Evaluating Low-Resource Lane Following Algorithms for Autonomous Vehicles

Beñat Froemming-Aldanondo, Tatiana Rastoskueva, Michael Evans, Marcial Machado, Anna Vadella, Rickey Johnson, Luis Escamilla, Milan Jostes, Devson Butani, Ryan Kaddis, Chan-Jin Chung, Josh Siegel

Purpose

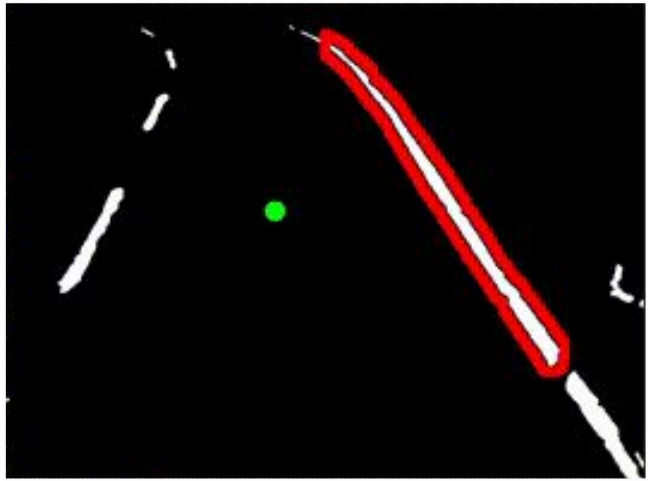
- Reliable lane following is essential for automated vehicles, but they often require extensive computational resources
  - According to estimates released by Nvidia in 2017, a fleet of 100–125 vehicles can generate 203–595 PB of raw data per year, resulting in 104–487 TB after preprocessing
  - Training deep models on these data volumes can take upwards of 113–528 days on a single NVIDIA DGX-1, necessitating 97–1,056 machines to achieve a 7-day target
  - Smaller autonomous platforms (e.g., mobility aids, golf carts, and delivery robots) can't afford those computational resources

Overview

- We designed five low-resource lane-following algorithms designed for real-time operation on vehicles with limited computing resources using Python OpenCV library and ROS.
- They were tested both on the simulations and real drive-by-wire vehicles on Parking Lot H of Lawrence Technological University.
- The way our algorithms function is by detecting the lane center and then using that information to remain within the lane.

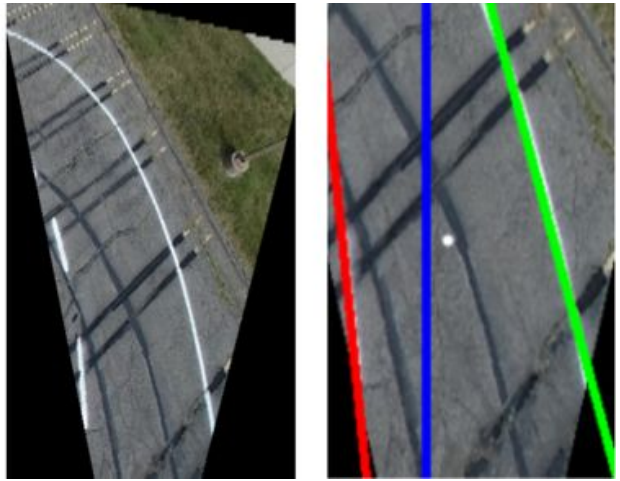
Largest White Contour

- Detects the largest contour - which we assume to be the largest contour
- Then we compute the offset to this contour, which we consider the center of the lane
- Sensitive to noise and offset value



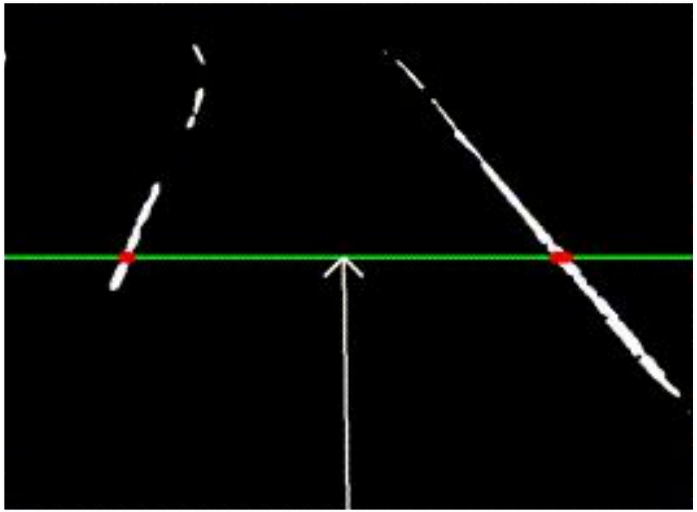
Birds-eye-view with Least Square Regression line

