ISSN:1226-7244 (Print)
ISSN:2288-243X (Online)
j.inst.Korean.electr.electron.eng.Vol.21,No.4,358~367,December 2017
논문번호 17-04-05
http://dx.doi.org/10.7471/ikeee.2017.21.4.358
28

CPU-FPGA 구조를 이용한 실시간 FCWS 구현

Real-time FCWS implementation using CPU-FPGA architecture

한 성 우 *, 정 용 진*★

Sungwoo Han*, Yongjin Jeong*★

Abstract

Advanced Driver Assistance Systems(ADAS), such as Front Collision Warning System (FCWS) are currently being developed. FCWS require high processing speed because it must operate in real time while driving. In addition, a low-power system is required to operate in an automobile embedded system. In this paper, FCWS is implemented in CPU-FPGA architecture in embedded system to enable real-time processing. The lane detection enabled the use of the Inverse Transform Perspective (IPM) and sliding window methods to operate at fast speed. To detect the vehicle, a Convolutional Neural Network (CNN) with high recognition rate and accelerated by parallel processing in FPGA is used. The proposed architecture was verified using Intel FPGA Cyclone V SoC(System on Chip) with ARM-Core A9 which operates in low power and on-board FPGA. The performance of FCWS in HD resolution is 44FPS, which is real time, and energy efficiency is about 3.33 times higher than that of high performance PC enviroment.

요 약

최근 운전자의 편의와 안전을 위해 전방 차량 추돌 감지 시스템(Front Collision Warning System : FCWS)과 같은 다양한 운전자 보조 시스템(Advanced Driver Assistance System : ADAS)이 개발되고 있다. FCWS는 주행 중 실시간으로 동작해야 하기 때문에 높은 처리속도를 필요로 한다. 또한 자동차의 전장화에 따라 FCWS를 차량용 임베디드 시스템에서 동작시키기 위해 저전력 시스템이 필요하다. 본 논문에서는 FCWS를 CPU-FPGA 구조에서 실시간 처리가 가능하도록 구현하였다. 차선 검출은 Inverse Transform Perspective(IPM)와 슬라이딩 윈도우 방식을 이용하여 CPU에서도 빠른 속도로 동작할 수 있도록 하였다. 차량검출은 높은 인식률을 가지는 Convolutional Neural Network(CNN)을 이용하였고, FPGA에서 병렬처리로 가속하였다. 제안하는 구조는 저전력으로 동작하는 ARM-Core A9과 FPGA를 내장한 Intel FPGA Cyclone V SoC(System on Chip)에서 검증하였다. HD해상도에서 FCWS는 44FPS로 실시간으로 동작하며, 고성능 PC 환경보다 처리속도 대비 에너지 효율이 약 3.33배 높은 것을 확인했다.

Key words : Embedded system, ADAS, FPGA, Convolution Neural Network, Vehicle Detection

---

* Dept. of Electronics and Communications Engineering, Kwangwoon University

★ Corresponding author

E-mail : yjjeong@kw.ac.kr , Tel :+82-2-940-5551

Manuscript received Dec. 15, 2017; revised Dec. 28, 2017 ; accepted Dec. 29, 2017

Ⅰ. 서론

최근 교통사고로 인한 사망사고를 줄이기 위해 Lane Daparture Warning system(LDWS) 장착을 대형 버스 및 트럭에 의무화하였고, 적용 범위를 점차 늘려 나가고 있다. 하지만 미국 도로교통안전국(NHTSA)의 2012년 교통사고 원인 분석 결과에 따르면, 전체 교통사고의 74% 가량이 추돌 발생 3초 전 운전자의 부주의로 인한 상황에서 발생하였다. 그 중 차선 이탈 사고는 12%에 불과하지만, 전방 부주의로 인한 사고는 29%라는 결과가 나왔다. FCWS 장착시범사업 결과, 사망사고가 82%가 줄어 교통 안전을 위한 FCWS의 필요성이 증가하고 있다[1].

차량 전장화로 인해 차량제어부터 사용자 편의를 위한 다양한 시스템이 차량 내 임베디드 시스템에서 동작하고 있다. 또한 다양한 Micro Controller Unit(MCU)와 Sensor가 차량에 내장되어 있고, 전기를 이용하여 차량을 주행하기 위해 장치들이 저전력으로 동작해야 한다. 따라서 높은 연산량을 위해 Graphic Processing Unit(GPU) 같이 비싸고 높은 전력을 필요로 하는 고성능 장치를 사용하기에는 무리가 있다. 최근 일부 블랙박스에서는 주행 영상 녹화, FCWS, LDWS를 처리할 수 있는 Ambarella社의 A12W Wearable Camera SoC와 같은 제품을 사용하고 있다. A12W는 ARM-Core A9과 영상처리 전용 Digital Signal Processing(DSP)로 동작하며, 12W의 적은 전력을 소비하여 차량용 임베디드 시스템에 적합하다. 본 논문은 FCWS에서의 차선 및 차량검출의 실시간 처리를 위한 CPU-FPGA 구조를 제안한다. 차선은 직선과 곡선 둘 다 검출하기 위해 IPM과 슬라이딩 기법을 이용하였다. 그 결과, ARM-Core에서 HD급의 영상에서 약8.5ms로 동작하였다. 차량검출은 빠른 속도와 높은 인식률을 위해, 도로 위 관심 영역에서 그림자 정보를 이용하여 차량의 후보군을 검출하였다. 검출된 후보는 차량으로 확실히 인식하기 위해 CNN[2][3]을 이용하였다. CNN의 높은 연산량은 FPGA에 다수의 Multiply-Accumulate(MAC) 유닛을 이용하여 병렬처리 하여 가속하였다. 데이터 전송은 DDR 접근 횟수를 줄이고 데이터를 재사용하는 블록 단위의 연산을 이용해 가속하였다. CNN 모듈은 소형 FPGA에서도 동작시킬 수 있도록, 연산 단위를 Fixed(16bit) 연산 방식으로 구현하였다. Fixed 방식에 의해 Float 대비 FPGA 사용량을 대폭 감소시켰다. FCWS에 대한 검증은 HD 입력영상에 대해 약 44FPS로 동작하여 실시간 처리가 가능한 것을 확인하였고, 고성능의 PC보다 처리속도 대비 에너지 효율이 약 3.33배 좋은 것을 확인하였다.

