Developing, Analyzing, and Evaluating Self-Drive Algorithms Using Electric Vehicles on a Test Course

Ryan Kaddis
Department of Math and Computer Science
Lawrence Technological University
Southfield, MI
rjkaddis@gmail.com

Enver Stading
Department of Mathematics
Nebraska Wesleyan University
Lincoln, NE
estading@nebrwesleyan.edu

Aarna Bhuptani
Department of Math and Computer Science
Vanderbilt University
Nashville, TN
aarna.h.bhuptani@vanderbilt.edu

Heather Song
Department of Statistics
The Ohio State University
Columbus, OH
heatherj.song@gmail.com

Chan-Jin Chung
Department of Math and Computer Science
Lawrence Technological University
Southfield, MI
cchung@ltu.edu

Joshua Siegel
Department of Computer Science and Engineering
Michigan State University
Lansing, MI
jsiegel@msu.edu

***Abstract*-- Reliable lane-following is one of the most important tasks for an automated vehicle or ADAS. The intent of this project was to design and evaluate multiple lane-following algorithms for an automated vehicle using computer vision. The implemented algorithms' performance was then evaluated on a testing course and compared with a human driver. ROS and OpenCV were used to detect and follow lanes on the road. A street-legal vehicle with a high-definition camera and drive-by-wire system was used to implement and evaluate driving data. Each algorithm was evaluated based on time for completion, speed limit infractions, and lane positioning infractions. The recorded evaluation data determined the most reliable lane-following algorithm. All of our algorithms had a success rate of at least 60% on certain lanes of the testing course.**

Keywords—Automated Vehicles, Lane-Following, Computer Vision, Robot Operating System (ROS), OpenCV

I. Introduction

The focus of this project is enhancing an automated vehicle's ability to follow lanes using lane-following algorithms. Lane following is one the most important and essential tasks an automated vehicle needs to accomplish reliably. Three different algorithms were designed for a street-legal, low-speed vehicle to achieve lane-following, even in harsh roadway and environmental conditions. Each algorithm was designed with ROS and OpenCV, and written in Python. The algorithms we wrote are Hough Line Detection, Line Detection with Spring Method Center Approximation, and Blob Line Detection with Shifted Line Following.

These algorithms must account for poor road conditions, sharp curves, and external marks, like parking lot lines. The course used to test and evaluate these algorithms is meant to reflect harsh roadway conditions. Many of the challenges faced during development were derived from using computer vision, and various techniques were employed to counteract each problem. [2] Similar research has identified problems when using computer vision to lane-follow. Such approaches use a Region of Interest (ROI) [3], perspective transformations [2], and image preprocessing techniques [2, 3] to remove unwanted noise and objects. Our approach for each algorithm uses a unique blend of image filtering and ROI cropping to obtain only the relevant portions of the camera view.

The best algorithm was determined by comparing the results of all three algorithms. The algorithms were also analyzed in comparison to a licensed human driver, to determine whether the algorithms can out-perform a human. The driving data proved that while humans were more successful at staying centered in the lane, the algorithms were able to avoid more speed infractions.

II. Environment

A. Mako Camera and Autonomous Campus Transport

ACTor 1 (**A**utonomous **C**ampus **T**ransp**OR**t, see Fig. 1) is built on top of a Polaris Gem e2 provided by MOBIS. Lawrence Technological University, DENSO, Dataspeed, Veoneer, SoarTech, and Realtime Technologies provided Dataspeed's drive-by-wire system, vision sensors, 2D and 3D LIDARs, GPS, on-board computers, and all other hardware. Width of the vehicle is 55.5 inches (141cm) and length is 103 inches (262cm). The Polaris Gem e2 has a top speed of twenty miles per hour, and a range of approximately twenty miles.

This material is based upon work supported by the National Science Foundation under Grant No. 2150096 and 2150096

2155-6814/22/$31.00 ©2022 IEEE
DOI 10.1109/MASS56207.2022.00101

To lane-follow, the vehicle relies upon image data taken from the Mako G-319 camera from Allied Vision. This camera has a resolution of 2064x1544 with a max frame rate of 37 fps at max resolution, and importantly, it has ROS support which made it ideal for our purposes [1].

