# Vision-based Autonomy for Self-Driving Cars

BJ Kim[1], Greg Lund[1], Matt Strong[1], Brad Hayes[1]

***Abstract*— There have been many advancements in regards to making vehicles autonomous. This requires for a vehicle to have the most reliable navigation controller possible. These controllers be implemented in many ways, such as utilizing vision, lidar, or creating a map of the track that is being tested on. We decided to take the approach of utilizing RGB cameras mounted on our Tesla Model 3 in the Webots simulation in order to get through a track. The front mounted RGB camera continuously checked where the lane markers were relative to the field of vision allowing for our vehicle to adjust the tires' angles as well as decelerate or accelerate to specific velocities dependent on how far off from the center of the frame the lane markers were. This vision based controller was tested on a given evaluation track as well as another evaluation track that we created with harsher turns and performed well on both evaluation tracks.**

## I. INTRODUCTION

With the rise of research in autonomous vehicles, there has been a rise in the need for a reliable navigation controller for these vehicles, which can be implemented in a number of ways. Some vehicles may decide to rely on lidar sensors to get a sense of how close it is to other objects nearby and navigate around obstacles accordingly, while others may use visual SLAM as a method of creating a map of the path to be navigated and follow the map. Our approach to the problem primarily focused on utilizing vision from RGB cameras in order to navigate around test tracks successfully.

We were given a Tesla Model 3 with front and back mounted RGB cameras inside of a Webot simulation for this project. Only the front mounted RGB camera was used in order to navigate around the given evaluation track as it visualized where the white lane markers were relative to the frame that the camera was seeing. By calculating the angle that the white lane markers were away from the center of the frame, we were able to determine whether or not the vehicle was turning or in a straightaway part of the track. This allowed us to adjust the vehicle's velocities accordingly, as we would not want to go into a corner at the same velocity as a straightaway. Naturally, we increased the velocity on portions of the track that the camera visualized as straightaways, and if the angle of the lane marker relative to the center of the frame crossed a specific threshold, then the velocity was lowered. This method allowed us to reliably turn corners in a smooth fashion as well as ensuring that the car was not tipping over from entering a corner at too high of a velocity.

We show, through experimental results, that our simple controller is robust to various turns and twists, while being able to maneuver around high difficult areas with speeds up to nearly 40 kilometers per hour. Our method is fast, cheap, and easy to understand. Additionally, we do not use deep learning by leveraging known factors about the environment.

*This work was not supported by any organization.

[1]Computer Science Department, University of Colorado Boulder.

## II. RELATED WORK

Vehicle autonomy using only onboard sensing is a widely researched topic, and various works have addressed the problem. For the purposes of this work, we have attached two separate literature reviews the address related works regarding autonomous vehicles – one regarding the ethics of autonomous vehicles, the other regarding deep learning in autonomous vehicle control.

## III. METHODS

### A. BASE CONTROLLER

We opted to develop a controller relying soley on camera data, without the need for including lidar point clouds. While lidars can help in the perception pipeline, we show an end-to-end, simple visuomotor policy that is computationally cheap and easy to deploy. We only make use of the front-end camera for all control purposes, making our implementation financially and computationally cheaper than it would be compared to using lidar sensors to map out the vehicle's surroundings.

Our camera-based controlled exploits the consistent coloring of the lane markings on the track to calculate steering and velocity commands. The raw image taken from the camera, seen below in Figure 1, is the only input to the controller. We decided to slightly increase the resolution of this camera from 128x64 to 256x128 in order to extract more concrete lane lines.

Fig. 1: The raw image taken from our car's front facing camera at the start of the course. Note that the resolution was increased to 256x128 in order to extract more reliable lane markings.

This raw camera image was then processed to extract lane line information. The first step in this image processing

pipeline was to convert the image to the HSV (hue, saturation, lightness) color space, as can be seen in Figure 2 below.

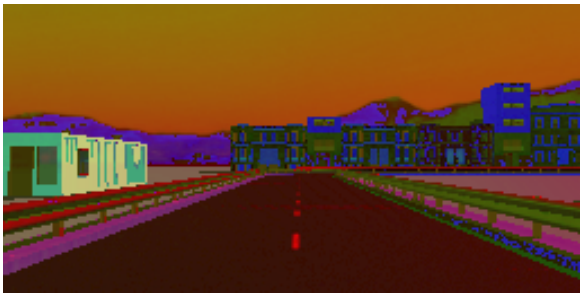

Fig. 2: The front camera image converted into HSV space. This allows for easy masking to extract pixels corresponding to the lane lines.

Given the image in HSV format, a color range was selected to mask out all pixels except for those corresponding to the lane markings. Through experimental trial and error, a good color range was selected, resulting in a mask as shown in Figure 3 below.

