NSF REU: Self Driving Vehicles

Alexander Quezada, Seth Rodriguez, Shika Rao, Cebastian Chinolla

Introduction

We are developing, analyzing, and evaluating self-drive algorithms using real street legal electric vehicles in real-time on a predefined environment based on line following and lane centering algorithms.

Novelty

1. Lane following algorithms for very sharp turns under all weather conditions (further elaborated in challenges).

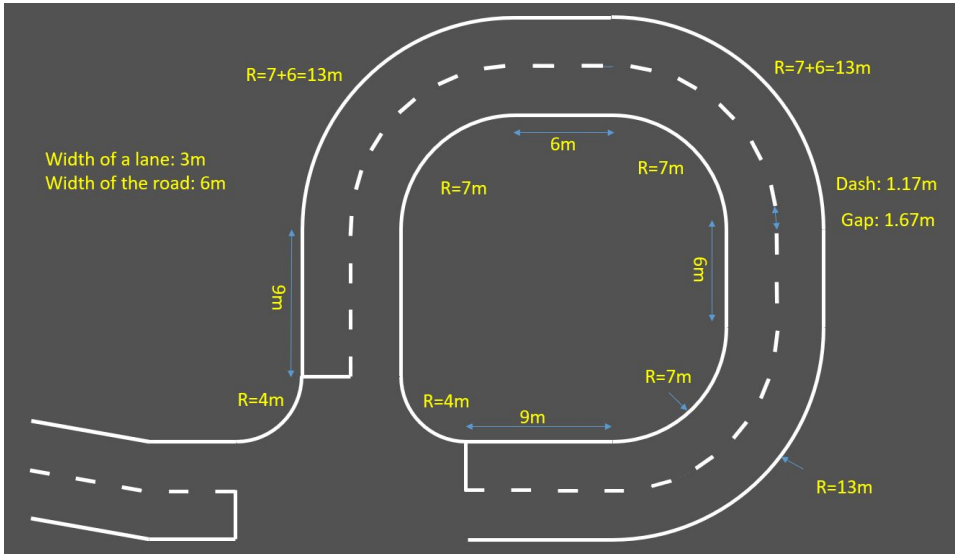
2. Testing the algorithms on a real self driving electric vehicle and course.

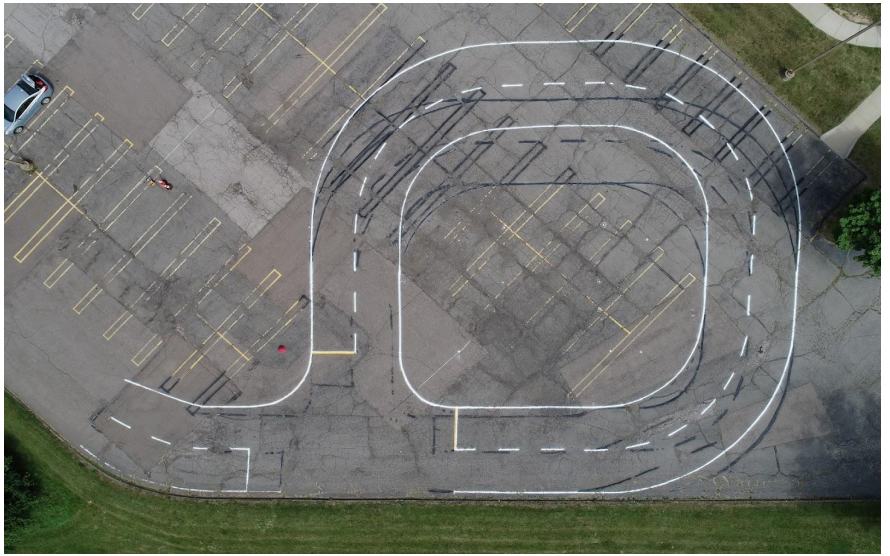
3. Combining the contour blob detection and Hough line methods to develop an algorithm for lane following.

