Real-Time Lane Tracking for Mobile Devices

Jaineel Dalal (jdalal@andrew.cmu.edu) William Ku (wku@andrew.cmu.edu)

December 13, 2015

Abstract

Driving within the traffic lane is a difficult task - even for seasoned drivers. Drivers often have different clues for driving in the same lane (e.g. aligning certain parts of the vehicle to the road) developed through years of driving experience. Furthermore, multitasking between following the traffic rules, paying attention to other cars and pedestrians, and driving the car can become a daunting and an overwhelming experience for a novice. In this project, we provide a visual guidance solution on an iPad Air 2 for drivers to keep within their driving lanes. The goal is to simulate an easy-to-follow standard for drivers to maintain their driving lanes. The end result is a mobile application that detects and renders lane lines, and projects the center of the lane onto the screen in real-time (40 frames-per-second).

1 Introduction

While most of the high-end mobile devices today are equipped with powerful hardware, they are not capable of running computer vision applications such as lane tracking in real-time. The reason is twofold - the software is not optimized to leverage the onboard architecture (e.g. GPU), or the hardware is unable to deliver the required performance due to power budget constraints. Our project aims to solve this problem by taking advantage of the architecture in the iPad Air 2 by leveraging the onboard GPU through the GPUImage library and exploiting efficient Hough transform implementations.

The flow chart for a typical lane detection and tracking algorithm is shown in Figure 1. The feasibility of running lane detection in real-time can depend on the hardware architecture [1]. While using OpenCV for image smoothing, edge detection and line detection can be trivial on a desktop with enough computational power and memory. Doing the same on a mobile device like an iPad with limited hardware capabilities can be difficult. In such scenarios, there needs to be an efficient implementation of the existing algorithms balance the computational load.

In this paper, we employ the GPUImage library - an efficient GPU-based open-source iOS framework - for image and video processing [4] that takes advantage of the multi-core and GPU architecture of an iPad. This allows us to do all the operations required for lane detection in real-time.

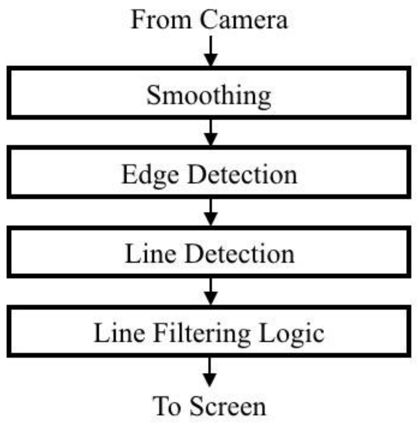
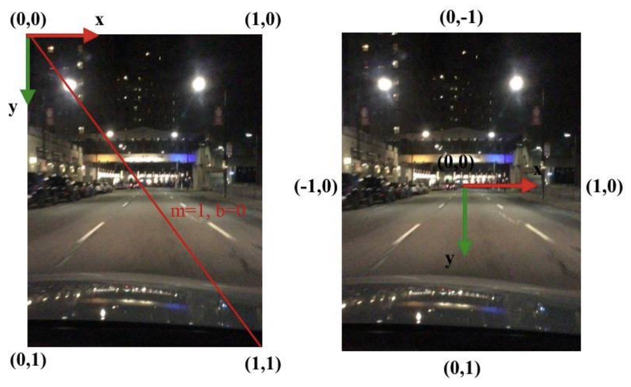


Figure 1: Flow chart for Lane Tracking



(a) Conventional (b) Hough Lines.

Figure 2: Image Coordinates within GPUImage.

2 Background

The GPU-based implementation for this project was derived from Brad Larson's open source GPUImage library [4]. Apart from that, a large part of the inspiration for this project came from [1], [5] and [6]. The iPhone-based real-time lane tracking system described in [1] shows the feasibility as well the limitations of real-time lane detection on mobile devices. Besides the traditional Canny edge detection and Hough transform combination, the spline-fitting technique described in [2] highlights an alternative approach for line detection. This class of techniques can further extend to deal with curved roads. Applying the latter technique remains an area of interest and future work as described in the Conclusion section.