Fig. 3: The final mask including only the lane markings.

There are many possibilities for converting this mask into direct controller commands, but the method we opted for seemed the most intuitive. OpenCV's Canny Edge detector was applied to the mask, extracting only the outside edges of the lane markings [1]. It would be reasonably simple to fit a line to these edges via least-squares, for example, but instead we opted to use a built-in OpenCV function: the Hough Transform [2]. The Hough Transform takes each continuous (or mostly continuous) edge and fits a line to those pixels, returning the endpoint positions of the fitted lines. The angle from horizontal and relative x-position of each line could then easily be calculated and averaged. The angle and offset information for the average lane marking in frame could be fused to produce a single controller command, but we opted to do the simplest thing and only used the offset information. Matching intuition, if the lane lines are to the right of center in frame, the car should turn right to move towards the center of the road and the same goes for offsets to the left. This offset was therefore converted into a single steering angle command as input to the controller.

Due to inconsistencies in lane marking detection between frames and variations in lighting, setting the cars steering angle directly to the calculated command each frame would result in an extremely inconsistent controller. To remedy this, we utilized a simple P (proportional) controller that reduced the magnitude of desired steering angle outputs. The P value used was 0.55. Furthermore, in order to decrease the lap time, similar to human drivers, the car should accelerate on the straightaways and only slow down enough to reliably make it around the corners. The velocity command was set to be inversely proportional to the steering command, so that the car slows down enough to successfully and safely make it around sharp turns. This velocity command was given by:

$$v_t = \frac{39}{u_t + 1} \tag{1}$$

Where $v_t$ is the desired tangential speed of the vehicle at time $t$, and $u_t$ is the raw commanded angle *before* multiplying by the p gain. This was chosen to allow the controller to reduce more speed when approaching sharp turns. 39 was selected as the max speed from both our tracks.

There was minor tuning involved both for the proportional gain in the velocity (and steering) command, as well as for the braking intensity for the car. A value of 0.75 was settled on for the brake intensity after some experimentation, as it provided to be the best amount of damping the springs had for each rotational axis of each of the four wheels. With a lower braking intensity, the damping constant for the rotational axis would not be able to absorb enough of the downward force during the turns, resulting in the vehicle tipping over.

### *B. OBEYING STOP SIGNS*

Another challenge attempted was to extend our controller to detect and obey stop signs. Many methods were proposed and investigated in order to achieve this task, but all came short of producing a reliable controller except for the most basic approach.

Initially, we decided to use the built-in Recognition module in Webots, which could provide detection information for various objects in the simulation much like a standard object detection pipeline would in real life. After some experimentation with using a Recognition node on our front camera, it was evident that this method would not work. The recognition was very unreliable only detecting certain signs, giving large variations in bounding box size when it did supply detections, and would detect signs that were facing the opposite way more reliably than correctly facing signs. These issues combined made it almost impossible to use this strategy.

The next method considered was to load in a pre-trained image classifier to give bounding box predictions for stop signs in frame. We investigated using OpenCV's Cascade Classifier, YOLO and various pre-trained SVM models. For each of these cases, the overhead for running the classifier was very large, decreasing the real-time factor below 10%. Although we saw adequate results with the Cascade Classifier, we opted to not use this approach in order to preserve the runtime.

The final method attempted to take the same approach as the base controller and simply masked the raw image to only include the stop signs (or other red objects). This mask can be seen in Figure 4, below. We utilized OpenCV's HoughCircle's function to extract circles from the mask with a minimum radius of 4 and maximum radius of 8 [3]. If a valid circle exists, then we classify it as a stop sign. While the controller is running, if it detects a stop sign multiple times (at least 8 out of 10), we set a brake command by iteratively multiplying $b_i$, or the initial break value of 1, by $0.95$. The corresponding value, $b_c$, or the brake command, is then multiplied by the current speed to slowly reduce it down to 0.

Fig. 4: Front camera image and mask for a distant stop sign. Note that the yield sign also appears in the mask, but is ignored when we look for circles.

As the car approached a stop sign, the overall sign in the frame increased as can be seen in Figure 5 below. After the stop sign is detected, our stop sign controller began sending brake commands to the main controller slowly decrease the speed. This gave the effect of increasing the brake command as the car gets closer to the sign, ensuring that the vehicle would stop before the sign.

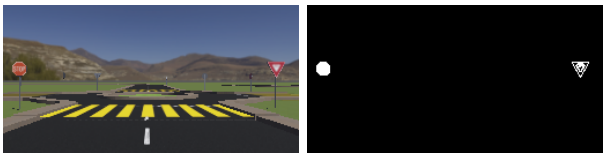

Fig. 5: Front camera image and mask when close to a stop sign. Note that the yield sign also appears in the mask, but is ignored when we look for circles.

