations or instability.
    - **No long-term correction:** Drift or bias in the environment cannot be compensated over time.
- **PID Control:**
  - Adds integral and derivative components to improve performance.
  - The integral term eliminates long-term errors, and the derivative term predicts error trends to dampen the response.
  - Provides smoother and more stable control in real-time.
  - Suitable for dynamic environments where sensor noise and path curvature vary.
  - Ensures the robot converges to the desired path with minimal overshooting or oscillation.

In summary, while proportional-only control is simple and may work under ideal conditions, PID control offers significantly better stability, responsiveness, and accuracy for robust autonomous navigation.

B. Tuning PID Parameters

We spent a lot of time carefully tuning the PID parameters (K_p, K_i, and K_d) to achieve optimal performance. Although there is a lot of interplay between each variable, we used the following guidelines to help tune these parameters based on our observations:

- **If the robot zigzags too much:**
  - Reduce K_p slightly to decrease aggressive corrections.
  - Increase K_d to smooth movements.
- **If the robot takes too long to align with the path:**
  - Increase K_p to make corrections happen faster.
  - Increase K_i if steady-state errors persist.
- **If the robot overshoots the path frequently:**
  - Increase K_d to add damping to the corrections.
  - Reduce K_p if corrections are too strong and cause overshooting.
- **If the robot drifts slowly off track but doesn't correct:**
  - Increase K_i to help correct long-term errors.

These tuning guidelines helped in adjusting the PID controller for optimal navigation performance in real-time, ensuring the robot follows the path efficiently and smoothly.

VI. Traffic Sign Detection

This is an important challenge, because the successful detection and adherence to traffic signs, including stop signs and school zone is crucial for the safe operation of autonomous vehicles. To achieve these challenges, we identify the sign of interest, and either stop the robot for 3 seconds at a stop sign, or slow down in red school zone until the school zone ends, as indicated by the green sign.

A. Color Masking

The system defines specific ranges of hue, saturation, and value to isolate the colors corresponding to stop signs, red school zones, and green school zones. These ranges are used to create binary masks, where the regions matching the target colors are highlighted, while all other areas are suppressed.

- **Stop Sign Detection:** The system detects stop signs by applying color thresholding in the HSV color space to isolate red hues typically associated with stop signs. Specifically, two red ranges are defined: one for lower red hues (0–8 degrees) and one for higher red hues (160–180 degrees), with appropriate saturation and value thresholds to exclude noise. Binary masks are generated for each range to capture all red regions in the frame. Contours are then extracted from the combined mask. For each contour, the system approximates its polygonal shape and computes the area. A contour is considered a potential stop sign if it has six or more vertices (indicating an octagonal or near-octagonal shape) and a sufficiently large area (greater than 7500 pixels). Upon detecting such a contour, the system publishes a Boolean message to a ROS topic (`/stop_sign_detected`) to signal the detection. To prevent repeated detections of the same sign, an internal flag ensures that the message is only published once per stop event.

Fig. 3. Stop Sign

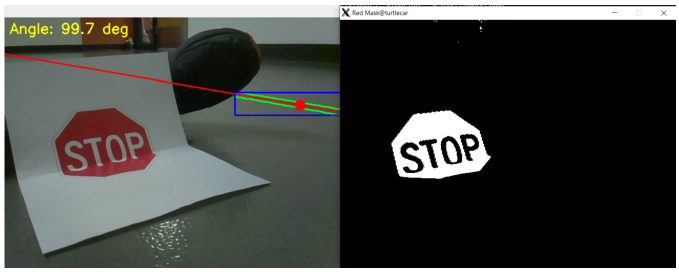

Fig. 4. Demo of Stop Sign Detection

- **Red School Zone Detection:** Similar to stop sign detection, red school zones are identified using color segmentation in the HSV color space. The system defines two red hue ranges—one on the lower end (0–10 degrees) and another on the upper end (160–180 degrees)—with appropriate saturation and value thresholds to detect darker shades of red typically used in school zone signs. These two masks are merged using a bitwise OR operation to create a single binary mask highlighting red areas in the image. Contours are extracted from the combined red mask. If any contours are found, the largest one is selected, and its pixel size is measured. A contour is considered a red school zone sign if its size exceeds 75 pixels. Upon detection, a Boolean message is published to the `/red_school_zone_detected` ROS topic. This message triggers the robot to enter a low-speed "school zone" driving mode until a green school zone sign is detected.