Testing Equipment

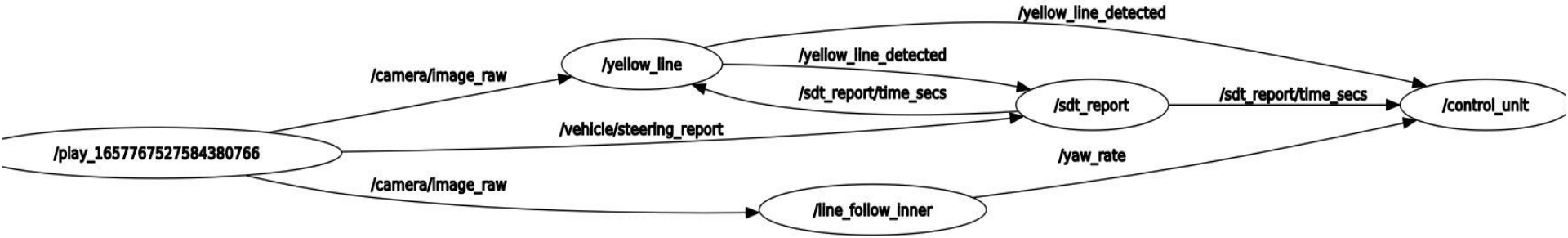
- Modified Polaris Gem 2
  - Max speed of 20 mph
  - Range of 20 miles
  - Dataspeed drive-by-wire system
  - 150 in. turn radius
- Mako G-319 Camera from Allied Vision
  - 2064 X 1544
  - 37 fps
  - ROS support

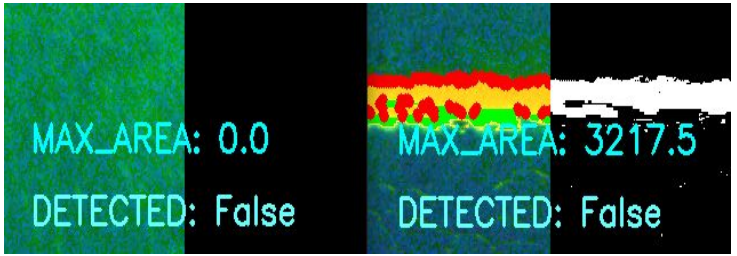
Environment





Code Architecture





Filters & ROI

Due to the difficult course and lighting conditions, we were faced with a cluttered and noisy image that made it difficult to detect lane lines.

The colors which we were interested in detecting were white and yellow. This was done by using an HLS color space to “select” only those colors into a mask.

After filtering, we were left with an image that contains anything either white or yellow, however the image remains noisy. This problem was minimized as much as possible through the implementation and modification of OpenCV filtration functions such as canny, in range, and a custom, adjustable white balancing filter.

In addition to filters, a Region of Interest was added to the image in order to crop out any unnecessary visual information. This was done through the utilization of a numpy array which was used to plot a polygonal “mask”, further boosting efficiency.