Fig. 1. Image of the Polaris GEM e2 "ACTor 1" vehicle from the official testing day. Sensors on the roof were covered due to rain.

B. Environment

To test the algorithms, the research team tested on a track drawn in a parking lot on Lawrence Technological University's campus (see Fig. 2). It is a two-lane course, with an intersection where the car is programmed to stop at a yellow line before crossing it using a dead reckon turn. The course is meant to reflect poor road conditions found in real environments on highways, riddled with potholes and fading road lines. Many of the yellow parking lines have been blacked out to prevent the algorithm from detecting yellow, as it might think it is the yellow at the intersection and stop the car in the middle of a lane. The goal for each of the three algorithms is to make two laps around the course in both the inner and outer lanes, making sure to stop at the yellow line and then make the right or left turn and continuing for a second lap before stopping once more at the yellow line.

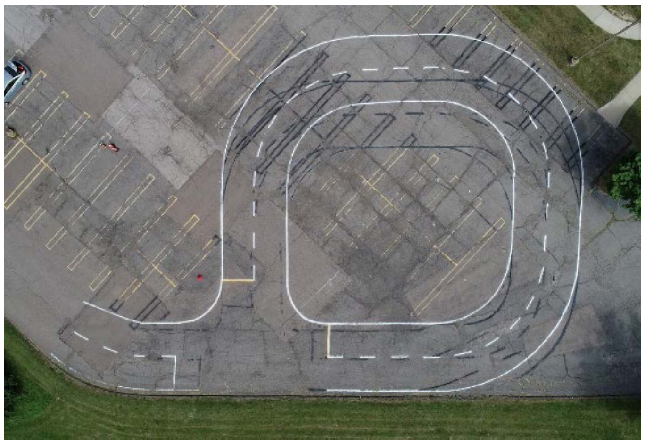

Fig. 2. Aerial Drone Image of Test Course. Image provided by Scott Lehman, LTU eLearning.

III. Methods

A. Software Architecture and Algorithm Commonalities

Each of the lane following algorithms will share the same software architecture for the sake of simplicity and modularity. Each algorithm will make use of two ROS nodes: the controller and yellow detection nodes. The controller node handles publishing control messages to the drive-by-wire system both during lane following and dead reckoning, during the turn at the intersection (see Fig. 2). In order to know when to switch from lane following to dead reckoning, and vice-versa, the controller subscribes to messages sent by the yellow detection node, which is responsible for detecting the yellow line at the beginning of the intersection. The yellow detection is accomplished by using an HSV mask and blob detection, which will determine whether a large amount of yellow is detected. Once the yellow line is detected, then leaves the camera's view, the controller will then use dead reckoning to pull forwards towards the line, stop for three seconds, then turn at the intersection. The turn at the intersection was done with dead reckoning, as there are no lane lines to follow. The way the lane following nodes interact with the controller is simple: the lane following node only has to provide a desired yaw rate, and the controller will package it properly and give the command to the drive-by-wire system.

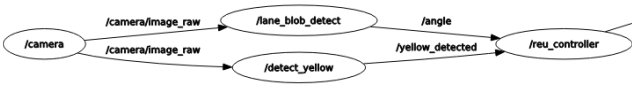

Fig. 3. ROS Node Architecture Diagram from rqt_graph. The lane-following node (top center) can be easily exchanged.

B. Probabilistic Hough Lines

The first lane-following method we will utilize is Probabilistic Hough Lines. Hough lines have been used in previous research over lane keeping and lane centering algorithms, so it is known that it is a viable solution [4]. The methods we utilize from the Probabilistic Hough Lines deviates from prior research: the method starts by using Computer Vision to get images from the road that is currently in front of the vehicle. By cropping for a region of interest (ROI) that only looks at the road, other external noise that the camera would normally pick up, like trees and the sky, is reduced. After cropping down the source image to only use the ROI, the image is then smoothed using a median blur function from OpenCV. Then the image is masked to only see white by using HSV to filter out colors and the Canny Edge Detection filter is applied. Canny Edge Detection is an OpenCV function that further reduces noise by only showing edges. The edges are found from gradient changes in pixels, non-maximum suppression, and thresholding [5] as shown in (1).