Fig. 5. Red School Zone Sign

- **Green School Zone Detection:** Similar to what we did for other signs, green school zones are detected using HSV-based color segmentation. The system isolates green hues typically found in the range of 35–85 degrees, along with suitable saturation and value thresholds to capture the vivid green shade used in school zone exit signs. A binary mask is generated by thresholding the HSV image for the defined green color range. Contours are then extracted from this mask, and the system checks for the largest contour. If the number of pixels in the largest contour exceeds a set threshold (e.g., 75 pixels), it is considered a valid green school zone detection. When detected, a Boolean message is published to the `/green_school_zone_detected` ROS topic. This informs the controller to exit the low-speed school zone mode and resume normal driving behavior.

Fig. 6. Green School Zone Sign

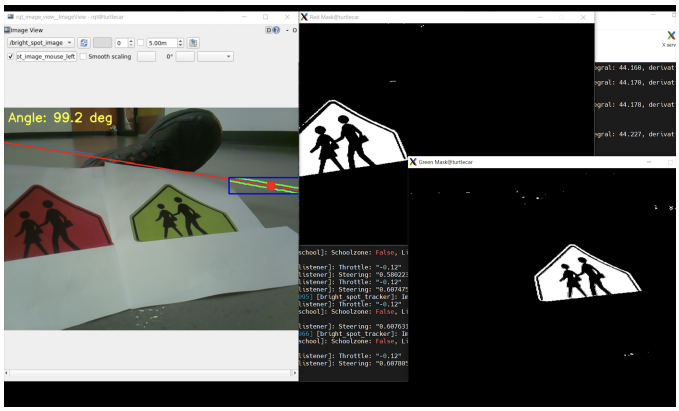

Fig. 7. Demo of School Zone Sign Detection

Once the signs are detected, we publish to the appropriate topics, allowing the controller to subsequently set state flags that affect its behavior.

If the stop sign is detected, the robot sends 0 velocity commands to stop the robot for 3 seconds before starting again. It only searches for a new stop sign after a sufficient amount of time has passed to avoid stopping at the same stop sign multiple times.

If a red school zone is detected, the robot enters a school-zone state, where it drives at a minimum speed until the green school zone sign is detected, upon which it returns to normal linear speed.

B. Challenges and Limitations

The navigation of the vehicle and the challenge problems introduce many difficult problems, particularly with respect to robustness on top of the accuracy, efficiency, and overall functionality. A hardware implementation is especially challenging to achieve, due to the variability in physical conditions and the robot's characteristics.

There is unique lighting in every part of the track, different colored and textured floors and walls throughout the track, and robot drift that is variable to the drive battery's charge and the tightness of the robot's screws and such.

To deal with these difficulties and achieve a robust solution, we spent many hours tuning the parameters (including gains, color ranges, and pixel thresholds) while also iterating upon the algorithms and solutions overall.