본 논문은 2장에서 제안하는 FCWS와 관련된 연구들을 소개하고, 3장에서는 FCWS의 알고리즘을 설명하며, 4장은 CPU-FPGA 구조 내부의 CNN 하드웨어를 설명한다. 5장에서는 CPU-FPGA 구조를 이용한 FCWS의 성능을 분석한다.

Ⅱ. 관련연구

1. 차선 검출 알고리즘

차선 검출은 LDWS, FCWS 등 다양한 알고리즘에서 사용된다. 차선을 검출하기 위해 대표적으로 Hough Transform, EdLines[4], CannyPF[5] 등 다양한 방법을 이용해 차선을 검출한다. 세 가지 알고리즘 중 CannyPF가 선분을 검출하는데 가장 뛰어난 성능을 가지고 있으나, 수행시간이 너무 오래 걸리고 직선 선분만 검출할 수 있는 단점이 존재한다. Alberto Broggi는 카메라 파라미터를 이용하여 영상의 원근 효과를 제거하고 하늘에서 보는 것과 같은 Bird's-eye View를 구현하였다. Alberto Broggi는 곡선을 포함한 차선을 검출하는 방법[6]을 제안하였고, IPM을 이용한 방법은 지금까지도 활발히 연구되고 있다.[7][8][9]

본 논문에서는 곡선을 포함한 차선을 검출하기 위해 IPM을 이용하였다. 하지만 IPM은 Bird's eye View를 구하기 위해 높은 연산량을 필요로 하여 차량용 임베디드 시스템에서 실시간으로 동작하기에는 어려움이 있다.

[4][5]를 이용한 처리시간과 곡선검출에 어려움이 있어 본 논문에서는 [6]-[9]와 같은 IPM을 이용하여 직선과 곡선 차선을 검출하였다.

2. 차량 검출 알고리즘 및 하드웨어

전방 차량을 검출하는 방법에는 차량의 외곽선, 그림자, 후미등, 대칭성 등 다양한 특성을 이용하여 연구가 진행되었다[10]. 차량의 특성을 이용하여 차량 후보를 검출한 뒤, 차인지 아닌지 판단

하기 위해 Support Vector Machine(SVM)[11]과 Histogram of Gradient(HoG)[12], Haarlike[13]와 HoG[14], AdaBoost[15]와 같은 다양한 특징점 추출기와 기계학습 알고리즘의 조합이 개발되어 왔다. 최근에는 학습 기반의 딥러닝 알고리즘인 CNN이 영상인식 국제 대회인 Image-Net Scale Vision Recognition Challenge(ILSVRC) 2012[16]에서 기존의 기계학습보다 더 높은 인식률을 기록하였고, 차량 인식 분야에서도 많은 연구가 진행되고 있다. CNN을 이용한 차량검출은 높은 인식률을 가지는 장점이 있지만, 깊은 네트워크 구조와 반복적인 MAC연산으로 연산하는데 오랜 시간이 걸린다. 또한 수많은 학습 데이터에 의해 연산 수행 시 잦은 메모리 접근과 높은 메모리 대역폭을 필요로 한다. 이를 해결하기 위해 고성능의 GPU[17], DSP(Digital Signal Processing), FPGA(Field Programmable Gate Array)를 이용한 연구가 진행되고 있다.

본 논문에서는 높은 검출률을 위해 [11]-[14]와 같은 방법이 아닌 [16]과 같은 CNN을 차량검출에 사용하였다. 하드웨어는 파워가 제한되는 차량용 임베디드 시스템을 위해 GPU를 이용한 방식[17]이 아닌 FPGA를 이용한 CNN 하드웨어를 구현하였다.

III. Front Collision Warning System

FCWS 알고리즘은 주행 중인 차량과 전방 차량의 추돌을 감지하고, 운전자에게 미리 경고해 주는 알고리즘이다. FCWS 알고리즘은 그림 1과 같은 순서로 동작하며 추돌 결과를 출력한다.

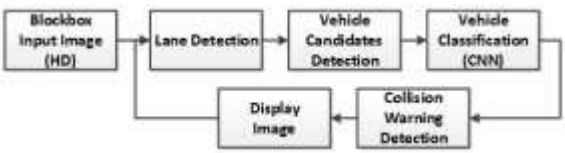


Fig. 1. FCWS Algorithm Flow
그림 1. FCWS 알고리즘 흐름도

1. 차선 검출 (Lane Detection)

FCWS 알고리즘에서 차선 검출은 전방 차량의 차선 위치를 인식하고 운전자의 전방에 있는 차량과의 추돌을 경고하기 위해 사용된다. 연산량을 줄이고 곡선을 포함한 차선을 검출하기 위해 알고리즘은 그림 2와 같은 순서로 진행된다.