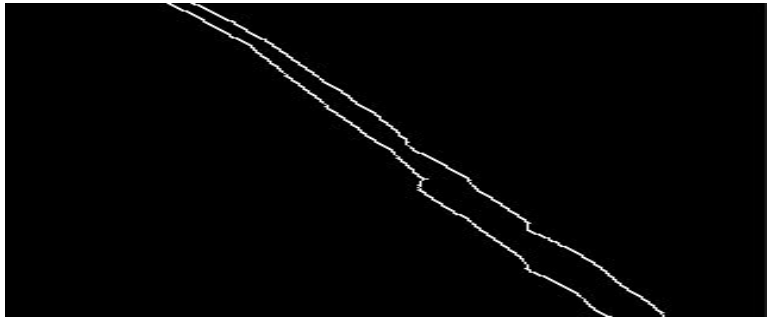


Fig 1. Image after being filtered

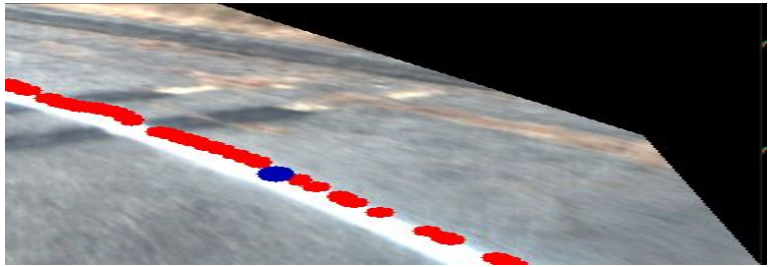


Fig 2. Image with ROI applied

Filters & ROI

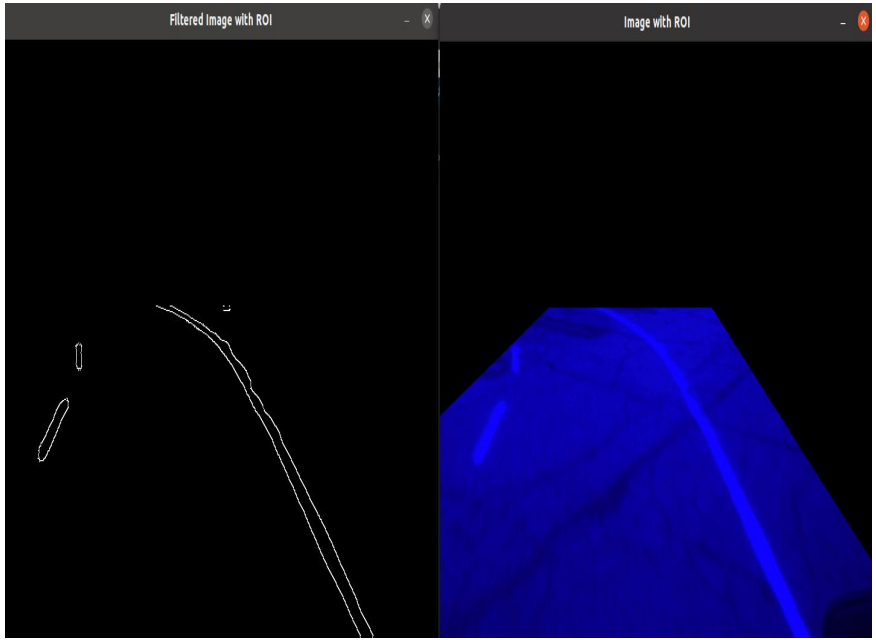


Fig 3. Image with custom ROI and filters

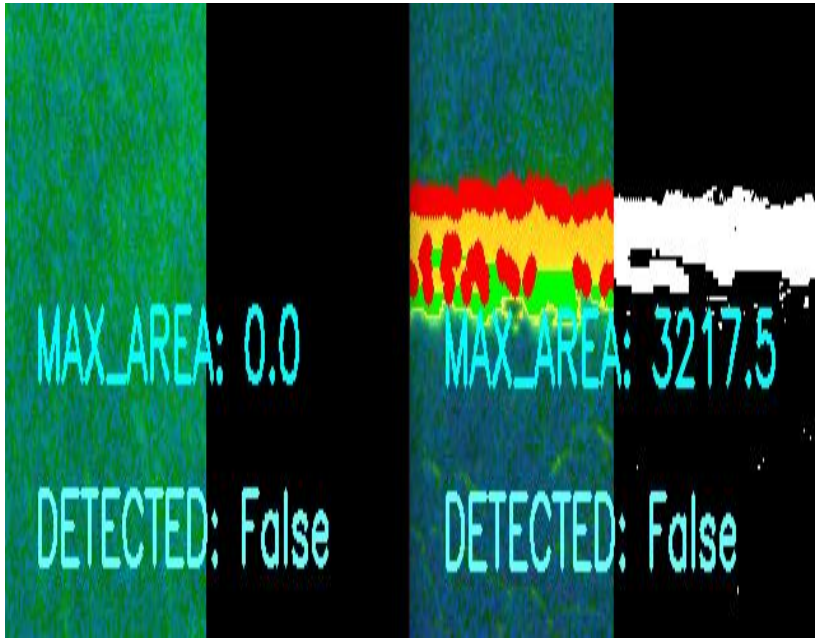


Fig 4. Yellow detection mask

Algorithm 1: Blob/Shifted Line Following

Our first algorithm was the simplest but ultimately allowed us to familiarize ourselves with the environment and the challenges that would be present throughout the course of testing.