- **Wheel Drift:** A significant challenge observed during the vehicle's operation, especially when following a straight line, is the drifting of the wheels. Despite only providing linear velocity, the vehicle still tends to drift off course. This issue could be attributed to the mechanical characteristics of the wheels and the inability to maintain perfect traction, which results in slight deviations in the vehicle's path. The drift was particularly evident when operating at low speeds, where the wheels had insufficient grip to maintain a straight trajectory. To address this, we continually test how bad the drift is, and add a set angular velocity to all velocity commands to keep the robot straight, when it intends to.
- **Turning and Sharp Turns:** Another limitation arises when the vehicle attempts to navigate sharp turns. Since the tape is set at a right angle, there is no gradual curve for the robot to continue following all the way around the track. Therefore, the robot loses sight of the tape often. To accommodate this fact, we reverse at an angle when the occurs to pick up the tape again. We set different thresholds for the number of pixels in the tape contour to enter and leave this recovery period, which is reduces the amount of stop-and-go that occurs when correcting. Additionally, we apply a slight delay during recovery in which it won't reacquire the tape and start going forward. This of course had to be balanced with losing sight of the tape all together while reversing, but with enough testing and iteration this became consistent and improved the robot's performance dramatically.
- **Communication time:** A huge limitation, particularly at the turns, is the latency between the robot arriving at a turn, performing image processing, and realizing it should make the turn. This especially occurs at high speeds, which are necessary to complete the course on time. To address this, we tuned the buffers associated with publishers and subscribers, so that we balance the quality of service with the recency of data.
- **Color Detection Limitations:** The color detection system, although effective in isolating specific colors like red and green, faces challenges in real-world applications:
  - **Lighting Variability:** One of the primary challenges is the system's sensitivity to varying lighting conditions. Changes in ambient light alter the appearance of the target colors, leading to inaccurate detection or false positives/negatives.
  - **Color Confusion:** In some cases, certain colors might closely resemble each other, especially under non-ideal lighting. For instance, a school zone sign might appear similar to a traffic signal in certain lighting conditions, causing confusion in the color classification algorithm.
  - **Accuracy of Thresholds:** Setting appropriate thresholds for hue, saturation, and value is crucial for detecting the target colors. However, this is not a trivial task, as even small variations in color can lead to misclassification. The system may either fail to detect markers that are slightly outside the defined thresholds or incorrectly identify background elements as target markers if the thresholds are too broad.
  - **Detection of Blue Objects:** Another limitation of the color detection system is its occasional misidentification of blue objects within the robot's environment. If a blue object is within the field of view, the system incorrectly classify it as a target marker and cause the vehicle to follow it.
  - **Green Targets:** We spent a lot of time trying to solve the target finding solution, as our image

and control pipeline was well suited to do so. We experimented with simply prioritizing green contours over blue ones to determine where to drive. The problem however, came with the color of the floor and especially the glare. There was no range of color values to identify the targets from afar without mistakenly identifying the floor as a target sometimes as well. This inconsistency means a more sophisticated solution, which requires much more computation as well, would be necessary to solve this problem.

VII. Challenge Performance

A. Time Trial And Line Following

Our PID control for the line following challenge using the RGB camera is able to complete the course in roughly 5 minutes with an average lap time of 1 minute and 40 seconds. The primary barrier to a more efficient lap time is taking corners. If our speed is too high, the car will overshoot the corner, then spend several seconds backing up until it can detect the line again and recenter it in our camera. We found that if we weighted the centroid of the detected line to the right, the robot will initiate its turn early, better aligning itself and reducing the amount of time spent re-locating the line.

Another challenge we encounter semi-regularly is with the threshold values we use to detect the blue line. Occasionally, the wall cove along the hallways would be detected as the line, which would cause the robot to try to reset relative to the wall cove and get trapped. Some minor re-tuning of our HSV threshold values (using `[100, 150, 0]` for the lower threshold value, and `[140, 255, 255]` for the upper value) fixes this issue.

B. Sign Detection

Similarly when detecting signs (stop sign or school zone signs) our design variables need to be tuned based on the color of the sign, the angle of the camera, the height of the sign, and the distance at which we want to detect it. When tuned correctly, our algorithms preform flawlessly, but any change to the setup of the challenge would require a process of recalibration.

VIII. Conclusion

In this project, we implemented a system that leverages color-based detection for autonomous vehicle navigation by following blue-tape and detecting traffic signs. By utilizing the HSV color space, masking, OpenCV contours, and threshold values we were able to create a robust object detection framework that allowed the robot to navigate the track quickly and consistently using PID control.