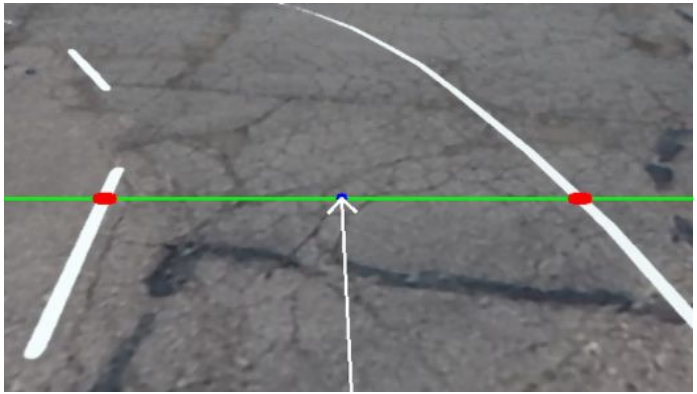
- Change the image perspective to bird's eye view, which results in lanes being parallel and straight
- Represent the lines as two least squares regression lines and compute the middle point between them
- More robust than a single-line approach, but outliers may still affect the result



Linear Lane Search with K-Means

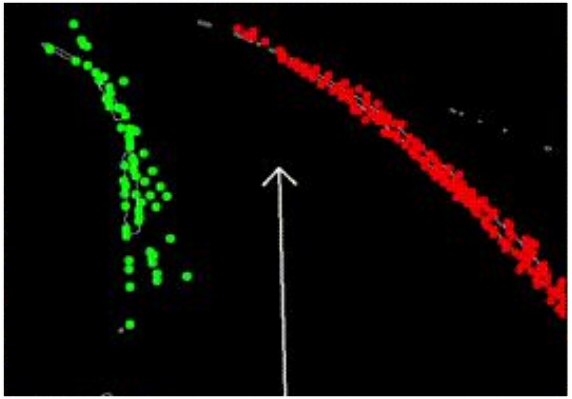
- Uses unsupervised learning
- Detects 2 clusters on a line and finds a midpoint between them
- If one or both lane lines are missing, the method reuses the previous frame's centroids
- Algorithm is computationally efficient, but sensitive to noise