$$Edge\ Gradient\ (G) = \sqrt{G_x^2 + G_y^2}$$
$$Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right) \quad (1)$$

Following the Canny Edge filter, Probabilistic Hough Lines are used to find lines from the image produced by Canny Edge Detection. Probabilistic Hough Lines are also used in OpenCV and are found by using the parametric form for a standard line equation [6] shown in (2).

$$\rho = x\cos\theta + y\sin\theta \quad (2)$$

The lines found are then overlaid onto the ROI of the source image to visualize these transformations. In Fig. 4 one can see the image filtering and modification steps from the masked image to Canny Edge filtering, lastly to drawing Hough Lines.

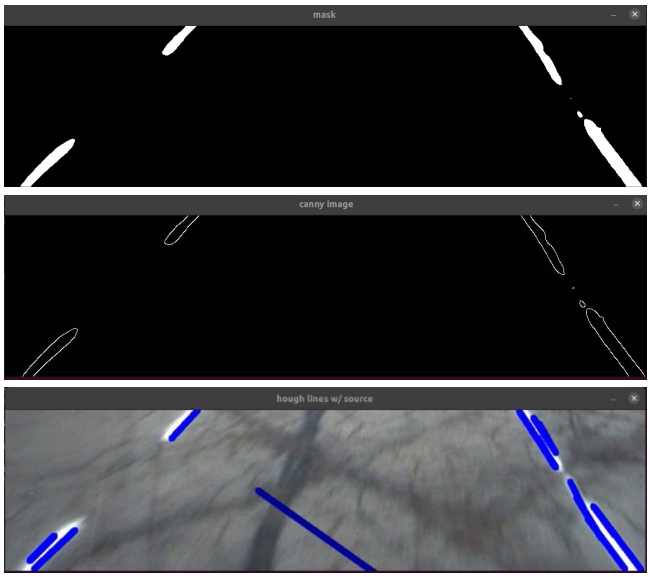

Fig. 4. Image of the steps Hough Line Detection follows. First, it masks for the lines, then applies a Canny detection, finally, the Hough lines.

After the preprocessing is done with an image, the vehicle still needs a position to move to or needs a direction in which to drive. For this self-driving algorithm, the slope of each line is calculated and if the slope is between the upper and lower thresholds the slope is summed with all other slopes from lines found in the image that also meet this condition. An average of the slopes is then computed and used to draw out a line originating from the center of the vehicle. The drawn line allowed for the calculation of a yaw rate to then be passed along to the vehicle to give it a direction to move while staying in the lane lines.

C. Canny/Hough Line Detection with Spring Method Center Approximation

Like the aforementioned algorithm, this algorithm will begin with the Canny/Hough line detection method. The method for this is to detect edges using Canny, detect lines using Hough line detection, filter those lines to only the lane lines, and then create a mask. First, the Canny color method is used to detect edges of the picture. Next, a Hough transform detects only lines that are near 45° and then extends these lines in case of broken or dashed lines, allowing the car to follow a solid line throughout. The challenges with detecting lines, and why we use these methods, are to get rid of the noisy data such as random lines around the road, the horizon, as well as if the lane lines are broken or dashed that could also create problems in the algorithm. Once we get rid of this noise, we are left with the extended lane lines that form an X, which we can simply crop to leave us with the 45° lane lines, and the spring method center approximation method can be implemented. The main goal of this algorithm is to push the center of the vehicle (C_v) to the center of the lane (C_L) using spring physics as a dynamic control model [1]. This works because when the vehicle is in the center of the lane, the x component of the push force of the spring is in equilibrium. For the force to actually be translated to steering input, the method is as follows. First, rays will be generated from the C_v point, and then the rays that intersect with the line mask must be identified. With the ray lengths, the force can be simulated as a push or pull force on the point C_L. For the last step, the steering input is calculated using the horizontal component of the force to shift the vehicle right or left to center it in the lane. [3] Some expected drawbacks of this algorithm are that the original [1] was implemented in C++ while the research team adapted/implemented it in Python [8]. The research team believes this algorithm provides a lot of promise as Canny/Hough line detection provides the program with two solid lines to follow consistently and the use of spring physics is more reliable in the long run than simply relying on camera data or a line-following algorithm.

