FPGA Realization of Lane Detection Unit using Sliding-based Parallel-Segment Detection for Buffer Memory Reduction

Heuijee Yun[1] and Daejin Park[1*]
[1]School of Electronic and Electrical Engineering, Kyungpook National University, Daegu, Republic of Korea
*Correspondence to: Daejin Park (boltanut@knu.ac.kr)

***Abstract*—With the development of various chips, such as VLSI chips, and the development of semiconductor technology and artificial intelligence, autonomous driving technology is advancing daily. Lane recognition technology, which can be considered the most important in the implementation of autonomous vehicles, requires a large amount of computation and processing time because data must be received from a camera attached to the vehicle and processed in real time. To design a more efficient and implementable lane recognition algorithm, we proposed a method to reduce the usage of buffer memory by using parallel operation. Most of the boards used in autonomous vehicles are lightweight, so the lane recognition algorithm is also lightweight to use a minimum library. First, after reading the image, canny edge-detection is executed through grayscale conversion, Gaussian smoothing, a Sobel operator, non-maximum suppression, and hysteresis in parallel. The lane is inspected using the Hough transform as an input to the image with the edge detected by canny edge-detection. Due to parallel operation's nature, the effect is insignificant when a single image input is received, but the operation is more efficient when multiple images are input in real time. We used a relatively low-level C language for efficiency and processed images with loops and operations.**

***Index Terms*—Autonomous driving, lane detection, parallel processing, canny edge detection, Hough transform**

I. Introduction

Currently, with the development of semiconductor design technology, small, inexpensive, and high-speed processors are being developed. Therefore, it is becoming easier to divide a program into several small parts and process them in parallel. The lane recognition function, which is the most important function of autonomous driving, receives a camera image in real time and makes calculations using extensive data, which take a long time to process. In addition, because the board used for autonomous vehicles is a lightweight embedded board, the amount of computation and energy consumed is high, resulting in low efficiency. Therefore, in this study, we used a method that implements a lane recognition algorithm using parallel operation to reduce processing time. We deemed the operation suitable for parallel processing because it used multiple loops in the operation process when processing the image. We used and implemented C language, which is a relatively low level language and uses a minimal library.

This study was supported by the BK21 FOUR project funded by the Ministry of Education, Korea (4199990113966, 10%), and the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2018R1A6A1A03025109, 10%, NRF-2022R1I1A3069260, 20%). This work was partly supported by an Institute of Information and communications Technology Planning and Evaluation (IITP) grant funded by the Korean government (MSIT) (No. 2021-0-00944, Metamorphic approach of unstructured validation/verification for analyzing binary code, 20%) and (No. 2022-0-00816, OpenAPI-based hw/sw platform for edge devices and cloud server, integrated with the on-demand code streaming engine powered by AI, 20%) and (No. 2022-0-01170, PIM Semiconductor Design Research Center, 20%), and the EDA tool was supported by the IC Design Education Center (IDEC), Korea.

II. Proposed Method

A. Overall Structure

Fig 1 (a) illustrates the system's structure. After receiving the image input, the image data is copied to memory. After the image data is copied, it can be processed in a parallel fashion on a row by row basis. We used a De1-SoC board [1] for a lightweight processor. Because the DE1-SoC board has dual-core Cortex-A9 embedded cores, we thought it would be easy to perform parallel image processing. Since libraries such as Opencv are large and take a long time to execute, we decided to implement each image-processing process ourselves.

B. Image Processing Structure

We used a canny edge detection algorithm [2]for edge detection. First, the image should be converted to grayscale. Then, to remove noise from the image, we applied a Gaussian filter. To identify edges in the image, we used a Sobel operator, which operates by finding changes in horizontal and vertical image intensity. Non-maximum suppression can enhance and sharpen the edges by making them thinner. In the hysteresis stage, we can remove pixels that do not belong to the edge by determining whether each pixel exceeds the user-defined threshold. To extract lanes, we used the Hough-transform algorithm [3]. We described lines using Hesse normal representation, which uses two parameters (ρ, θ) to describe a line. The Hough transform uses an array called accumulator. Each edge pixel the Hough algorithm swipes contributes +1 to the accumulator. We used local maxima from the accumulator to eliminate false duplicates. Then, we overlaid the line on the input image. Fig 1 (b) shows the result of the lane detection.

As explained earlier, this image-processing algorithm requires many program loops. Because we needed to execute the Hough transform algorithm in the whole-image data, we used parallel processing in canny edge detection. Also because we needed to determine the image's height and width, we first copied it to memory. Since global memory access is slow, it should cache the pixels into local memory. We used two types of parallel programming, posix thread and OpenMP (Open Multi-Processing). The Posix library, known as pthread, is low-level implementation and OpenMP is higher level implementation. We parallelized by data because the Gaussian

2023 IEEE International Conference on Consumer Electronics (ICCE) | 978-1-6654-9130-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICCE56470.2023.10043180