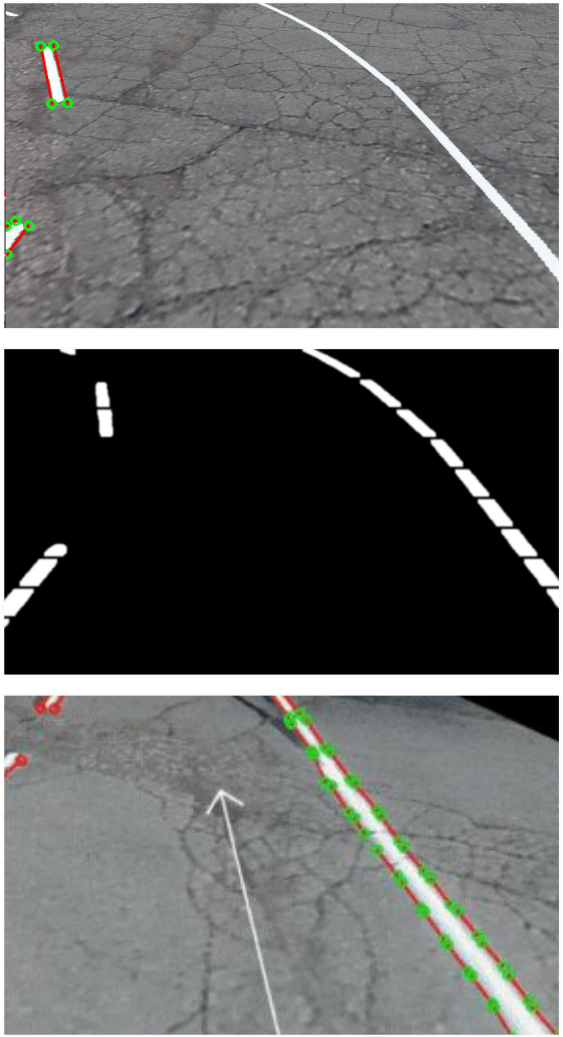
Lane Line classification using DBSCAN

- Uses unsupervised learning
- Separates dense groups of points from outliers, making it a good fit for lane line clustering
- DBSCAN separates lane lines more effectively
- Less sensitive to noise, but more sensitive to parameter settings



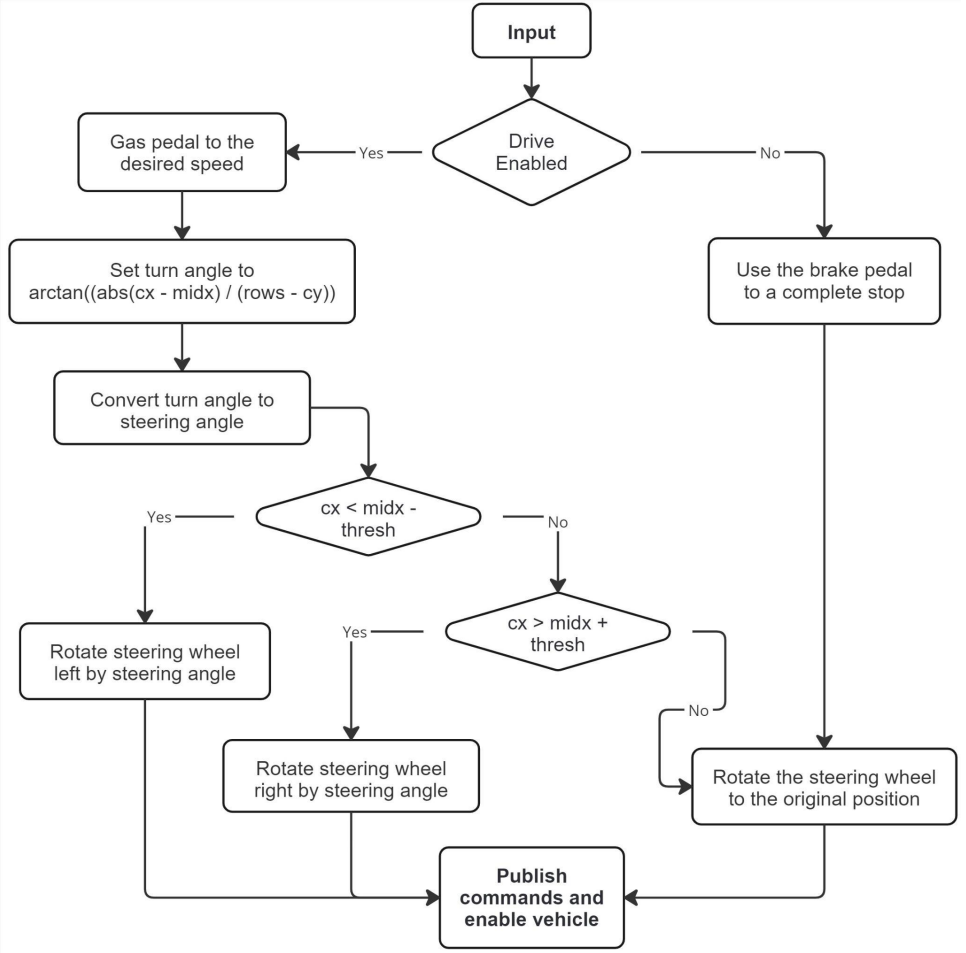
DeepLSD Lane Detection

- Deep learning-based line detector, DeepLSD
- DeepLSD doesn't need to be retrained like other open-source lane detectors
- DeepLSD can't detect curved lines
- Draw horizontal lines into the image to convert curved lines into short straight segments
- Inference time is slow (~5 images processed per second) on our laptops