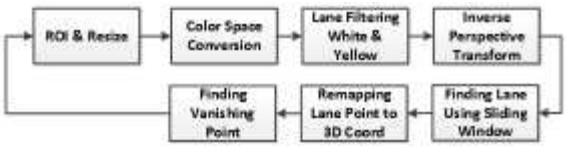


Fig. 2. Lane Detection Flow
그림 2. 차선 검출 흐름도

가. 관심영역 설정(Region of Interest & Resize)

전체 영상 영역 중 차선이 존재하는 영역을 설정하고 HD급 입력 영상을 VGA급으로 낮추어 불필요한 연산을 줄이는데 사용한다. 또한 관심영역 설정은 다른 영역에서의 오검출을 방지하는 효과도 가진다.

나. 색 공간 변경(Color space conversion)

Fig. 3. (a) ROI for lane detection
(b) Lane detection result
그림 3. (a) 차선 검출을 위한 ROI
(b) 차선 검출 결과

RGB방식의 입력 영상에서 YCbCr 색공간으로 변경하여 효과적으로 차선을 검출한다. YCbCr은 밝기성분(Y)에 많은 비트를 할당하고, 색차정보(Cb,CR)를 분리하여 표현하는 색상 모델이다. 따라서 CbCr은 외부에 노출된 차선이 부분적으로 어둡거나 밝더라도 세밀한 밝기 성분을 이용한 차선 검출이 용이하다.

다. 차선 추출(Exract White & Yello Lane)

YCbCr 공간의 이미지에서 도로에 존재하는 흰색 및 노란색 차선을 검출한다. 차선이 도로 노면의 위치와 일조량에 대해서도 검출할 수 있도록 히스토그램을 이용하여 임계값을 결정한 뒤 흰색과 노란색 차선을 추출한다.

라. 역 투영 변환(Inverse Perspective Transform)

IPM을 이용하여 그림 3의 이미지를 그림 4의

(b)와 같은 이미지로 변환하게 되는데, 이때 미리 계산 해놓은 IPM 테이블을 이용하여 Pixel을 새로운 이미지에 맵핑하면 추가 연산 없이 빠른 시간 내에 이미지를 생성한다.

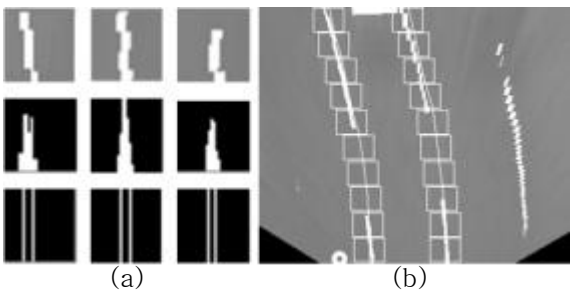
마. 슬라이딩 윈도우(Sliding Window)

IPM을 이용해 생성된 2D 좌표계의 차선 영역 내에서 선분을 검출하기 위해 슬라이딩 윈도우를 사용한다. IPM은 이미 계산된 테이블을 참조하여 2D 이미지를 생성하기 때문에 굴곡이 심한 도로를 주행하게 되면 모든 선분이 직선의 모양을 갖지 못한다. 따라서 일정 크기의 윈도우를 이용하여 차선을 찾아 나간다.

차선 검출은 이미지 내에서 모든 영역을 참고하지 않고 가장 가까운 하단에서부터 윈도우의 히스토그램을 계산하며 차선의 진행 방향으로 윈도우의 중점을 이동시킨다. 이때 차선이 발견되지 않는 부분은 이전 윈도우의 간의 누적 차이 값을 이용해, 새로운 윈도우의 좌표 (x‘, y’)를 획득한다. 그림 4의 (b)와 같이 새로운 (x’, y’)의 값으로 슬라이딩해 나아가며 그림 4의 (a)와 같은 Histogram 기반 차선 검출 방식으로 진행된다.

바. 소실점 찾기(Finding Vanishing Point)

슬라이딩 윈도우를 통해 검출한 좌우 각각 10개의 점을 맵핑시킨다. 맵핑된 차선 정보 를 이용하여 소실점을 구하고 그림 5와 같이 입력 영상에 차선 검출 결과를 출력한다. 생성된 소실점과 차선정보는 앞 차량과의 거리를 측정하거나 차량의 위치를 판단할 때 사용된다.



(a) (b)

Fig. 4. (a) Window within horizontal histogram result
(b) IPM based lane detection result
그림 4. (a) 윈도우 내의 수평 히스토그램
(b) IPM 기반 차선 검출 결과

2. Vehicle Candidates Detection

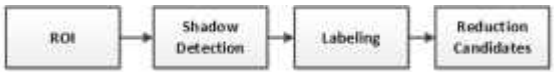


Fig. 5. Vehicle Candidates Detection Flow
그림 5. 차량 후보 검출 흐름도

차량의 특성을 이용하여 차량 후보를 검출하고, CNN은 차량 검출을 하는데 사용된다. 차량 후보 검출 알고리즘은 그림 5와 같은 순서로 진행된다.

가. 관심 영역 설정(Region Of Interest)

관심 영역은 차선 검출에 사용된 관심영역과 동일하고, 차량 후보의 오검출을 방지한다.

나. 그림자 검출(Shadow Detection)

차량은 모두 차량 아래에 그림자를 가지는 특성을 가진다. 또한 하단부의 그림자는 차량의 하단 시작 부분을 알 수 있는 특징을 가지고 있다. 그림자는 차선 검출과 동일한 방식으로 도로 노면의 히스토그램을 구하고 Gray-Scale의 임계값을 적용하여 그림자 영역을 검출한다.

다. 레이블링(Labeling)

레이블링 알고리즘은 인접한 픽셀끼리 연결하여 하나의 객체로 인식시키는 방법이다. 최소, 최대 연결 가능한 픽셀의 개수를 설정하여 너무 크거나 작은 객체를 제거시켜 오검출을 방지하였다.