978-1-6654-9130-3/23/$31.00 ©2023 IEEE

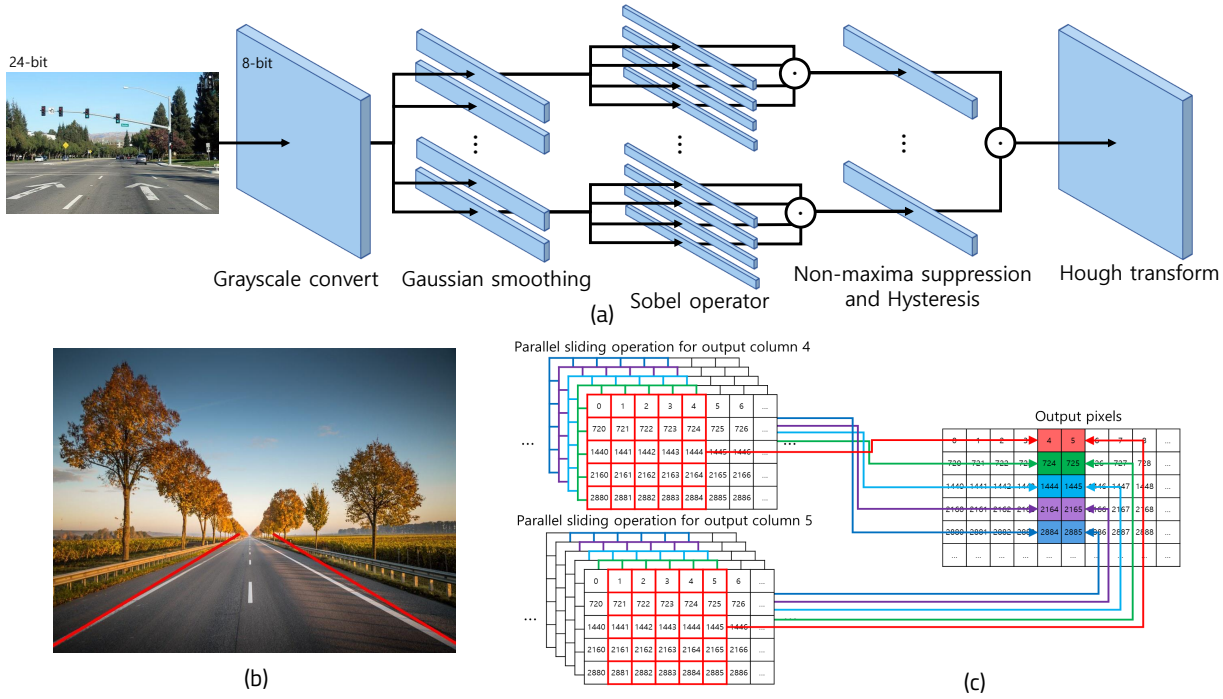

Fig. 1. Structure of program and result (a) Overall structure (b) Canny Edge detection structure (c) Lane detection structure (d) Lane detection result

smoothing and Sobel operator stage uses three nested loops to process the image by row. A thread processes edge detection by row, processes rows by pixel, and processes pixels by filter. We divided the image into 5 pixel rows for a thread. Fig 1 (c) illustrates the process of parallel operation in pixels in the Gaussian-smoothing stage. Gaussian smoothing is a box operation that works on a box of pixels to determine each pixel's output. When processed sequentially, all of the image's pixels are scanned from start to finish and then calculated with a Gaussian filter, which is a 5 x 5 matrix. However, in parallel programming, we can process the Gaussian operation by row at the same time.

C. Measurement

Fig 2 illustrates time and memory measurement of serial and parallel processing. Processing the algorithm serially took 255.83ms, and it took 187.53ms in edge detection . However, the program used posix thread processing, took 131.52ms and 62.95ms in edge detection. Also, edge detection took 60.41ms with OpenMP. It can be seen that using parallel processing can reduce processing time by two thirds. We measured the size of the board's total allocated virtual memory for the program. The sequential program used 3712KB of memory and each parallelized program used 12276KB, 10835KB memory. The parallelized programs used almost twice as much memory as the serial program. We concluded that oOpenMP took less time and memory despite the posix thread because in the light-weight process model, the repeated task is not complicated and does not require a large amount of memory.

Program	Serial processing		Parallel processing			
			Thread		OpenMP	
Edge detection	187.53 ms	3216 KB	62.95 ms	11784 KB	60.41 ms	10336 KB
Hough transform	68.30 ms	496 KB	68.57 ms	492 KB	69.26 ms	499 KB
Total time / memory	255.83 ms	3712 KB	131.52 ms	12276 KB	129.67 ms	10835 KB

Fig. 2. Structure of program and result

III. Conclusion

In this paper, we proposed a lane detection method using canny edge detection and Hough transformation processed in parallel for advanced accuracy and a small amount of computation. We proposed this structure because it is difficult to achieve high efficiency in real time due to computational volume and memory limitations. When an image is input, its edge is detected using the canny edge detection algorithm. Then we can process a Hough transformation to extract lanes. Based on processing time, our program clearly recognizes lanes with greater efficiency. With this structure, we can optimize real-time lane detection in less time.

References

[1] T. Technologies, "http://de1-soc.terasic.com," in *Terasic-DE Main Boards-Cyclone-DE1-SoC Board*.

[2] J. Canny, "A computational approach to edge detection," *IEEE Transactions on pattern analysis and machine intelligence*, no. 6, pp. 679–698, 1986.

[3] J. Illingworth and J. Kittler, "A survey of the hough transform," *Computer vision, graphics, and image processing*, vol. 44, no. 1, pp. 87–116, 1988.