This algorithm utilizes a proportional turn rate that is calculated by finding the centroid of the largest contour (generated from lane lines and the center of the camera), an ROI(Region of interest), and functions from OpenCV Library to navigate the course.

Through rigorous testing we found that although this algorithm is reliable at low speeds and with smooth curves, once the speed increases and turns get sharper, the accuracy of the algorithm decreases.

This algorithm was used as a base for our other 2 algorithms

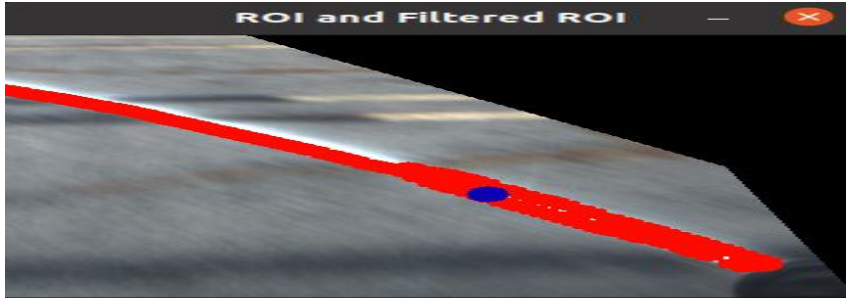


Fig 5. Image showing Algorithm 1 on the outer lane of course

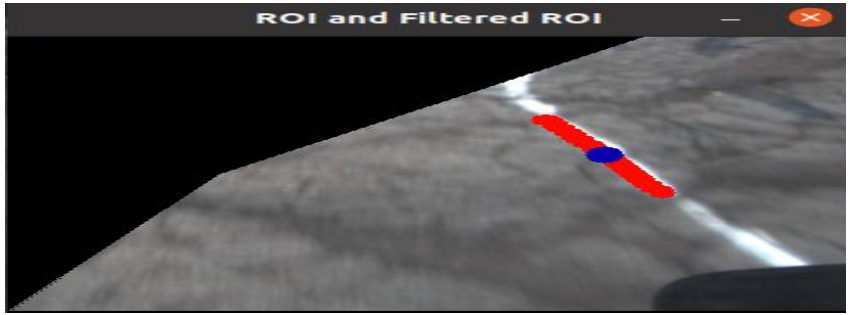


Fig 6. Image showing Algorithm 1 on inner lane of course

Algorithm 1: Blob/Shifted Line Following

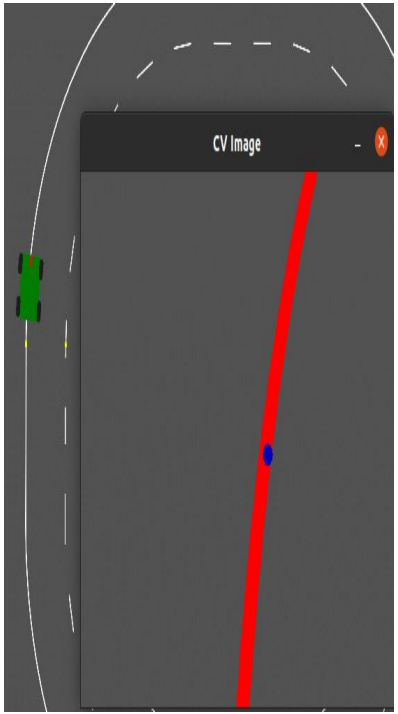


Fig 6. Shifted line following in 2D Simulation

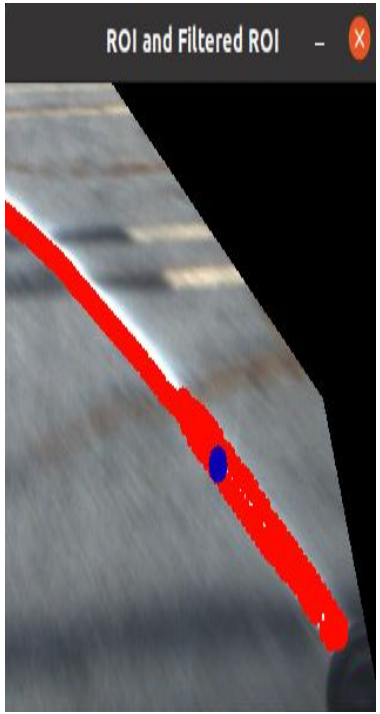


Fig 7. Image showing Algorithm 1 on the outer lane of course

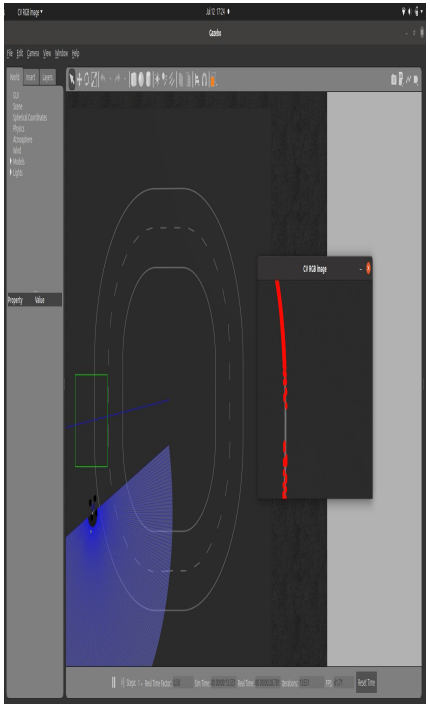


Fig 8. Image showing Algorithm 1 in Gazebo Simulator

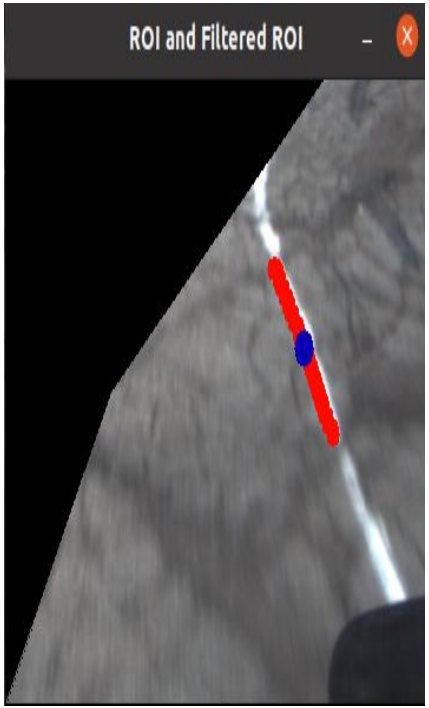
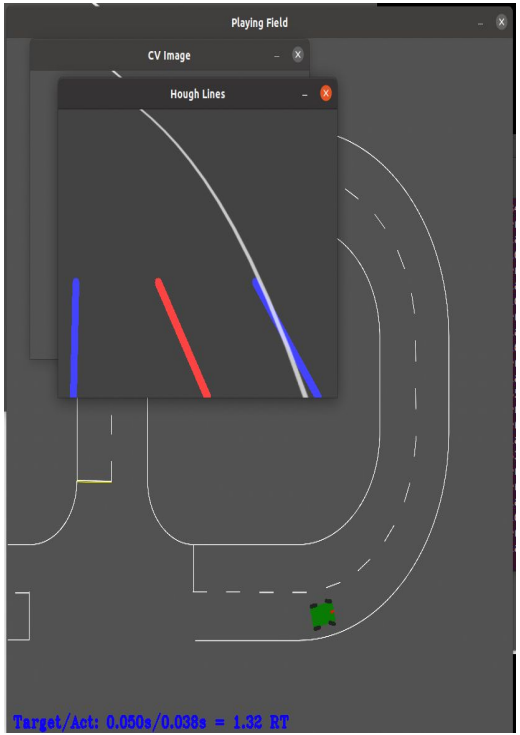