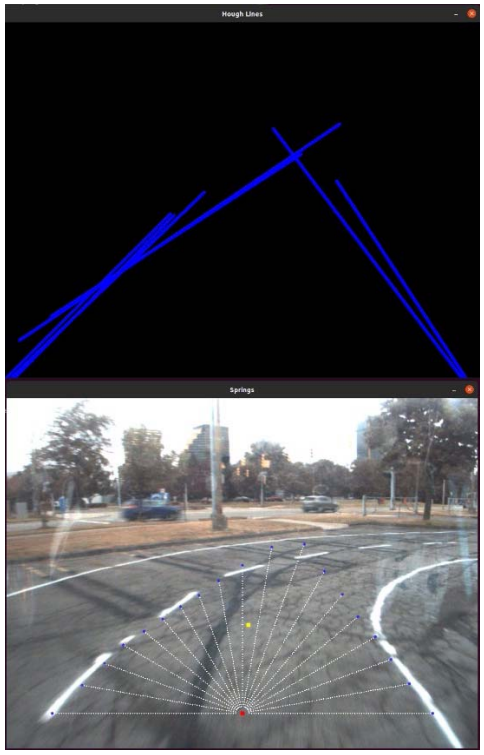

Fig. 5. Camera-view Visualization of Hough Lines and Spring-Center Approximation. The spring rays stop and "compress" at the lane lines.

D. Blob Line Detection and Shifted Line Following

The final lane-following algorithm we implemented is simpler than the prior two. The goal of all lane following algorithms is to approximate the center of the lane and to steer the vehicle towards it [7]. This can be accomplished by detecting only one of the lane lines, then driving the car while maintaining a certain distance from the detected line. Computer Vision is used to detect the line in real time by first masking and thresholding the image. The mask, as seen in the image below on the left, is accomplished by filtering pixels by their Hue, Saturation, and Value (HSV) levels. HSV filtering has proven to be more reliable than basic grayscale filtering, since HSV values can accommodate for different line colors and light conditions [1]. The mask will create a high contrast between the lane line and the road around it with a binary image (Fig. 6), which indicates that the program has successfully differentiated the line from the rest of the road. To isolate the line, blob detection in OpenCV is used to create a contour around the largest group of white pixels available in the masked image. A point is then placed in the center of the blob, as seen in the image on the right. Once the program is aware of the center of the detected line, the vehicle can then be commanded to try to keep that point in the same area in the image while in motion.

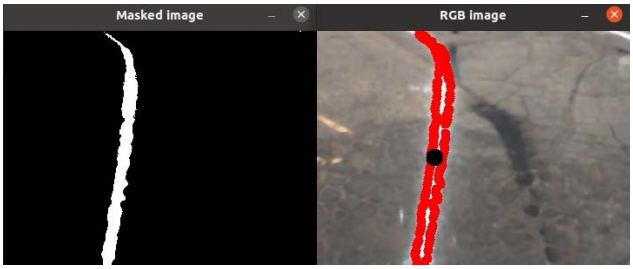

Fig. 6. Mask and Blob Visualization Windows. The red dots form a contour around the lane line, and the black dot is the calculated center of the contour.

IV. Challenges

The uniqueness of this research project derives from the numerous obstacles our team had to overcome to get our three algorithms to function during conditions such as different lighting, different road conditions, and distractions that would confuse our camera and mess with our algorithms. Considering people drive at all times of the day as well as on imperfect roads, the research team had to prepare for and overcome these challenges.

A. Shadows

While testing in the morning and in the evening, the course was overshadowed by the tree pictured below in Figure 7. Within this shadow, the camera was unable to detect the white lines, and the algorithm was unable to continue functioning. To fix this, the research team came up with many solutions. The most successful one being when the algorithm lost sight of the contours to continue at the same yaw rate. This was less successful on the outer lane than the inner as on the outer lane the vehicle approaches the shadow at more of a straight line, leading the vehicle to cross the white lines and continue for the sidewalk next to the tree. On the inner lane, however, the vehicle approaches at a curve, so the vehicle continues to follow at this yaw rate, leading the vehicle out of the shadow and back to the light where it can once again detect the white lines.