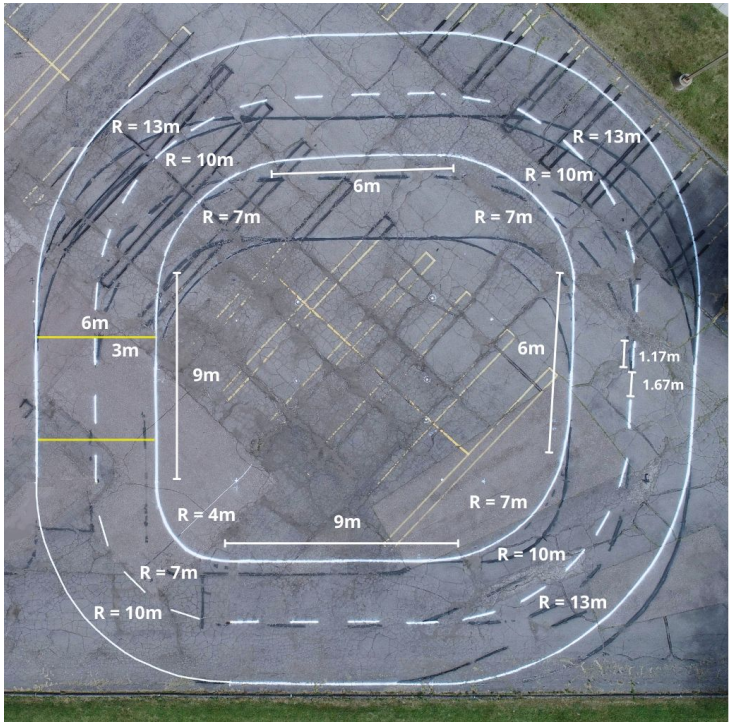
Lane following

- After detecting the lane, need to execute motion commands for our vehicle
- Lane detection algorithm gives us the location of the middle of the lane on the image
- Depending on this value we compute how much we should rotate the steering wheel



Experiment

- The performance of the five lane detection algorithms was evaluated on the Lot H course
- The objective was to complete five consecutive laps on both inner and outer lanes, driving on the right side
- In the inner lane, there is a sharp turn with a radius of just 4 meters where the lane-following algorithms fail more frequently
- In cases where an algorithm failed to complete all five laps, it was re-run at reduced speed, as necessary



Results

- All five algorithms successfully completed five consecutive laps on both inner and outer lanes
- The only algorithm to successfully complete all 10 laps on the first attempt was DBSCAN
- We also assess the metrics for reliability and comfort

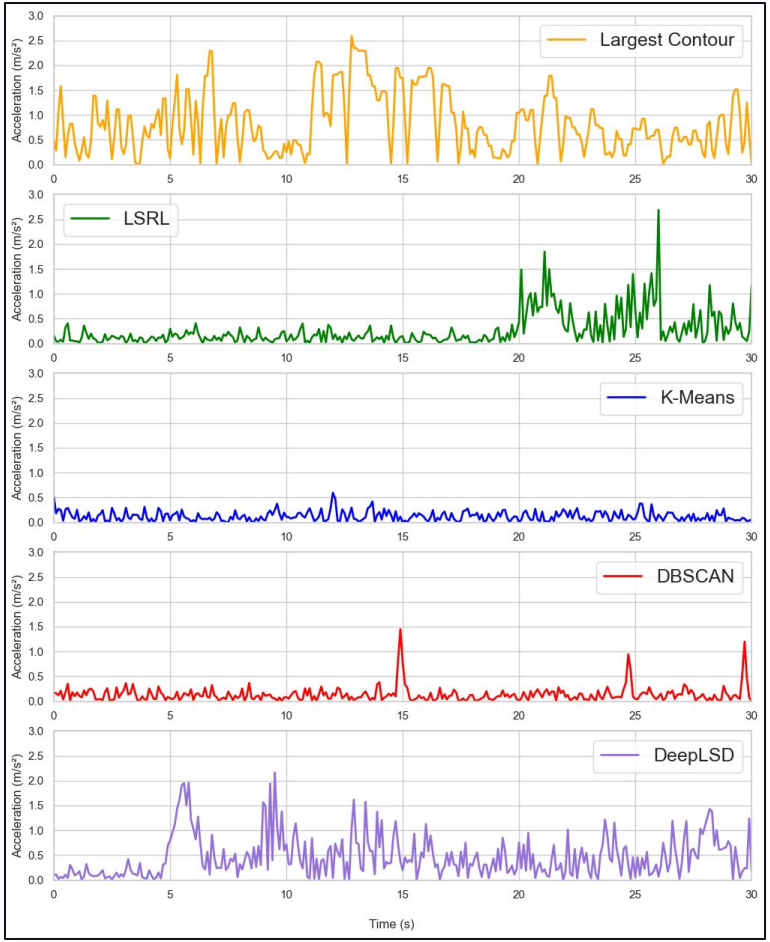
ATTEMPTS NEEDED BY EACH ALGORITHM.