### C. CUSTOM COURSE

Lastly, our navigation method was tested further by creating another evaluation track that our controller navigated on. This new course can be seen in Figure 6 below.

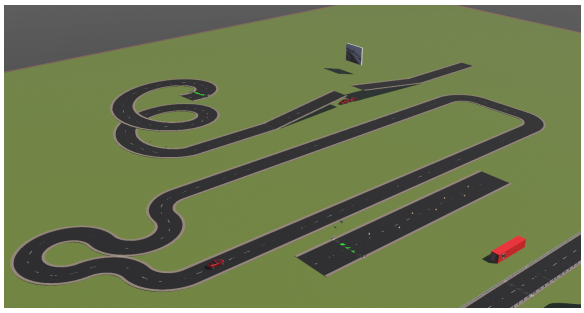

Fig. 6: The custom course made to fully test the limits of our controller.

Unlike the given evaluation track that has the most extreme turns being around 180°, which our controller has successfully navigated through, our new evaluation track has a turn that is nearly 270° to 360°. This tested the limits of our controller to see if we had the correct velocities. If the velocity the controller signaled to the vehicle was too fast during turning actions and too slow during the straightaways of the tracks, then we would not be able to get around the track quick enough. Thus, by including these harsh turns, we were able to see that the controller successfully signaled for a good velocity that would keep the vehicle from tipping over during the corners, but also would be making the turns quick enough so that we could maintain a well performing lap time. This new evaluation track also did not include guard rails, which also exhibited that our controller successfully utilized the front mounted RGB camera purely to mask just the white lane markers and follow only those lane markers. Therefore, even without guardrails, our controller will safely and reliably navigate a vehicle through the tracks. This would be applicable in the real world as well, as every road needs some sort of lane marker, meaning that our controller could be applicable outside of simulation with some changes to how the lane marker is followed.

## IV. RESULTS

We evaluated our controller on its ability to repeatably complete both the standard track the custom course and its ability to reliably stop at stop signs. This data is shown in tables I and II, below.

| Lap Number | Time [secs] |
|---|---|
| 1 | 40.0 |
| 2 | 39.2 |
| 3 | 39.2 |
| 4 | 39.3 |
| 5 | 39.3 |
| 6 | 39.1 |
| 7 | 39.2 |
| 8 | 39.1 |
| 9 | 40.3 |
| 10 | 39.1 |

TABLE I: Lap times for our car on the standard track. These times were taken in a single run, showing that the controller can do many laps with out any issue.

For the standard track, our controller had a mean lap time of 39.4 seconds with a standard deviation of 0.4 seconds

over 10 consecutive laps. We performed the same test for our custom evaluation track as well, which resulted in the following lap times.

| Lap Number | Time [secs] |
|---|---|
| 1 | 48.15 |
| 2 | 47.35 |
| 3 | 47.3 |
| 4 | 47.3 |
| 5 | 47.4 |
| 6 | 47.3 |
| 7 | 47.3 |
| 8 | 47.3 |
| 9 | 47.5 |
| 10 | 37.4 |

TABLE II: Lap times for our car on our custom track. These times were taken in a single run, again showing that our controller is robust and can handle long run times on this new course.

For the custom course, our controller had a mean lap time of 47.4 seconds with a standard deviation of 0.3 seconds over 10 consecutive laps. Both of these tests illustrate that our controller is robust and allows the car to complete laps on these courses in a reasonable amount of time.

## V. Discussion

Our vision based controller was able to complete the main and custom track extremely reliably, with consistent, desirable lap times. Our choice of the velocity function was intentional, as it forces the car to quickly slow down at tight turns, such as the difficult maneuver on our custom track, inspired by a human driver. Additionally, our P controller required minimal tuning to a point where heuristic methods like that of Ziegler–Nichols were unnecessary.

## VI. Conclusion

In this paper, we demonstrated a simple policy for controlling a self-driving car on a challenging course. Our approach, inspired by a human driver, is simple to deploy. For future work, we are interested in the real world deployment of our method, where with regards to sensing, only a cheap, low-resolution front-facing camera would be required. Future work includes relying on more features from images and applying optimized deep learning techniques to improve inference speed.

We envision the future of self-driving cars to be more and more influenced by camera sensing; similar to how humans are guiding with two RGB cameras in our eyes, we are hopeful that self-driving cars will follow a similar methodology.

## References


[1] “Canny edge detection.” [Online]. Available: https://docs.opencv.org/master/da/d22/tutorial_py_canny.html

[2] “Hough line transform.” [Online]. Available: https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html

[3] “Hough circle transform.” [Online]. Available: https://docs.opencv.org/master/da/d53/tutorial_py_houghcircles.html