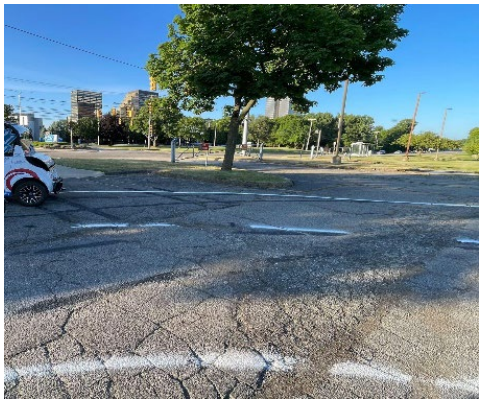

Fig. 7. Image of Tree Shadow Covering Portion of the Course. The shadow covers a large portion of the outer lane.

B. Road Conditions

The course was meant to represent harsh roadway conditions. Large portions of the lanes had potholes, cracks, and bumps, which interfered both with the vehicle's speed control, as well as the algorithms' vision detection. The lane lines were also left in a rough state. As the picture below in Fig. 8 shows, they are much thinner than a real road would be, and therefore were much harder to detect with our different algorithms. Many of the challenges occurred when the algorithm would mistake the yellow lines for the white lane lines and move the vehicle out of the lanes. Furthermore, even when the white lines were able to be detected, much of the paint was worn in some places and segments of the lanes' road surface were missing.

Fig. 8. Example of Poor Course Condition. Potholes and parking lot lines contributed to unwanted noise.

C. Environmental Conditions

Many environmental conditions, like lighting conditions and weather, greatly affected the reliability of our algorithms. During overcast conditions, the algorithms worked as intended,

as shadows and sunny conditions did not affect the filters we had in place and did not interfere with the Mako camera. During sunny conditions, however, the research team would have to adjust the filters and the parameters the code used to detect the white lines. During rainy conditions, the authors found that the algorithms, especially Hough line detection, did not work. The rain streaks across the window were confused for Hough lines, and the algorithms struggled to differentiate between lane lines and rain streaks, which led to calculations of incorrect yaw rates.

D. Camera

Our algorithms struggled in sunny conditions due to overexposure. The solution the research team found was to tape a sunglass lens to our camera and turn the exposure completely down, but many times even this did not suffice and our algorithms could not work properly as we solely relied on the camera to guide us through the course. As seen in Fig. 9, this is the camera view when no sunglass was attached versus when the sunglass was taped on for testing.

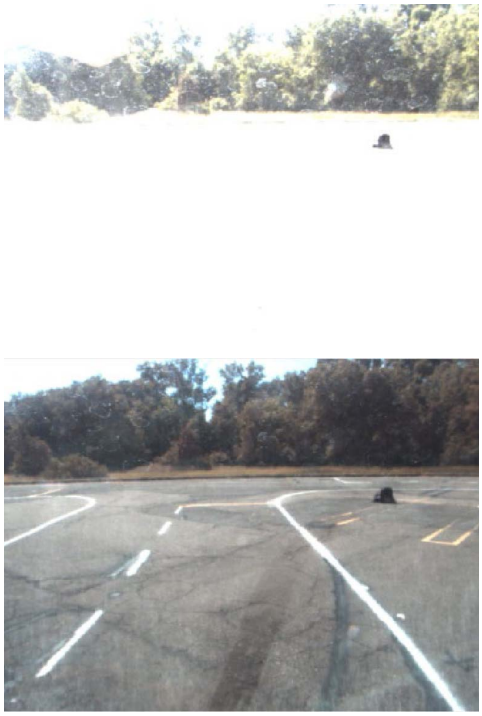

Fig. 9. Example of Camera View without lens (top) and with lens (bottom). Without the sunglass lens, line detection was impossible.

V. Analysis and Evaluation of Results

The table I below shows the recorded data for each successful run of each algorithm, along with data from a human driver. An evaluation program was used to collect the total time in seconds, average speed in miles per hour, and speed infractions of a successful run for each method. An external evaluator recorded the number of times the vehicle would either touch a lane line or drift outside the proper lane. On the official testing day, an evaluator would follow behind the vehicle and note at which points during the test the vehicle went out of the lane. The total number of recorded runs for each algorithm was used to determine the average success rate. A run was successful if it could complete two continuous laps on the course.