3 Approach

Our approach is based on the open-source GPUImage framework provided by Brad Larson [4]. The GPUImage library builds upon OpenGL-ES, which accesses the onboard GPU of the iPad for parallelization speedup.

3.1 Coordinate Transform

The GPUImage library adapts the conventional image coordinate frame C_i (Figure 2a), where the origin O_i is located at the upper-left corner of the image. Input video width and height, $[w_i, h_i]$, are normalized such that $w_i \in [0,1]$, $h_i \in [0,1]$. The red diagonal line in Figure 2a will have a slope $m = 1$ and intercept $b = 0$.

However, the lines returned by the Hough transform operation reside in the Hough coordinate space C_H (Figure 2b). The origin O_H is located at the center of the image, where $w_H \in [-1,1]$, $h_H \in [-1,1]$.

The coordinate transform from the Hough line coordinates to the image coordinates is therefore:

$$x_i = x_H/2 + 0.5$$
$$y_i = y_H/2 + 0.5$$

where (x_i, y_i) is a pixel location in the image coordinates and (x_H, y_H) is the pixel location expressed in the Hough coordinates.

3.2 Acceptance Bound

Given an array of detected lines - as the slopes m and intercepts b for the line equation $y = mx + b$ - we are interested in selecting only the ones corresponding to the current driving lane. The slopes of such lines are a function of the iPad view angle on the vehicle, as shown by Aly et al. [2]. In our experiments, we fixed the slope of the lane lines by locking the iPad position in our test vehicle - a mid-sized compact car. The slope value was determined empirically to be approximately ± 0.5. However, there can be several lines detected that bound our driving lane. The vibration of the iPad may also affect the detection output. As a result, the slope values may vary. Therefore, we relax the slope requirements - which we call the Acceptance Bound - such that we keep the lines whose absolute slope values fall within a given bound. Then we average the slopes and the intercepts of all left and right lines, respectively, in the Acceptance Bound to obtain the left and right line which bound our driving lane.

3.3 Road Center Calculation

To successfully aid the driver, our screen needs a visual guidance that marks the center of the lane. We decide to use the center point of the x-intercepts at $y_i = 1$ (i.e. the bottom of the screen). We extend the two lines determined from our line filtering logic to find their x-intercepts. The equation can be derived from the line equation:

$$y = mx + b$$
$$x = \frac{y - b}{m}.$$

After finding the two x-intercepts of the left and right lane, we simply average the two to obtain the center of the lane in the x coordinate at $y_i = 1$.

3.4 Architecture

The architecture for our solution is shown in Figure 3. The onboard camera is accessed via the `GPUImageVideoCamera` class. The live video stream is passed to the `GPUImageHoughTransformLineDetector` class for processing. Internally, the Hough transform class calls the `GPUImageCannyEdgeDetectionFilter` class to detect edges in the video image. The Canny edge class in turn converts the video image into gray scale, passes through a blur filter (through convoluting with a Gaussian filter), and applies Sobel gradient operators to find the edge response. Then the Hough transform class applies a non-maximum suppression filter to find the maximum line responses.

The GPUImage library implements a version of the Hough transform that uses the concept of the *parallel coordinate space* [5, 6]. Similar to the traditional Hough transform implementation, this new version accepts a hyperparameter for thresholding the line response. In another words, the hyperparameter controls the threshold for the density of line crossings in order to declare detection of a line. This enables Hough transform to be performed efficiently in a GPU parallelized environment.

The output of the Hough transform is relayed through a callback function. The function is called with a C-array of slope-intercept pairs for each line detected, the number of lines detected, and a timestamp. We perform our line filtering logic - acceptance bounding, road center calculation - within the callback function. Lastly, the results are blended with the original video stream for display on the iPad's screen.

