HIGH SCHOOL EDITION
Journal of Student Research
Volume 13 Issue 1 (2024)

Computer Vision Based AI for Control of High Velocity Autonomous Vehicle with Lane Detection

Jeffrey Aaron Jeyasingh[1] and Tony Smoragiewicz[2]

[1]Arcadia High School, USA
[2]Northeastern University, USA

ABSTRACT

Line following algorithms often fail to account for the position of the vehicle within the lane, only depending on the curvature of the lane to determine steering output. For example, with previous algorithms, the vehicle could be to the right of the lane and the steering angle from the algorithm's output would not correctly return the car to the center of the lane. Additionally, error can result from incorrect calibration: if the vehicle is truly pointing at a steering angle of 61 degrees when the correct angle should be 60 degrees, the vehicle will drift 1 degree throughout the turn. The error accumulates and ultimately will cause the vehicle to leave the lane. By fitting a polynomial curve to the input image data, the exact location of the line can be determined relative to the vehicle. Furthermore, the center of the lane can be pinpointed, offset calculated, and trajectory modified to smoothly return to the center of the lane.

Problem Formulation

The goal of this project was to develop an algorithm that could process data from a video input, detect a line, and then compute a steering angle such that the car follows the line. Line-following robots are often used in industrial applications as these can predictably follow a path from point A to point B with low computation. Because the goal was to control the car at high velocities, decisions had to be made quickly with low latency in order to ensure a successful algorithm. A chain of if-else statements would not be optimal because it would result in the robot unable to handle certain cases (Bojarski et al. 2016). Another constraint that was faced was the limited processing power on single board computers such as the Raspberry Pi. In order for the robot to consume power efficiently, a single board computer such as the Raspberry Pi had to be used.

Initially, the problem was approached by using a machine learning approach, similar to the approach used in this paper: (Bechtel et al. 2018). With this approach, the control of the car is done end-to-end by a Deep Learning Convolutional Neural Network (CNN) model. However, the car did not do a good job of staying on the track. This is because even though the CNN could accurately predict the angle, it could not determine its position relative to the track itself, causing the robot to be offset from the track.

In order to fix this issue, a computer vision approach was used. With computer vision techniques, the position of the track can be pinpointed, and the robot can modify its trajectory to return to the track.

Code from this website (Tian 2019) 1 was used to guide the creation of the deep learning approach.

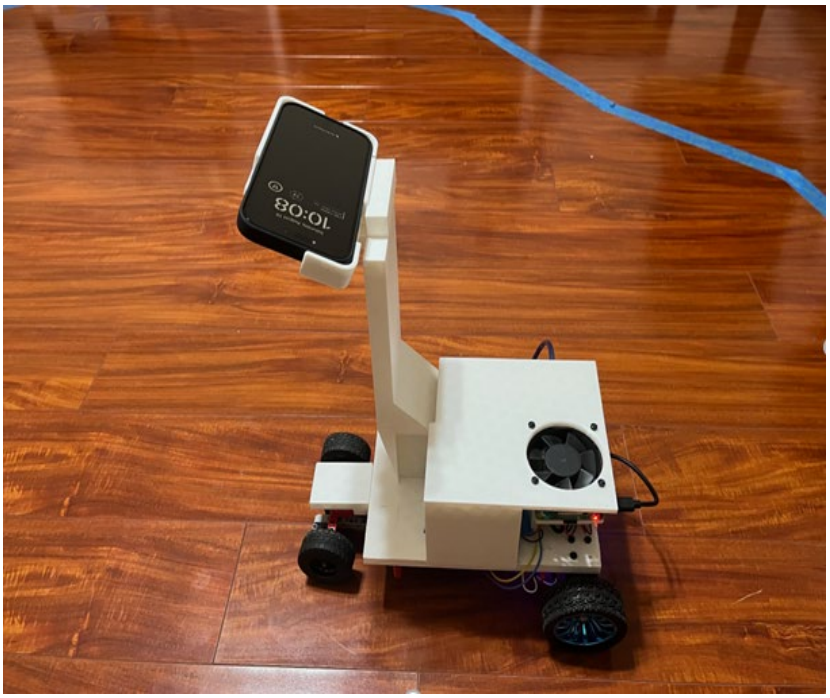


Figure 1. Image of the robot created to run the machine learning algorithm

Hardware/Software Used for Robot

In order to capture the floor, an iPhone 14 Pro is used and connected wirelessly to a MacBook M2 Air through Apple's Continuity Camera. The MacBook processes the camera data and runs the computer vision algorithm and then sends a steering and motor speed command to the Raspberry Pi 4b on the robot through ZMQ networking. Then, through a serial connection, the Raspberry Pi 4b forwards the controls it receives from the MacBook to an Arduino on the robot. Finally, the Arduino controls the motors and the steering based on the serial data it receives.