Table I. Results Data

Results Data	*Lane*	*Success Rate*	*Time (s)*	*Speed (mph)*	*Distance in Error (m)*	*Line Touches*
Blob	Inner	16.67%	160.42	2.164	2.337	5
	Outer	75%	200.44	2.211	3.197	1
Hough	Inner	10%	171.80	2.170	2.413	3
	Outer	62.5%	204.99	2.184	2.048	3
Spring	Inner	71.43%	164.90	2.171	2.740	3
	Outer	33.3%	195.59	2.188	3.162	6
Human	Inner	100%	63.17	5.843	46.280	0
	Outer	100%	80.98	5.738	18.148	0

The runs were processed further into a speed-time graph for visual purposes. A few of the graphs will be available here, and the rest will be available on the NSF REU site [9]. The graph below in Fig 10 details the speed over time for the blob method running on the outside, counter-clockwise direction of the course. The speed control for this algorithm is noticeably more consistent, especially in comparison to the human driver. The bumps in the graph are the result of the vehicle trying to make corrections for bumps and inconsistencies in the road.

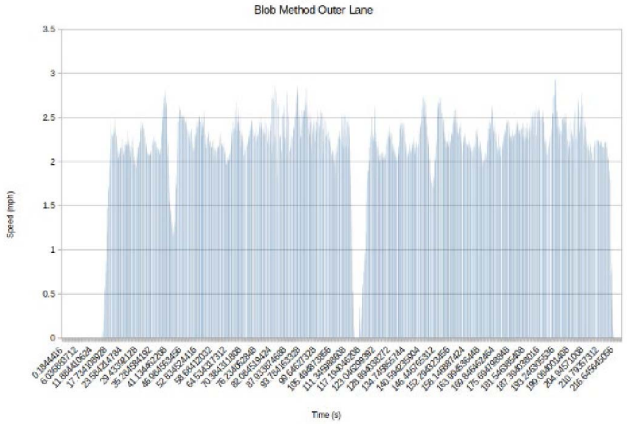

Fig. 10. Histogram of Driving Data from Blob Algorithm on Outer Lane.

The next graph in Fig 11 details the performance of the spring method algorithm on the inside of the course. In comparison to the previous algorithm, the speed control has more variance, which is the result of the Hough line detection. Hough lines are dependent on the accuracy of the HSV mask, which varies greatly depending on weather and light

conditions. The sharp peaks and troughs of the graph are the result of losing a Hough line in the mask, then picking the line up again.

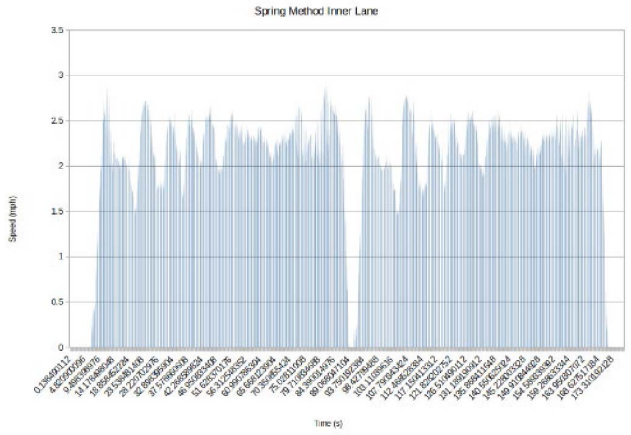

Fig. 11. Histogram of Driving Data from Spring Algorithm on Inner Lane.

On average, a human driver was able to drive faster than the algorithms, near the set speed limit of 7mph. The human would often exceed the speed limit, then, once the infraction was noticed, overcorrected and went far below 7mph. Humans, while able to follow a lane easier, struggled greatly with speed control when compared to the automated drive-by-wire system.