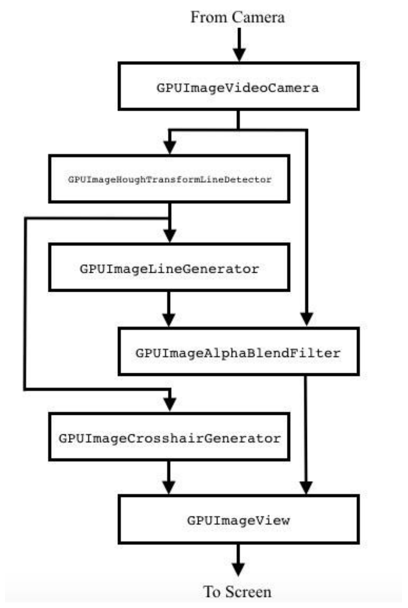


Figure 3: Project Architecture based on GPUImage.

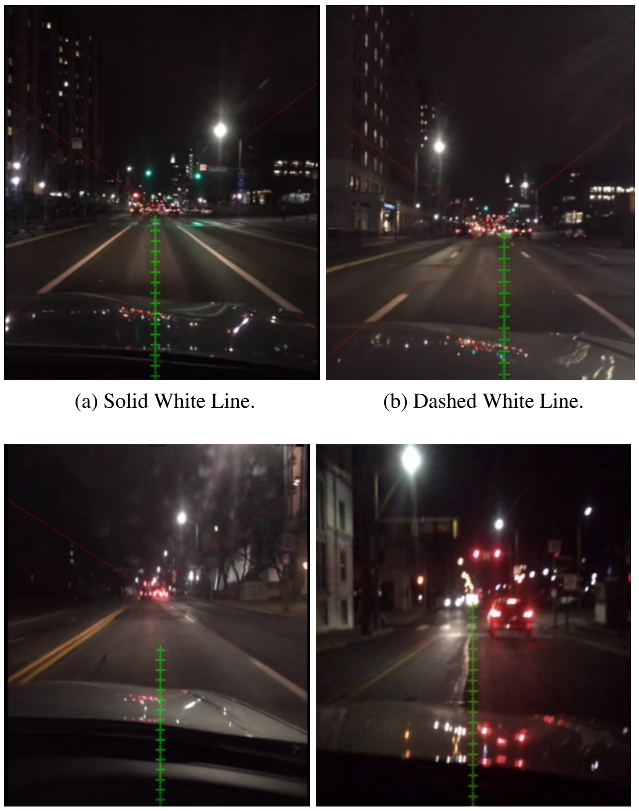
4 Results

4.1 Different Lane Scenarios

We tested our application for real-time lane detection with different types of lanes: solid white lines (Figure 4a), dashed white lines (Figure 4b), and double yellow lines (Figure 4c). The hyperparameter for the Hough transform was tuned to be 0.20. The Acceptance Bound used was $[0.4, 0.8]$. As seen from these figures, the lanes can be detected successfully in different scenarios.

4.2 Lighting conditions

In order to measure the sensitivity under varying lighting conditions, we tested our application from optimal to poor lighting. While the application performs well under sufficient lighting at night, it fails to detect any lanes given little light (Figure 4d).


(a) Solid White Line. (b) Dashed White Line.

(c) Double Yellow Line. (d) Fails in Poor Lighting.

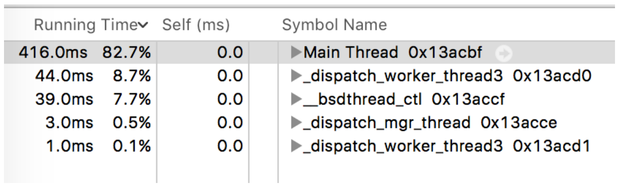
Figure 4: Lane Detection Testing

4.3 Road center estimation

The current lane tracking system estimates the center of the lane and projects it onto the screen as green cross-hairs (Figures 4a-4d). In order to do so, we follow the road center calculation approach described in Section 3.3 whenever both the left and right lanes are detected (as in the Figures 4a-4c). In case fewer than two lanes are detected as in Figure 4d, the center of the lane defaults to the center of the iPad (i.e. $(x_i, y_i) = (0.5, 1.0)$).