Algorithms Investigated

Image Processing

The first goal was to detect the position of the track which was a line. Because the camera input was tilted at an angle to capture more of the track than the robot, the image was slightly distorted. The raw input from the camera is shown in Figure 2.

Warp Perspective

In order to correct the error and create a 1:1 correlation with physical units and pixels on the camera data, the 'warpPerspective' command from OpenCV was used as shown in Figure 3. The distorted image was set up such that each pixel represents a millimeter.

Gaussian Blur

Before extracting the data from the image, it was blurred using a Gaussian blur to eliminate sources of noise as seen in Figure 4.

HSV Color Space Clipping

To extract the data, the im- age was first converted to the Hue, Saturation, Value (HSV) color scheme as shown in Figure 5. In this color scheme, Hue represents a certain color while Saturation and Value allow the shade and brightness of the color to be adjusted.

Threshold Masking

The HSV color scheme includes the color as only one component as opposed to the RGB color scheme which has three components containing color data. This makes it easier to segment and isolate colors by masking based on a high and low value. This result masking a range of HSV values is shown in Figure 6.

Figure 2. Raw video from the camera

Figure 3. Image Warping

Figure 4. Gaussian Blur

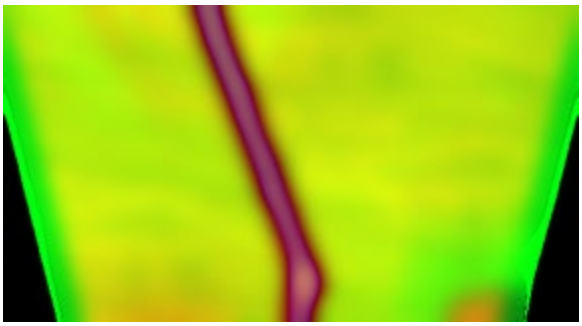


Figure 5. HSV Color Space

Figure 6. Threshold Masking

Lane Detection

Curve Fitting with Polynomial Regression

After extracting the pixels that represent the lane as shown in Figure 6, it needs to be converted to a form that is easy to manipulate and measure. A polynomial equation solves this problem effectively. By fitting the data to a polynomial equation, it is possible to utilize the power of an equation. The curvature, slope, and length of a curve can be calculated with basic calculations giving us access to more metrics to make a better angle prediction.

In order to fit the data to a curve, each pixel is treated as a data point. Then polynomial regression is run in order to determine a polynomial of a certain order that fits the data the best. Examples of different polynomial fits are shown in Figure 8.

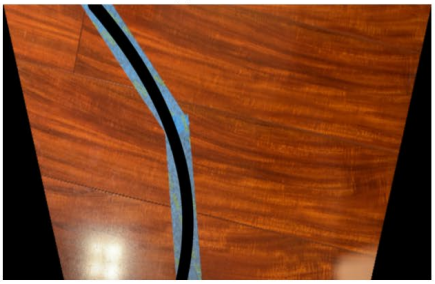
To reduce processing time, the image can be downscaled in size to retain information and reduce the number of data points. The RANSAC algorithm could also be tried in order to reduce the effect of outliers in the data. However, this method was not tested in this paper because the effect of outliers did not significantly impact performance of the algorithm.

Where Curve Fitting Doesn't Quite Fit

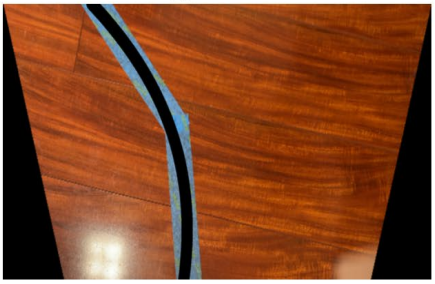
Due to the nature of a curve fitting algorithm to minimize the error or distance between the input data and the output function, it can produce results that are mathematically optimal but not visually optimal. Some examples of this happening are shown in Figure 9. These are edge cases that this algorithm can't handle, but these occur only momentarily and can be alleviated with some type of filter on the output of the algorithm.