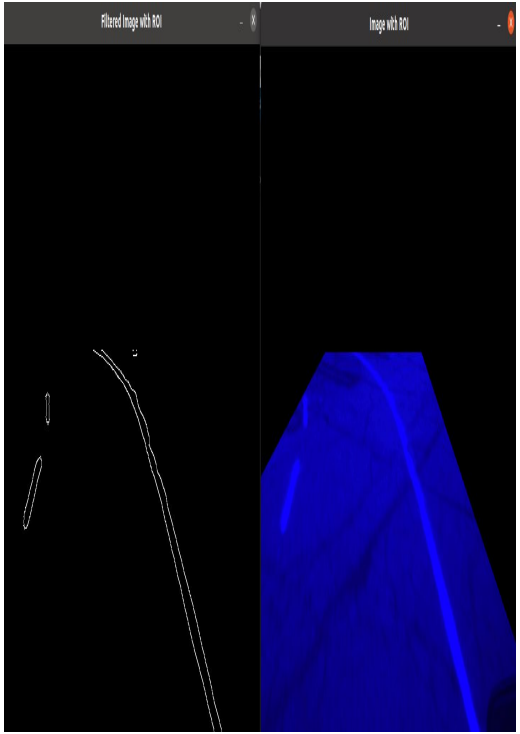
Fig 9. Image showing Algorithm 1 on inner lane of course

Algorithm 2: Shifted Hough Lines

	Line Detection	Lane Centering	Proportional Yaw Rate Control	In Simulation	In Real-Time Environment (When the sunlight is not too harsh)
Algorithm 2	Hough Lines, then Contour	Average Center Lane Line	Using the x coordinate value from centroid	Worked, but jitter present even at low speeds.	Could not do the sharp turns, lane departures present.
	Hough Lines	Average Center Lane Line	Using x coordinate value from average center lane line	Worked better than above in simulation. Slight jitter even at low speeds.	Could not do the sharp turns, lane departures present.
	Hough Lines	Offset	Using x coordinate value from the offsetted center lane line	Worked well. Jitter present at high speeds.	Works, but requires adjustment to find an equation to calculate the proportional yaw rate.
	Hough Lines, then Contour	Offset	Using the x coordinate value from centroid	Worked well. Jitter present at high speeds.	Works. Algorithm being used for demonstration. Fastest and smoothest algorithm. Jitter present in real life only at higher speeds.

Algorithm 2: Shifted Hough Lines





Algorithm 3: Blob/Spring Lane Following

This algorithm was designed by Nicholas Paul, we modified his code to successfully operate on our course and then ultimately implemented his algorithm onto the ACTor platform.

The spring lane following utilizes a unique approach to the problem which entails generating rays(springs) based upon image date which ultimately “push” the car based on a weight.

Upon testing we found this approach to be reliable, however the speed it was able to operate at was significantly less than that of Algorithm 2(Shifted hough lines)

Fig 10. Spring algorithm

Evaluation of Algorithm Efficiency

Parameter measured	Outer/Inner	Algorithm
Fastest Algorithm	Outer	Hough @ 6.7 mph
	Inner	Hough Spring
Smoothest algorithm (Less jerky movement)	Outer	Hough
	Inner	Hough or Spring
Best algorithm overall	Outer	Hough
	Inner	Hough and Spring
Most reliable algorithm (Depending on the success rate)	Outer	Hough
	Inner	Hough