라. 차량 후보 감소(Reduction Candidates)

레이블링을 통해 결정된 차량 후보는 ROI 내에 존재하고, 전방, 좌우측 차선에서 존재한다. 먼저 각 차선에서 (x, y)위치와 차량 후보의 폭 정보를 이용해 해당영역을 차량 후보에서 제외시킨다. 또한 차량 후보 중 추돌에 영향을 주는 차량은 가장 가까운 전방 차량과 좌우측에서 전방으로 들어오는 차량에 해당된다. 최종적으로 각 차량이 중앙 차선 내 90% 이상을 점유하고 있을 경우, 차량 후보로 사용된다.

3. Vehicle Detection using CNN

차량 후보 데이터로 차량의 후면을 인식하기 위해 CNN을 사용하였다. 차량 검출을 위한 CNN의 네트워크 모델은 그림 6과 같은 구조로 구성하였다. 차량 검출용 CNN은 오프라인 환경에서 학습을

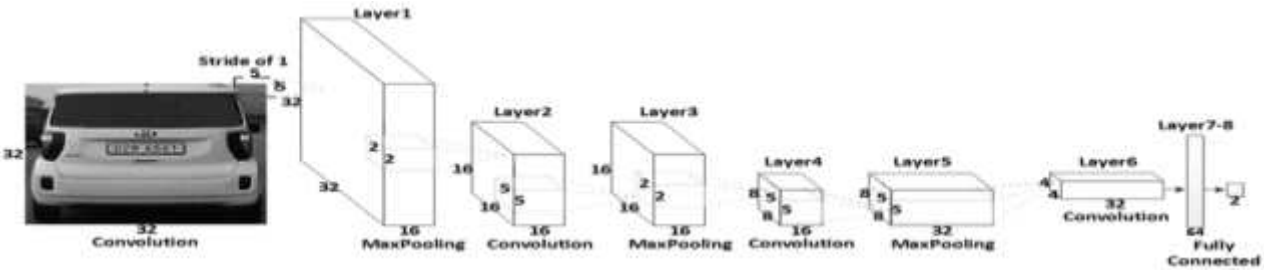


Fig. 6. CNN Network model for vehicle detection
그림 6. 차량 검출을 위한 CNN 네트워크 모델

하였고, 차량의 후면 이미지 3000장을 학습하였다. 학습한 파라미터를 이용하여 Caltech'99 차량 데이터셋에 테스트 한 결과 99.2%의 검출률을 얻었다.

학습된 데이터를 바탕으로 CNN 연산을 수행하고, 32x32의 Gray이미지를 입력 데이터로 사용한다. 총 8개의 레이어로 구성하였으며, 각각 Convolution, Maxpooling 레이어 3쌍과 2개의 Fully-Connected로 구성 되어있다. 마지막으로 Fully-Connected 레이어의 결과는 Soft-max 결과를 통해 차량인지 아닌지 판단하게 된다.

4. Warning Detection

그림 7와 같이 앞서 검출한 차선과 차량 위치를 기반으로 속도 및 거리를 계산한다. 전방 차량과의 거리는 차량의 주행 중인 W_1과 전방 차량의 폭 W_2를 이용하여 계산한다. 식 (1)을 이용하여 거리를 계산하고[17], 식 (2)를 이용하여 30 Frame 동안의 속도 평균과 상대 속도를 계산한다. 계산한 속도와 거리를 이용해 식 (3)과 같이 Time-To-Collision(TTC)를 계산하고 TTC가 3초 이내일 경우, 운전자에게 추돌 경보를 알려준다.

$$D(t) = 3.2955 * (W_1 / W_2) - 3.0852 \quad (1)$$

$$V(t) = \sum_{0}^{30} d(t)/30 - \sum_{0}^{30} d(t-30)/30 \quad (2)$$

$$TTC = D(t) / V(t) \quad (3)$$

Ⅳ CNN Hardware

이번 장에서는 3장에서 설명한 CNN 네트워크를 하드웨어로 구현한 방법 및 동작과정을 블록도를 통해 설명한다.

Fig. 7. Distance & TTC calculation result
그림 7. 거리 및 TTC 계산 결과

1. CNN 하드웨어

CNN 하드웨어는 그림 8과 같은 방식으로 FPGA에 구현하였다. Convolution + MaxPooling 모듈과 Fully-Connected로 구성하였다. 8개의 레이어 전체를 FPGA에 구현해 본 결과, 85K Logic Elements(LEs) FPGA에 생성이 불가능하여, 공통적으로 수행하는 Conv + MaxP와 FC를 모듈로 생성하여 각 레이어마다 모듈을 재사용 하는 방식으로 구현하였다. 또한 각 모듈은 블록단위로 데이터를 가져와 재사용 하여 DDR 메모리에 대한 접근 횟수를 줄이는 방식을 사용하였다. Conv + MaxP 모듈에서 가져오는 입력 데이터의 블록 크기는 MxM(12x12), Weight와 Convolution연산에 필요한 Kernel의 크기는 동일한 KxK(5x5)를 가져와 병렬로 수행된다. 블록단위의 Convolution 과 2x2 MaxPooling, Bias 연산 수행 뒤, 결과 값을 DDR 메모리에 저장한다. 위의 방법을 통해 Conv + MaxP 모듈을 3번 수행할 경우, Conv + MaxP 모듈이 종료된다.

Conv + MaxP 모듈 수행 후, FC 모듈은 동일한 크기 O(32) 만큼 입력 데이터와 Weight를 가져오고, Inner Product에 곱셈기의 개수 J(4)만큼 병렬로 연산을 수행한다. 블록단위의 연산이 종료된 이후 결과값을 DDR 메모리에 저장하고 다시 한 번 모듈을 재사용 한 뒤 FC가 종료된다. FC의 마지막 결과값 2개는 ARM-Core에서 Soft-max를 통해 CNN결과가 차일 확률을 판단하게 된다.