Figure 7. Polynomial Curve Fitting Example

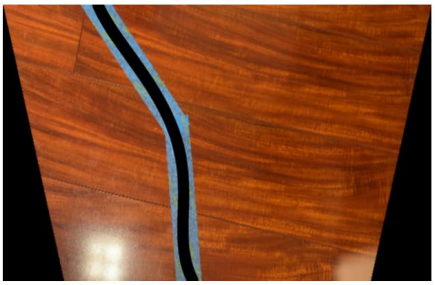
(a) 1st order



(b) 2nd order

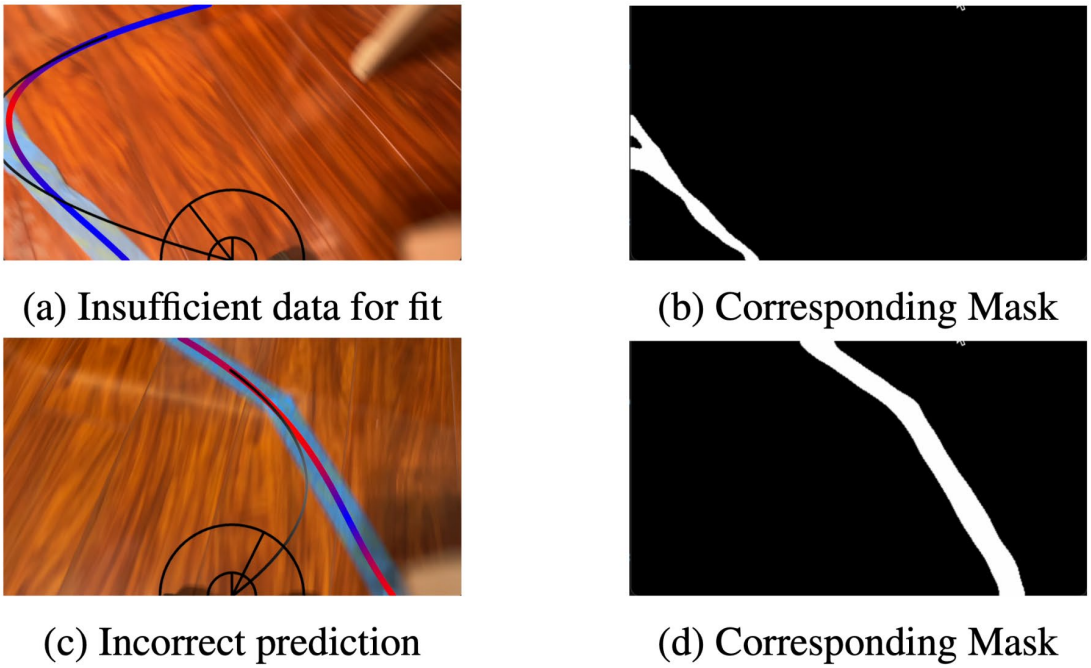


(c) 3rd order



(d) 4th order

Figure 8. Different orders of polynomial fits



(a) Insufficient data for fit (b) Corresponding Mask

(c) Incorrect prediction (d) Corresponding Mask

Figure 9. Instances where the algorithm doesn't work

Path Creation

In order to find the appropriate steering angle for the robot, a path needs to be computed that will allow the robot to return to its right location. In our example, the x axis runs from top to bottom while the y axis runs from left to right.

Find Target Point

In order to find the target point, we choose a point that is a fixed distance along the line obtained by curve fitting as shown in Figure 7. This point will be referred to as (x_f, y_f)
The starting point is the robot's current position, (x_0, y_0).

The coordinate system is set up such that left/right relative to robot is x-axis and the vertical (close/far) is the y-axis.

Calculate a Trajectory Line

A 3rd order polynomial was used for the line of best fit. Its form is: $y_{fit} = ax^3 + bx^2 + cx + d$

$$A = \begin{bmatrix} x_0^3 & x_0^2 & x_0 & 1 \\ 3x_0^2 & 2x_0 & 1 & 0 \\ x_f^3 & x_f^2 & x_f & 1 \\ 3x_f^2 & 2x_f & 1 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} y0 \\ 0 \\ yf \\ 3ax_f^2 + 2bx_f + c \end{bmatrix}$$

$$C = A^{-1}B$$

Matrix C holds all the coefficients for the final trajectory equation, which is in the form $y_{traj} = tx^2 + ux + v$

$$C = \begin{bmatrix} t \\ u \\ v \end{bmatrix}$$

To find the steering angle, find the slope of y_{traj} at an x value that is halfway between x_0 and x_f.