VI. Concluding Remarks

The purpose of this research is to introduce three working algorithms in which the autonomous vehicles can follow the course in both the inner and outer lanes. The data collected for each algorithm show no obvious outliers, and the graphs for each algorithm show that the ACTor vehicle traveled at an overall consistent speed during both laps. We tested these vehicles at the fastest speed they could safely achieve while maintaining the most accuracy possible. In the end, all three algorithms were able to complete the course for two laps at one point during testing and demonstration. Unfortunately, not all the algorithms work equally as well and as accurate for the inner and outer lanes. The most accurate algorithm for the inner lane is the Line Detection with Spring Method Center Approximation, and the most accurate algorithm for the outer lane is Blob Line Detection with Shifted Line Following. The blob and spring methods had the highest success rate in their respective lanes, and had the fewest line touches/departures. Those algorithms were also able to complete the course with a higher average speed. A high average speed means that the vehicle did not have to slow down as much for turns, and paired with the minimal amount of line touches, indicates good speed and centering control.

Many of the lane-following algorithms that currently exist are designed for smooth, well-indicated roads. [2, 3] Our algorithms were able to traverse the test course, despite the many challenges, like the sharp turns and poor road conditions. Computer vision-based lane-following has not been tested under these conditions, and we concluded that our algorithms form a baseline for navigating difficult stretches of road. The blob method, while it boasted the highest success rate, struggled on sharper turns, namely, the inside lane. Blob line detection had a major issue with sharp corners; it required that the left line be visible, and it was not always within the camera view. The spring method, while not as smooth of a ride, was able to handle all types of roadway sections more reliably, since it can use both lines to follow the road.

In the future, we hope to develop the algorithms to allow the vehicles to travel at a faster speed and faster time, preferably as fast as the human data, and allow the vehicles to provide consistent data regardless of the weather and lighting conditions. We also hope to develop more accurate algorithms so that the vehicles avoid touching lane lines and stay within the given space. The evaluation data files of the algorithms driving were kept for further research in the future.

Acknowledgments

We would like to thank our mentors, Dr. Joe DeRose and Prof. Nick Paul, as well as our teaching assistants, Mark Kocherovsky, and Joe Schulte.

Funding

This material is based upon work supported by the National Science Foundation under Grant No. 2150292 and 2150096.

References

[1] Paul, N., Pleune, M., Chung, C., Warrick, B., Bleicher, S., & Faulkner, C. (2018). ACTor: A Practical, Modular, and Adaptable Autonomous Vehicle Research Platform. 2018 IEEE International Conference on Electro/Information Technology (EIT), 0411-0414.

[2] Haque, M. R., Islam, M. M., Alam, K. S., Iqbal, H., & Shaik, M. E. (2019). A computer vision based lane detection approach. *International Journal of Image, Graphics and Signal Processing*, *10*(3), 27.

[3] Y. Xing *et al*., "Advances in Vision-Based Lane Detection: Algorithms, Integration, Assessment, and Perspectives on ACP-Based Parallel Vision," in *IEEE/CAA Journal of Automatica Sinica*, vol. 5, no. 3, pp. 645-661, May 2018, doi: 10.1109/JAS.2018.7511063.

[4] Berriel, R. F., de Aguiar, E., De Souza, A. F., & Oliveira-Santos, T. (2017). Ego-lane analysis system (elas): Dataset and algorithms. *Image and Vision Computing*, *68*, 64-75.

[5] *OpenCV: Canny Edge Detection*. (n.d.). OpenCV. Retrieved July 25, 2022, from https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html

[6] *OpenCV: Hough Line Transform*. (n.d.). OpenCV. Retrieved July 25, 2022, from https://docs.opencv.org/3.4/d3/de6/tutorial_js_houghlines.html (accessed July 27, 2022).

[7] Chan-Jin Chung, A Simple Lane Following Algorithm Using A Centroid of The Largest Blob, NSF Self-Drive REU 2022 Workshop at LTU, https://www.robofest.net/AutoEV/lanefollowing_algo22chung.pdf (accessed July 27, 2022).

[8] Nick Paul, Minimal python implementation of blob lane following algorithm, https://github.com/nick-paul/lane_follow_blob (accessed July 30, 2022)

[9] Team Star Final Presentation Slides in PDF, https://www.ltu.edu/uploads/media/arts-sciences/TeamStar22SelfDriveSlides.pdf (accessed July 30, 2022).