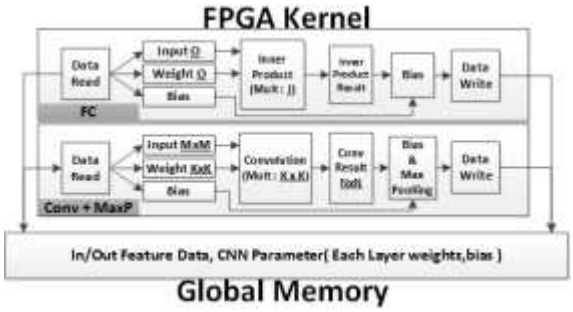


Fig. 8. CNN Hardware Block Design
그림 8. CNN 하드웨어 블록 디자인

2. Convolution + MaxPooling Module

Conv + MaxP 모듈은 CNN 네트워크 모델의 1-6 Layer를 수행하는데 사용된다. Conv + MaxP에서 사용한 블록단위의 Convolution 연산 및 MaxPooling의 블록도 아래 그림 9과 같다.

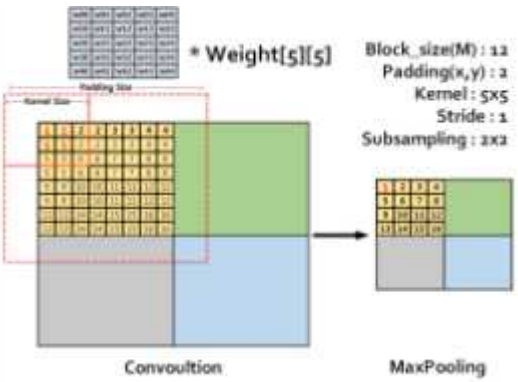


Fig. 9. Convolution + MaxPooling Module
그림 9. Convolution + MaxPooling 모듈

Convolution연산은 5x5의 Weight를 이용해 수행하고 Zero Padding을 x, y방향으로 2만큼 생성하고 1만큼 Stride된다. Convolution의 블록단위 연산은 그림 9의 점선 네모와 같은 방법으로 수행된다. Padding을 고려하여 12x12의 블록을 DDR메모리로부터 FPGA의 내부 Block RAM으로 Load하고, 5x5의 Convolution연산에 의한 8x8의 결과를 Feature map Block RAM에 저장한다. 한 번의 블록 연산결과는 그림 10와 같이 Input Feature Depth만큼 누적 연산한 뒤, Bias연산을 수행하고 2x2 MaxPooling 연산을 통해 Output Feature를 만들어낸다. Output Feature는 최종적으로 4x4의 블록 결과 값을 얻게 되고 다음 Layer의 Input Feature Data로 사용되기 위해 DDR메모리에 저장한다.

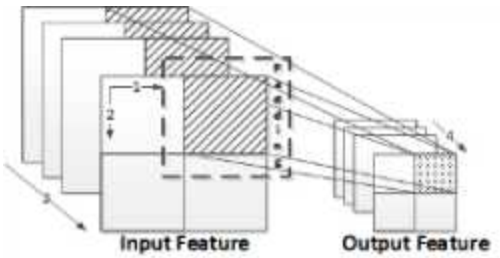


Fig. 10. Convolution + MaxPooling Block Sequence
그림 10. Convolution + MaxPooling 블록 순서

3. Fully-Connected Module

FC모듈은 Convolutional Layer와는 달리 모든 Input Feature Data가 연결되어 내적 연산을 수행하게 된다. DDR 메모리에서 한 번 접근할 때 마다, 많은 데이터를 가져오기 위해 32(O) 크기만큼 Input Feature와 Weight를 Load하고, 곱셈기(4) 개수만큼 내적 연산을 수행한다. 모든 Input Feature Depth만큼 Weight와 내적연산을 수행하면 하나의 Output Feature Data에 대한 결과가 계산되고, DDR 메모리에 다음 FC Layer의 Input Feature Data에 Store된다. 이를 반복하여 Output Feature 개수만큼 수행하게 되면 FC모듈의 연산이 종료된다. Conv + MaxP이 합성된 후 남은 공간을 FC모듈로 사용하기 때문에 하드웨어의 크기가 더 커질수록 버퍼의 크기와 곱셈기의 크기를 증가시켜 FC모듈의 성능을 향상 시킬 수 있다.

V 실험 환경 및 분석

이번 장은 구현한 FCWS의 수행시간, 검출률, CNN 하드웨어 성능 그리고 장치 별 FCWS의 처리속도 대비 에너지 효율에 대한 분석을 진행한다. 실험에 사용된 환경은 Terasic社 DE1-SoC 보드이며 ARM-Coretex A9, Cyclone V FPGA, DDR 1GB로 구성되어있다. FPGA에 CNN을 구현하기 위해 Intel FPGA for OpenCL[18]을 이용하였다. 입력 영상은 HD이며 검증을 위해 보드의 VGA Port를 이용하여 추돌 유무를 확인하였다.

1. FCWS

가. FCWS 수행 시간 분석

표 1은 FCWS의 알고리즘 및 SW, HW 구성에 따른 처리속도를 나타낸다. 본 논문의 FCWS는

HD급에서 동작할 수 있게 구현하였다. 차선 검출에 IPM과 슬라이딩 윈도우 기법으로 처리 속도를 증가시켰다. 차량 후보 검출은 작은 ROI 내에서 Morphology 연산 없이 수행할 수 있도록 Labeling 단계를 거치고, 후보 감소를 통해 수행시간을 감소시켰다. Vehicle Detection(CNN)은 CNN이 FPGA에서 수행되는 2.641ms와 32x32 Gray 입력데이터 변형, -1 ~ +1 입력데이터 맵핑, Soft-max와 차량 업데이트 유무를 ARM-Core에서 동작시키는 시간을 포함한다.