Experiments

Some characteristics used to judge the effectiveness of each algorithm are below:

- Max Rotational Acceleration - used to judge how smooth the car's rotation is
- Max Linear Acceleration - used to judge how smooth the movement and braking of the car is
- Lap time - how fast the car can complete a single lap

Lap time is only included in Experiment 5 because it was the only experiment that could traverse the path at high speeds. All values have been averaged over 4 trials (4 laps).

Experiment 1 to 2 - Deep Learning

A deep learning approach was used for the first two experiments. The deep learning model used was based of this paper: https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf

This model takes in the image data from the camera and outputs a steering angle for the robot to drive in.

Experiment 1

Experiment 1 used the deep learning model described above. Although the robot could follow the track, it did not perform well at sharp bends where the camera could no longer see the track.

- Portion of track completed: Full
- Max Rotational Acceleration: 1.12 $\frac{rad}{s^2}$
- Max Linear Acceleration: 3.60 $\frac{m}{s^2}$

Experiment 2

Experiment 2 builds upon Experiment 1 by adding a median filter to the output steering angle. This helps

the model be less affected by noise or random erroneous outputs. From the data

- Portion of track completed: Full
- Max Rotational Acceleration: $0.78\ \frac{rad}{s^2}$
- Max Linear Acceleration:$3.93\ \frac{m}{s^2}$

Experiment 3 to 4 - Computer Vision (Hough Line Transform)

These experiments tried a new approach by using computer vision to control the process. By masking the blue color and extracting the pixels of the lane as shown in Figure 6, a Hough Line Transform was applied to extract all the straight-line segments. Then the slopes of the lines were averaged to find the angle of the lane, which corresponds to the angle that the car should steer in.

Experiment 3

Experiment 3 used the computer vision algorithm described above.

The results from this experiment were similar to those of Experiment 1-2: Although the robot could follow the track, it did not perform well at sharp bends.

- Portion of track completed: Full
- Max Rotational Acceleration: $1.35\ \frac{rad}{s^2}$
- Max Linear Acceleration:$2.50\ \frac{m}{s^2}$

Experiment 4

Experiment 4 builds upon Experiment 3 by adding a median filter to the output steering angle.

- Portion of track completed: Full
- Max Rotational Acceleration: $0.67\ \frac{rad}{s^2}$
- Max Linear Acceleration:$3.26\ \frac{m}{s^2}$

Findings of Experiment 1 - 4

After performing experiments 1-4, it was noticed that the car was incapable to returning to the center of the lane when it was displaced. This is because the car can only determine the angle of the lane relative to its orientation. It has no knowledge of whether it is to the left, right, or centered on the lane.

Experiment 5 to 6 - Linear Regression with Computer Vision

Experiment 5 includes the new algorithm described in the Algorithms Investigated section.

Experiment 5

- Portion of track completed: Full
- Max Rotational Acceleration: $0.64\ \frac{rad}{s^2}$
- Max Linear Acceleration: $2.87\ \frac{m}{s^2}$
- Lap time: $7.21s$

Experiment 6

The goal of experiment 6 was to increase the speed of the lap time. This was done by increasing the robot's velocity on straight steering output, while decreasing speed on sharp turns to prevent the car from exiting the designated path. Algorithm 1 was utilized.

- Lap time: $5.88s$

Algorithm 1: Speed control algorithm

Input: Steering Angle (-1 to 1 range) **Output**: Velocity of car

Let $maxSpeed = 200$.
Let $minSpeed = 200$.
Let $diff = maxSpeed - minSpeed$
return $maxSpeed - diff * abs(steeringAngle)$ =0

Other Experiments

After performing the previous experiments, an efficient algorithm was found to control the car reliably. Now, the algorithms are tested for speed. By measuring the lap time of the car under the control of different drivers, it is possible to analyze how the algorithm performs against other human drivers. Figure 10 shows the results of the data collection mentioned.