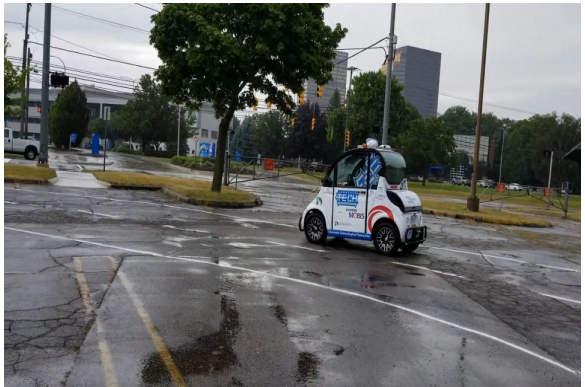
Successes during Official runs only

Algorithm	Inner/Outer	No. of Line Touches	Location of line touch	No. of Lane Departures	Location of lane departure	Yellow Stop Line Violation (yes/no), (before/ after)	Dead Reckon Turn Violation (yes/no), (left/ right)	Best time taken to complete 2 laps (s)	Average speed for both laps (mph)	Distance covered above or below speed limit (m)	Weather Condition
Blob	outer	1	1 dead reckon lane touch	0	NA	Yes, after	Yes, left				Overcast
Blob	inner	3 in lap 1 4 in lap 2	All turns in both laps and 1 dead reckon lane touch	0	NA	Yes, after	Yes, left				Rain
Hough	outer	0	NA	0	NA	Yes, after	No				Rain
Hough	outer	0	NA	0	NA	No	No				Rain
Hough	inner	0	NA	0	NA	Yes, little after	No				Sunny
Hough	outer	1	1 dead reckon lane touch	0	NA	Yes, after	Yes, left				Overcast
Spring	inner	1	1 dead reckon lane touch	0	NA	Yes, after	Yes, left				Sunny
Spring	outer	infinite	everywhere	0	NA	Yes, after	Yes, left				Overcast

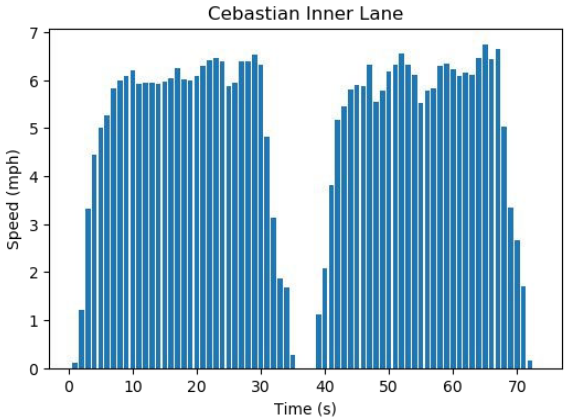
Fails during Official runs only

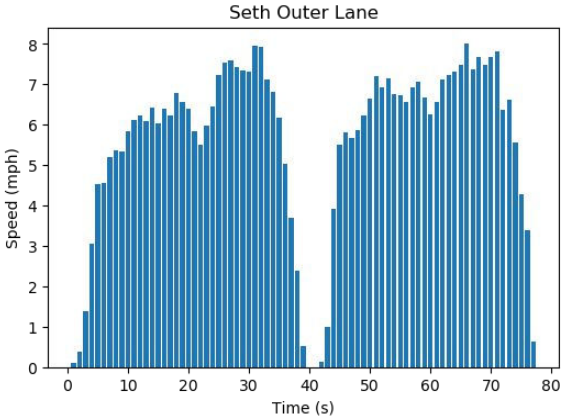
Algorithm	Inner/ Outer	Average Speed till stop	Location of Failure	Weather Conditions	Reasons for Failure	Remarks
Hough	outer		Turn number 3 (lap 2)	Overcast, Sunny	Overcast weather suddenly turned to sunny and the camera couldn't adjust fast enough.	Was going at very high speed too. Around 6.7 mph when it failed.
Spring	outer		Dead Reckon	Overcast	Did not do the dead reckon before lap 2 properly.	Had an infinite number of lane departures and line touches. Was “line following” the middle dashed line.