Table 1. Analysis of FCWS processing time
표 1. FCWS 수행시간 분석

Algorithm	[19] (ms)	Proposed (SW) (ms)	Proposed (SW + HW) (ms)
Lane Detection	23.2	8.58	8.58
Vehicle Candidate Detection	29.8	3.53	4.88
Vehicle Detection(CNN)		25.06	5.78
Collision Warning Detection	-	0.1	0.1
Display	-	3.3	3.3
Total	53	41.07	22.64

본 논문에서 제안하는 FCWS 알고리즘의 수행속도는 SW로 동작 시 41.07ms, SW + HW로 동작 시 22.64ms로 동작한다. [19]은 차선 인식 및 차량 검출을 FPGA + DSP 구조로 처리하였다. FPGA를 이용하여 입력 영상처리 및 전처리로 사용하고, DSP는 알고리즘을 처리하는 구조로 53ms 동작한다. 차량 검출은 동일하게 그림자 기반으로 검출하였으나, 제안하는 구조는 CNN을 이용하고도 [19]보다 처리속도 면에서 더 빠른 것으로 나타났다.

가. 검출률

본 논문의 FCWS는 주행 중인 전방차량에 대해 검출을 실시하였다. 촬영 환경은 국내 북부간선도로이며, 차량 내부 블랙박스 영상을 이용하였고 8개의 서로 다른 영상을 이용하였다. 차량 검출률 계산은 아래의 4번 식을 이용하였고, 결과는 표 2와 같다. [19]논문의 그림자를 이용한 차량 검출률은 약 90%로 본 논문의 차량 검출률이 더 높은 것을 확인할 수 있다.

$$Detection\,Rate = \frac{TP}{TP+FP} * 100(\%) \quad (4)$$

(TP : # of detected vehicles)
(FP : # of not detected vehicles)

Table 2. FCWS Detection rate
표 2. FCWS 검출률

	Test Frames	Rate(%)
Vehicle Detection	6107	97.05

2. CNN Hardware

3장의 CNN을 SW로 동작시켜보고, FPGA를 이용하여 가속을 한 결과는 표 3과 같다. Conv + MaxP모듈은 Block 12x12, Kernel 25x25, 2x2 Padding으로 동작하고, FC모듈은 32 Block, 4개의 곱셈기로 동작한다. 그 결과 CNN모델을 하드웨어로 처리하는데 걸리는 시간은 총 2.641ms이며, 같은 구조의 CNN모델을 PC환경에서 동작시킬 경우 1.511ms로 약 1.74배 느리지만, 임베디드 환경의 ARM-A9에서는 24.622ms로 9.32배 빠른 것을 확인하였다. 또한 CNN하드웨어를 Verilog HDL 방식으로 구현한 [20] 구조보다 CNN을 처리하는데 약 0.2ms정도 빠른 것으로 나타났다.

Table 3. CNN Processing Time
표 3. CNN 처리속도

	SW Time(ms)		HW Times(ms)	
Device	Intel i5-4690	ARM-A9	Xilinx XC7Z045 [20]	Intel FPGA Cyclone V
Frequency	3.5GHz	800MHz	100MHz	145MHz
Write	-	-	0.645	0.02
Conv1 MaxP1	0.255	4.765	0.176	0.357
Conv2 MaxP2	0.73	15.248	0.659	1.104
Conv3 MaxP3	0.366	3.805	0.328	0.594
FC1	0.16	0.804	0.327	0.519
FC2			0.001	0.045
Read	-	-	0.00027	0.002
Total	1.511	24.622	2.812	2.641

Conv + MaxP, FC 모듈단위의 연산에서 Float방식을 Fixed 방식으로 변경하기 위해 Input Feature data, Weight, Bias의 bit를 변경해 가며, 최종 Fully-connected의 결과 값을 기존 Float 방식과 비교하였다. 그림 11은 데이터 비트에 따른 CNN 결과 에러율을 나타낸다. 16bit로 Fixed할 경우

에러율이 1%이내이고, 줄어들수록 에러율의 폭이 커져 16bit를 이용한 Fixed를 진행하였다.

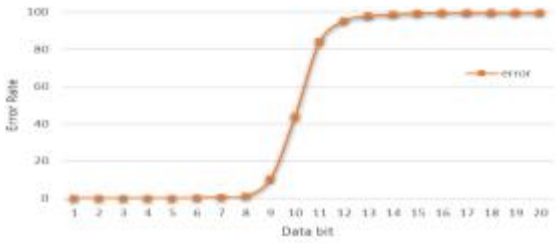


Fig. 11. Error rate according to CNN data bit
그림 11. CNN 데이터 비트에 따른 에러율

아래의 표 4는 Xilinx社에서 Verilog HDL로 구현한 하드웨어와[20] 본 논문에서 Intel FPGA for OpenCL을 이용하여 구현한 하드웨어의 사용량을 나타낸다. 동일한 CNN 네트워크를 Verilog HDL로 FPGA에 구현한 방식은 모든 Layer를 FPGA에 구현하여 연산하였다. 또한, Fixed(20-26bit)방식의 데이터 타입을 이용하여 구현하였으나, 약 14만 개의 LUT를 사용하였고, Register 또한 약 10만개를 사용하였다.