Type	DeepLSD	DBSCAN	K-Means	LSRL	Largest Contour
Inner	5	1	2	2	2
Outer	1	1	1	5	3
Total	6	2	3	7	5

Results - Acceleration

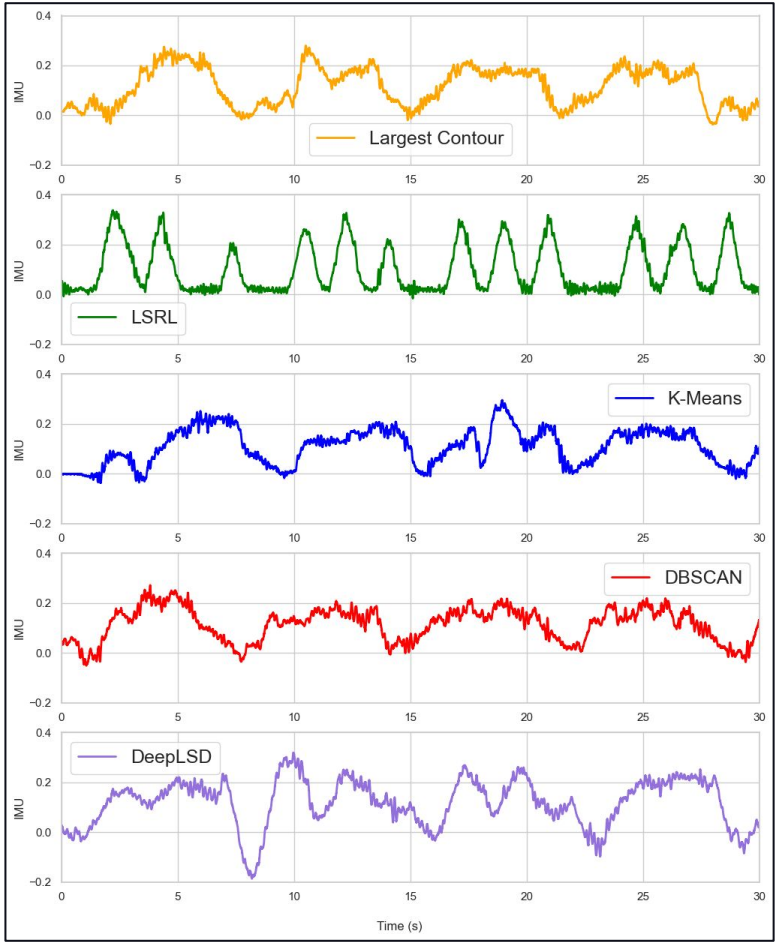
- Sharp turns, inclined slopes, potholes, and the algorithms themselves can cause variations from the target speed.
- Therefore we want smoother graph

$$v_i = \frac{d_i}{t_i - t_{i-1}}, \quad a_i = \frac{v_{i+1} - v_i}{t_{i+1} - t_i}$$



Results - Inertial Measurement Unit (IMU)

- Measures forces acting on the vehicles
- Smoother peaks mean better comfort.
- Negative peaks indicate that algorithmic overcorrection and recovery



Conclusion

- Based on our results DBSCAN and K-Means algorithm are the most robust lane following algorithms
- These algorithms achieved maximum speeds of 3.5 m/s in the outer lane and 2.5 m/s in the inner lane
- Processing times of 10 ms or less per frame, ensuring real-time performance

Questions