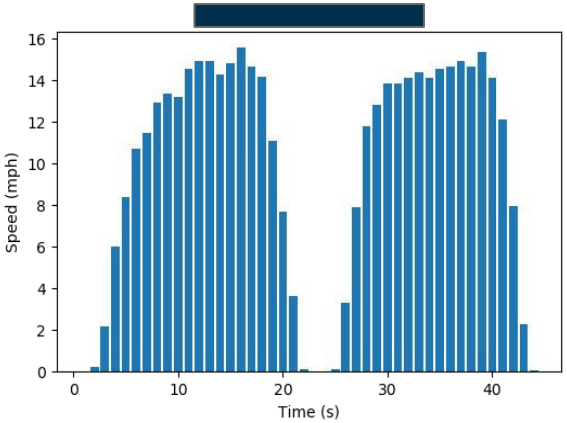
Challenges



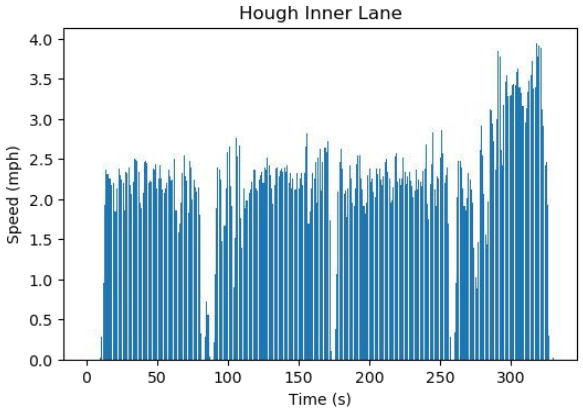
Best & Worst Human Drivers

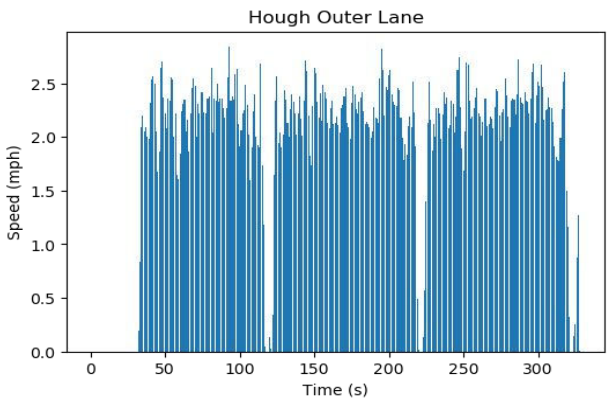


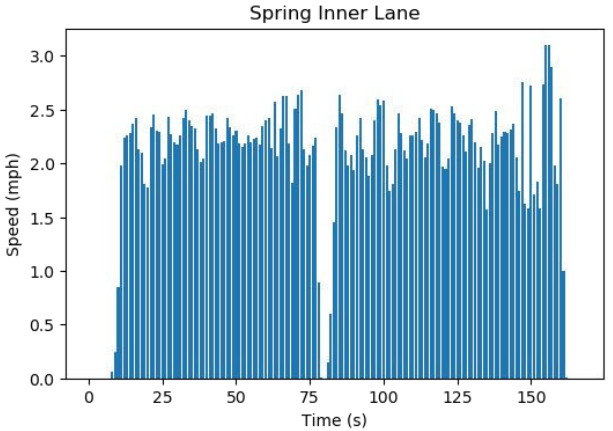




Best Algorithm graph







Closing Remarks

We proposed and implemented three different lane-following computer vision based algorithms which are tested on an electric vehicle in a controlled testing environment.

The real-time testing environment had sharp curves, poor road conditions, unclear and narrow line markings, and dynamic lighting conditions. These algorithms have been optimized to work under these conditions.

We evaluated human driving and the algorithms for the self-driving vehicles using a custom performance evaluation function, and then analyze and compare the human driving data with self-driving data under a specified speed limit using reports from the vehicle's drive-by-wire system.

Ultimately our most reliable algorithm overall ended up being our Hough lines implementation.

https://docs.google.com/document/d/1H9cp9qbUnU9kJjzTXFCzGNyd_w9aPVALcqfhVUAfGdE/edit