Table 4. CNN Hardware usage comparison
표 4. CNN 하드웨어 사용량 비교

Design Scheme	Verilog HDL[20]	Intel FPGA with OpenCL	
Vendor	Xilinx	Intel FPGA	
FPGA	XC7Z045	Cyclone V	
Frequency	100MHz	119MHz	141MHz
Data Type	Fixed (20-26bit)	Float (32bit)	Fixed (16bit)
Registers	97,799	71807	46882
LUT/ ALM	144,393	29,827	19,136
Memory Bits(KB)	218	157	65
RAM Blocks	-	328	203
DSP	9	42	16

Intel FPGA for OpenCL을 이용한 방식은 Float(32bit)를 이용한 방식과 FPGA 최적화를 위해 Fixed(16bit) 방식의 데이터 타입으로 구현하였다. Float방식은 32bit의 데이터 타입을 사용하나 Verilog HDL방식과 다르게 모든 Layer를 구현하지 않고 Conv + MaxP, FC 모듈만 구현하여 사용하였기 때문에 Register, LUT/ALM, Memory의 낮은 FPGA 사용률을 보였다. DSP의 경우, Verilog HDL은 LUT를 이용하여 곱셈기를 구현하였기 때문에 DDR 컨트롤 목적으로 9개의 DSP만 사용되었다. Intel FPGA for OpenCL은 Float타입의 곱셈기를 하나의 DSP로 생성하기 때문에, 34개 이상 사용되었다. 또한 블록단위의 메모리 접근을 위한 Input/Output Feature 메모리 접근 방법 때문에 DDR 컨트롤러 생성에 8개의 DSP가 사용되었다. Fixed 방식은 Float 방식에 비해 전체적으로 절반 정도의 FPGA 사용량을 기록하였고, Kernel의 복잡도에 따라 설정되는 Frequency 또한 빨라진 결과를 확인하였다. DSP의 경우 하나의 DSP에서 2개의 18x18 곱셈기를 구현할 수 있어 DSP 사용량을 감소시킬 수 있었다.

3. Power analysis for effiency

표 5는 FCWS를 PC 기반의 SW, DE1-SoC 기반의 ARM-Core, FPGA로 수행하였을 때 성능을 비교한 결과를 나타내고, 그림 15는 DE1-SoC를

Table 5. Energy efficiency compared to throughput
표 5. 처리속도 대비 에너지 효율

Device	Intel i5-4690	DE1-SoC (ARM Core)	DE1-SoC (ARM Core* + FPGA**)
Frequency	3.5GHz	800MHz	800*/141**MHz
Time(ms)	7.32	41.07	21.64
FPS	136.61	24.35	46.21
Power(W)	80.00	7.42	7.76
Efficiency (FPS/W)	1.71	3.28	5.69
Comparision	x1	x1.92	x3.33

이용한 FCWS의 동작 결과를 나타낸다. 고성능 PC 기반의 SW는 FCWS를 136 FPS로 빠르게 동작시키지만 80W로 많은 전력을 소모한다. 그에 비해 임베디드 환경의 DE1-SoC는 연산을 수행하지 않을 시, 6.9W, ARM-Core를 이용하여 FCWS 동작 시 7.42W, FPGA를 같이 사용할 시 7.76W의 전력이 사용된다. 이에 따라 처리속도 대비 에너지 효율을 FPS/W로 정의했을 때, DE1-SoC에서 CNN을 FPGA로 수행한 결과는 PC 대비 3.33배의 효율을 나타낸다. 분석 결과는 고성능 PC와 비교하여 낮은 Clock Frequency를 가지지만, 임베디드 시스템 목적에 맞게 실시간 처리가 가능한

FCWS로 에너지 효율 면에서 더 나은 결과를 나타낸다.

Ⅵ 결론

FCWS는 차량용 임베디드 시스템에서 동작하는 알고리즘으로 영상처리 기반의 실시간 동작과 앞차와의 추돌 경보를 정확히 경고하기 위해 높은 확률의 차량 검출률이 필요하다. 따라서 본 논문에서는 알고리즘을 DE1-SoC의 ARM-Core와 FPGA에 할당하여 실시간 처리 가능하게 하였다. ARM-Core는 차선 검출과 차량 후보 검출 그리고 추돌 경보를 수행하였다. 차량검출에 사용된 CNN은 Intel FPGA Cyclone V에 구현하였으며, Convolution + MaxPooling, Fully-Connected 모듈을 구성하여 블록단위의 연산을 수행할 수 있도록 하였다. HW사용량은 기존 Float(32bit) 연산 방식에서 Fixed(16bit)로 변경함으로써 하드웨어 크기가 절반으로 줄어들었다. CNN 하드웨어는 2.641ms로 동작하고, Verilog HDL로 구현한 모델보다 약 0.2ms정도 빠른 것을 확인했다. 검출한 차량은 거리계산 및 속도, TTC 계산을 통해 추돌 유무를 판단하였고 VGA 모니터를 통해 검증하였다. CNN을 이용한 차량 검출률은 97.05%로 높은 검출률을 기록하였다. CPU-FPGA를 이용한 FCWS는 전체 22.64ms로 동작하여 44FPS로 실시간 동작이 되는 것을 확인하였고, PC보다 FCWS의 처리속도 대비 에너지효율은 3.33배 더 나은 결과를 얻었다.

향후 과제로는 여분의 DSP, Memory, LUT등을 이용하여 Conv + MaxP, FC 모듈의 곱셈기의 개수를 늘려 속도를 향상 시킬 수 있다. 또는 Full-HD급 이상의 영상에서도 동작할 수 있도록, 차선검출, 차량후보 검출에 동일하게 사용 가능한 하드웨어를 구현하여 실시간 처리가 가능하게 하는 것이다.