The key achievements of this project include:

- Image processing to determine the location of the tape relative to the robot, and detect traffic signs.
- A PID control method that allows for smooth and fast traversal through the course.
- Adherence to traffic signs.

Many challenges were encountered throughout the development process, however, which were important points of learning for the team. primarily, these stem from the hardware implementation, which introduces variability on top of the inherent difficulties. For example, wheel drift had to be accounted for to maintain a straight path. Additionally, the image processing had to account for variations in the lighting conditions, shadows, glare, and even reflections, which would otherwise lead to false positives or missed detections. Therefore, the speed, control gains, color ranges, and thresholds had to be continually tuned for robustness and consistency.

Despite these challenges, our group created a successful system for autonomous navigation of the course with respect to both speed and consistency. The color-based detection and control worked very well under our pre-defined conditions, but in an unseen environment without a line to follow this is of course inconceivable for autonomous vehicle navigation. An improved system would take advantage of other sensors, including LiDAR, and more sophisticated algorithms including state estimation and mapping with SLAM, path finding algorithms, or CV using neural networks. This would also require increased computational resources, however, which is always a consideration for embedded systems and requires extensive testing and tuning to work well.

Therefore, we are proud of the work we accomplished, and hope to carry the lessons we gathered from this project with us to bigger and better projects in the future.

IX. Appendix

- **Attempted Challenge Features**:
  - 5 Points: Camera Navigation
  - 1 Point: School-Zone Detection, Stop Sign Detection, Control Approach Comparison
- **Group Contributions**:
  - **Florian Frick**: First, I created the orange paper detection/offset calculation and the PID controller for the hardware checkin. Then, I simply adapted the values to capture the blue tape instead of the orange paper for a rough solution. I worked extensively to tune this solution to work more consistently, and I helped implement turning via the strategy of reversing to realign with the tape. I adjusted the school zone sign detection algorithms to also use the contour methodology. I tried to implement target finding, but encountered too much noise from the floor and glare. I spent many hours tuning the color ranges, PID gains, pixel thresholds, wheel drift counter-action, pub/sub buffer size, and minimum/maximum linear velocity. Finally, I polished the report to accurately match our final robot implementation.
  - **Kirin Kawamoto**: Helped with the PID control law tuning, corner tuning methodology, and image detection integration with control law
  - **Nikhil Kishor Sawane**: Canny edge detection, stop sign detection (publishing node), red school zone

detection (publishing node), and green school zone detection (publishing node).

- **Sheetal Sharma**: Assisted with blue tape detection, line-following implementation, and hardware integration; also contributed to report writing and documentation.
- **Jay Warren**: Configured Ubuntu, installed ROS2, initialized and tested the ROS packages for the camera, LIDAR, and motors, and assisted with robot vision and sign detection algorithms (Canny Edge Detection, and Hough Transforms). Additionally provided assistance with the final report.
- **Github Repository**: https://github.com/Sheetal-CU/Advanced-Robotics-Project
- **Demo Videos**: https://o365coloradoedu-my.sharepoint.com/:f:/g/personal/kika3780$_{c}olorado_{e}du/EsGy2WGcTDZDmI-gHHqtxoQBZb9MdSOOIdyQCKZq2mtVfA?e=KJo7BY$

REFERENCES

[1] A. Formaggio, "Implementing the Hough Transform from Scratch," *Medium*, 5 Jul. 2018. [Online]. Available: https://medium.com/@alb.formaggio/implementing-the-hough-transform-from-scratch-09a56ba7316b

[2] A. Ekeany, "Detection and classification of speed limit signs," *GitHub*, 2020. [Online]. Available: https://github.com/Ekeany/Detection-and-Classification-of-Speed-Limit-Signs

[3] K. Brkic, "An overview of traffic sign detection methods," 201X. [Online]. Available: https://www.researchgate.net/publication/XXXXX

[4] "Canny Edge Detection," Google Docs. [Online]. Available: https://docs.google.com/document/d/1_gF5WvhMNWQ0cUtREA5oCZRjQZ2LfIqNLdieoTvMn9I/edit?tab=t.0