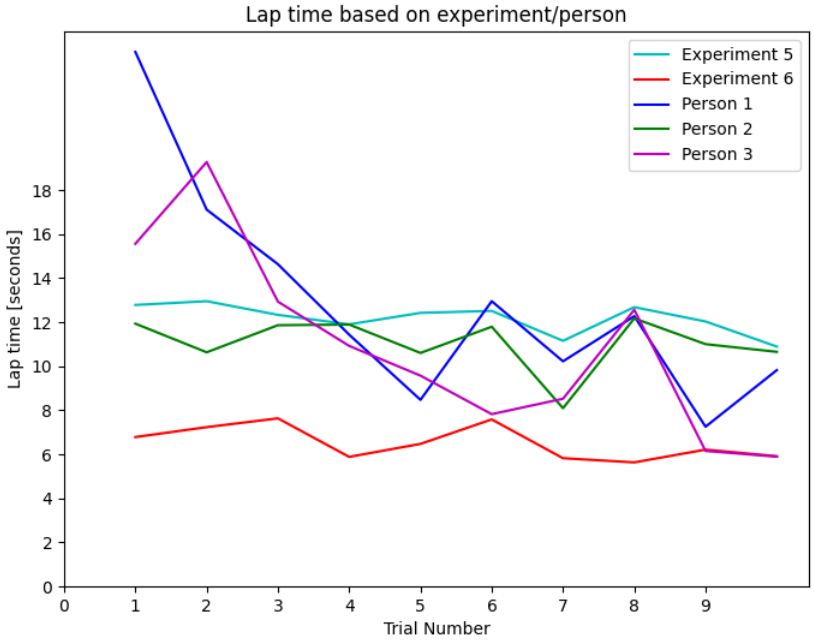


Figure 10. Different orders of polynomial fits

Results

From the data, it is clear that the algorithm used in Experiment 5 and 6, described in Algorithms Investigated section, are the most effective at turning smoothly. The max rotational acceleration was, on average, lower than in the other experiments.

Because we can mathematically represent the line as a polynomial equation, we are able to compute the appropriate line that the robot should take in order to return to the track. The thought process is inherently different: instead of trying to move in the direction of the track, we check how offset we are from the track and then produce a steering angle that fixes it. In this manner, the car will always return to the track, no matter its orientation.

From Figure 10 it is seen that humans are often slower than robots at first but improve over time and occasionally beat the car's time. However, the algorithm performs better because it has a faster reaction time and can instantly react when the car veers off the track.

Conclusion

The lane following algorithm developed in this paper effectively locates the lane and positions the robot to be within the lane. By continually centering the robot, the robot will always remain within its lane. In order to increase the speed of the robot, an algorithm was developed to adjust the speed of the robot based on the degree of turning being executed by the robot.

To add to this experiment, in the future, multiple lanes could be added, and the algorithm could be extended to allow the robot to identify lanes. Additionally, the algorithm could use better outlier detection in order to avoid misidentifying other blue objects on the track as lane lines.

Some modifications that could be made to the experiment are changing the color of the lane and changing its shape (ex. dashed, dotted, double line).

Potential Improvements

In order to improve the performance of the robot, the camera needs to be better at capturing fast moving objects. Although the iPhone 14 Pro camera outperforms other webcams that were tested, its images still contain large amount of motion blur when the robot is moving at its maximum velocity (around 1m/s). On another note, instead of replacing the camera, better image processing techniques could be used to account for the blur and noise from high-speed motion.

Acknowledgments

I would like to thank my advisor for the valuable insight provided to me on this topic.

References

Bechtel, M. G.; McEllhiney, E.; Kim, M.; and Yun, H. 2018. DeepPicar: A Low-cost Deep Neural Network-based Autonomous Car. arXiv:1712.08644. https://doi.org/10.48550/arXiv.1712.08644

Bojarski, M.; Testa, D. D.; Dworakowski, D.; Firner, B.; Flepp, B.; Goyal, P.; Jackel, L. D.; Monfort, M.; Muller, U.; Zhang, J.; Zhang, X.; Zhao, J.; and Zieba, K. 2016. End to End Learning for Self-Driving Cars. arXiv:1604.07316. https://doi.org/10.48550/arXiv.1604.07316

Tian, D. 2019. DeepPiCar-part 5: Autonomous Lane Navigation via Deep Learning.