References

[1] Karen Aldana, “NHTSA Announces Model Year 2012 Vehicles to be Rated Under Government 5-Star Safety Ratings Program,“ https://www.nhtsa.gov/staticfiles/communications/pdf/nhtsa1711.pdf

[2] Gao, Yongbin, and Hyo Jong Lee. "Vehicle Make Recognition Based on Convolutional Neural Network," *Information Science and Security (ICISS), 2015 2nd International Conference on. IEEE*, 2015. DOI:10.1109/ICISSEC.2015.7371039

[3] A. Krizhevsky, I. Sutskever, G. E. Hinton and et al., “ImageNet classification with deep convolutional neural networks,” *in Proc. Neural Information Processing Systems (NIPS'12), 2012*

[4] Cuneyt Akinlar and Cihan Topal, “EDLines: A real-time line segment detector with a false detection control,” *Pattern Recognition Letters,* vol. 32, no. 13, pp. 1633 - 1642, 2011 DOI:10.1016/j.patrec.2011.06.001

[5] X. Lu, J. Yao, K. Li, and L. Li, “Cannylines: A parameter-free line segment detector,” *In 2015 IEEE International Conference on Image Processing (ICIP),* pp.507 - 511, Sept, 2015. DOI:10.1109/ICIP.2015.7350850

[6] Massimo Bertozzi and Alberto Broggi, “Real-Time Lane and Obstacle Detection on the GOLD System,” *IEEE International Symposium on Computer Vision, 1995* DOI:10.1109/IVS. 1996.566380

[7] H.Tan, Y. Zhou, Y.Zhu, D.Yao, and K. Li, “A novel curve lane detection based on Improved River Flow and RANSAC,” *In Intelligent Transportation Systems (ITSC), 2014 IEEE 17th International Conference on,* pp.133-138. IEEE, 2014 DOI:10.1109/ITSC.2014.6957679

[8] J.Wang, T.Mei, B. Kong, and H.Wei, “An approach of lane detection based on Inverse Perspective Mapping,” *In Intelligent Transportation Systems (ITSC), 2014 IEEE 17th International Conference on* pp. 35-38. IEEE, 2014. DOI:10.1109/ITSC.2014.6957662

[9] Kaur, Gurjyot, and Amit Chhabra, "Curved Lane Detection using Improved Hough

Transform and CLAHE in a Multi-Channel ROI," *International Journal of Computer Applications 122.13* ,2015.
[10] Nur Shazwani A "Vehicle Detection Based on Underneath Vehicle Shadow Using Edge Features," *2016 6th IEEE International Conference on Control System, Computing and Engineering*,Penang, Malaysia, pp.25 - 27, Nov, 2016. DOI:10.1109/ICCSCE.2016.7893608
[11] C. Cortes and V. Vapnik, "Support-Vector Network," *Machine Learning*, Vol. 20, No. 3 pp.273-297, May, 1995. DOI:10.1007/BF00994018
[12] Hao Yu, "Vision-based Lane Marking Detection and Moving Vehicle Detection," *2016 8th International Conference on Intelligent Human-Machine Systems and Cybernetics,*2016. DOI: 10.1109/IHMSC.2016.240
[13] Sungji Han, "Vehicle Detection Method Using Haar-like Feature on Real Time System," *World Academy of Science, Engineering and Technology 59,* 2009.
[14] Zhipeng Di, "Forward Collision Warning System Based on Vehicle Detection and Tracking," *2016 International Conference on Optoelectronics and Image Processing,* 2016. DOI: 10.1109/OPTIP.2016.7528490
[15] O. Russakovsky et. al, "ImageNet Large Sclae Visual Recognition Challenge," *International Journal of Computer Vision,* Vol.115, No.3, pp.211-252, Dec.2015. DOI:10.1007/s11263-015-0816-y
[16] D. Strigl, K. Kofler, and S. Podlipnig. "Performance and scalability of gpu-based convolutional neural networks. Parallel," *Distributed, and Network-Based Processing, Euromicro Conference* on, 0:317 - 324, 2010 DOI: 10.1109/PDP.2010.43
[17] Kim, Jin-Soo, "Effective Road Distance Estimation Using a Vehicle-attached Black Box Camera," *Journal of the Korea Institute of Information and Communication Engineering.* vol.19, no.3, pp.651-658, 2015. DOI : 10.6109/jkiice.2015.19.3.651
[18] Chris Rauer and George S.Powley "Accelerating Genomics Research with OpenCL™ and FPGAs," https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/wp/wp-accelerating-genomics-opencl-fpgas.pdf
[19] Kim Il-Ho, Kim Gyeong-Hwan, "FPGA-DSP Based Implementation of Lane and Vehicle Detection," *The Journal of Korean Institute of Communications and Information Sciences,*vol.36, no.12, pp.727-737, Dce. 2011.
[20] Bang,Ji-Won,Jeong,Yong-Jin,"A Real-Time Hardware Design of CNN for Vehicle Detection ," *j.inst.Korean.electr.electron.eng.* vol.20, no.4, pp.351-360, Dec. 2016. DOI:10.7471/ikeee.2016.20.4.351

BIOGRAPHY

Yongjin Jeong (Member)

1983 : BS degree in Control and Instrumentation Engineering, Seoul National University.
1995 : MS, PhD degree in Electrical and Computer Engineering, University of Massachusetts, Amherst
1983~1989 : Research Engineer, ETRI
1995~1999 : Chief Researcher, Samsung Electronics
1999~current : Professor, Dept. of Electronics and Communications Engineering, Kwangwoon Univ.

Sungwoo Han (Student Member)

2016 : BS degree in Electronics and Communications Engineering, Kwangwoon University.
2016~ : Course of MS in Electronics and Communications Engineering, Kwangwoon University.