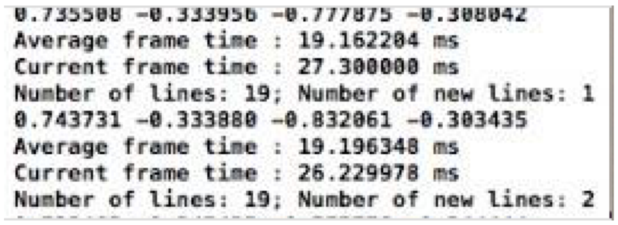
4.4 OpenCV vs GPUImage Benchmarking

To evaluate our application against the typical OpenCV implementation, we ran a test using Gaussian smoothing, Canny edge detection, and Hough line detection algorithms in OpenCV with the Instruments-Time Profiler tool in Xcode and compared it against the benchmark output from the GPUImage library. The results from this test are shown in Figure 5 and Figure 6. The experiment yields at least a 20 times speedup for the GPUImage implementation against the OpenCV version.



Running Time		Self (ms)	Symbol Name
416.0ms	82.7%	0.0	▶Main Thread 0x13acbf
44.0ms	8.7%	0.0	▶_dispatch_worker_thread3 0x13acd0
39.0ms	7.7%	0.0	▶__bsdthread_ctl 0x13accf
3.0ms	0.5%	0.0	▶_dispatch_mgr_thread 0x13acce
1.0ms	0.1%	0.0	▶_dispatch_worker_thread3 0x13acd1

Figure 5: Time taken by OpenCV.



```
0.735508 -0.333956 -0.777875 -0.308042
Average frame time : 19.162204 ms
Current frame time : 27.300000 ms
Number of lines: 19; Number of new lines: 1
0.743731 -0.333880 -0.832061 -0.303435
Average frame time : 19.196348 ms
Current frame time : 26.229978 ms
Number of lines: 19; Number of new lines: 2
```

Figure 6: Time taken by GPUImage.

5 Conclusion

We delivered a GPU-based implementation of the road lane detection algorithm using the GPUImage library. The application is able to identify the bounding lines of the current driving lane given their existence in the image and provide a rough estimation of the center of the current lane. All the required goals for this project were met - implementing lane detection and center of road estimation for different types of lane markers under varying light conditions, guaranteeing real-time performance on a mobile device (i.e. iPad Air 2 at approximately 40 frames per second). In future versions of our work, we look to incorporate the device's location in the vehicle during calibration to estimate the center of the vehicle. The driver will then be able to match the vehicle and road centers to stay in the center of the lane. We further pursue a spline-fitting approach as suggested in [4] to perform generalized lane detection (e.g. curves). We are also considering integrating the ability to ”cache” previous detection results for smoothing purposes as well as guaranteeing continuous detection of lines in the absence of road lanes. Apart from that, data from the global positioning system (GPS) can also be fused with the existing lane tracking framework for real time location-based lane detection.

6 List of Work

Equal work was performed by both project members.

Jaineel: Video camera interface, OpenCV benchmarking, testing, reporting.

William: Hough transform prototyping, lane filter logic, testing, reporting, and driving.

7 GitHub Page

All the code for this project is listed under `https://github.com/wiwiku/CVApps`.

8 References

[1] Feixiang Ren et al. *Lane Detection on the iPhone*

[2] Mohamed Aly et al. *Real-Time Detection of Lane Markers in Urban Streets*

[3] Albert S. Huang. *Lane Estimation for Autonomous Vehicles using Vision and LIDAR* 2010: Thesis, Massachusetts Institute of Technology

[4] Brad Larson. *An open source iOS framework for GPU-based image and video processing https://github.com/BradLarson/GPUImage*

[5] M. Dubsk, J. Havel, and A. Herout. *Real-Time Detection of Lines using Parallel Coordinates and OpenGL.* Proceedings of SCCG 2011, Bratislava, SK, p. 7.

[6] M. Dubsk, J. Havel, and A. Herout. *PClines Line detection using parallel coordinates*. 2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), p. 1489- 1494.