http://dx.doi.org/10.5392/JKCA.2013.13.04.044

SIFT의 descriptor를 위한 sin/cos 프로세서의 구현

Implementation of sin/cos Processor for Descriptor on SIFT

김영진, 이현수
경희대학교 컴퓨터공학과

Young-Jin Kim(jerryjin@khu.ac.kr), Hyon Soo Lee(leehs@khu.ac.kr)

요약

SIFT(Scale Invariant Feature Transform) 알고리즘은 현재 비디오 감시카메라, 자율 주행시스템 등과 같은 영상 시스템에서 많이 사용되고 있다. SIFT 알고리즘에서 연산량과 연산시간이 가장 많이 필요한 부분이 descriptor의 sin/cos 함수를 연산하는 부분이다. 그러므로 본 논문에서는 SIFT 알고리즘에 사용되는 descriptor를 위한 sin/cos 함수를 하드웨어로 구현하였다. Verilog-HDL 언어를 사용하여 FPGA로 구현하고 그 성능을 분석한다. Xilinx Spartan 2E(XC2S200E-PQ208-6) 를 사용하여 구현하였을때, 149 Slices에 233 LUTs가 소모되었으며, 최대 주파수는 60.01MHz로 동작하였다. 또한 descriptor에 적용하여 소프트웨어와 비교 하였을 때 40배 정도의 빠른 성능 향상을 얻었다.

■ 중심어 : | SIFT | Descriptor | sin/cos 프로세서 | FPGA |

Abstract

The SIFT algorithm is being actively researched for various image processing applications including video surveillance and autonomous vehicle navigation. The computation of sin/cos function is the most cost part needed in whole computational complexity and time for SIFT descriptor. In this paper, we implement a hardware to sin/cos function of descriptor on sift feature detection algorithm. The proposed Sin/Cosine processor is coded in Verilog and synthesized and simulated using Xilinx ISE 9.2i. The processor is mapped onto the device Spartan 2E (XC2S200E-PQ208-6). It consumes 149 slices, 233 LUTs and attains a maximum operation frequency of 60.01 MHz. As compared with the software realization, our FPGA circuit can achieve the speed improvement by 40 times in average.

■ keyword : | SIFT | Descriptor | sin/cos processor | FPGA |

I. 서론

최근 객체인식은 비디오 감시카메라[1], 자율 주행 시스템[2], 지능 모바일 로봇[3] 등과 같은 영상 시스템에서 많이 사용되고 있다. 객체인식은 입력된 영상과 비교 영상을 매칭시켜 이루어지며 매칭을 위해 객체의 위치 정보와 그 주변의 정보를 담고 있는 특징점들을 사용하게 된다. 그 중에서 가장 많이 사용되는 특징점 추출 방법이 SIFT(Scale Invariant Feature Transform) 알고리즘이다[4]. SIFT 알고리즘은 영상 데이터로부터 객체의 모서리나 꼭지점과 같은 부분에 생성되는 특징점의 벡터 성분을 추출하는 방법으로써 패턴 인식 방법

접수번호 : #130313-003
접수일자 : 2013년 03월 13일
심사완료일 : 2013년 04월 02일
교신저자 : 이현수, e-mail : leehs@khu.ac.kr

과 달리 비교 영상의 크기 변화 또는 회전으로 인한 변형에 대해 뛰어난 매칭성능을 갖는 알고리즘이다.

SIFT는 반복적인 연산의 속성을 가지고 있어 이미지가 커지거나 특징점이 많아질수록 연산량이 비례하여 폭발적으로 증가함으로 연산속도가 느려지는 단점으로 인해 실시간 처리가 어려워진다. 그러므로 SIFT 알고리즘을 빠르게 연산하기 위한 하드웨어로 구현하는 방법들이 다음과 같이 제안되었다. 첫 번째로 하드웨어/소프트웨어를 co-design하는 방법[5], 두 번째로 ARM 프로세서의 임베디드 환경에서 실시간 처리를 위한 하드웨어 구현 방법[6], 그리고 마지막으로 NIOS 프로세서를 사용하는 병렬 하드웨어 구현 방법[7]등이 있다.

기존 연구들에서 실시간 처리를 위해 SIFT 알고리즘의 하드웨어 구현 방법에 대해서 제안하였지만, descriptor를 구현하는데 있어서 삼각함수와 나눗셈 연산이 필요하여 descriptor는 소프트웨어로 처리하였다. 하지만 descriptor는 평균적으로 SIFT 알고리즘 연산시간의 65%를 차지한다[8].

또한 descriptor의 연산시간 중 99%가 사인(Sine)과 코사인(Cosine) 함수를 계산하는데 필요하다. 그러므로 본 논문에서는 descriptor에 필요한 삼각함수를 계산하기 위한 개선된 사인과 코사인 함수를 하드웨어로 구현하였다. 사인과 코사인 함수에서 나눗셈 연산을 효율적으로 구현하기 위해서 테일러(Talor) 급수의 Maclaurin 시리즈를 사용하여 구현하였으며, 또한 연산시간을 빠르게 처리하기 위해서 파이프라인 구조를 제안하여 연산시간을 빠르게 처리 하였다.

본 논문의 구성은 II장에서 관련연구로 SIFT 알고리즘에 대해서 알아보고, III장에서는 사인/코사인 알고리즘에 대해서 기술한다. VI장에서는 하드웨어 구조에 대해서 제안하였으며, V장에서는 구현방법과 기존 구조와의 성능 비교를 하였다. 마지막으로 VI장에서는 논문의 결론을 맺었다.

II. SIFT 알고리즘

일반적으로 SIFT 알고리즘은 다음과 같은 네가지의 과정으로 이루어져 있다. Scale-space extrema detection, Key-point localization, Orientation assignment 그리고 Key-point description(SIFT description)이다.

각 단계별 동작은 다음과 같다. 첫 번째로 영상에서 키포인트(key-point)를 찾는다. 키포인트를 찾기 위해서 서로 다른 크기의 영상을 정해진 분산에 따라 피라미드 가우시안 영상을 구한다. 각 스케일별로 두 개의 가우시안 영상의 차인 DoG(Different of Gaussian)를 계산한 후 현재의 DoG의 픽셀과 이웃한 두 개의 DoG 사이에서 3x3 영역의 26개 주변 픽셀을 비교하여 극대값(local maximum) 또는 극소값(local minimum)을 구한다. 구해진 극값이 키포인트가 된다. 두 번째로 키포인트의 후보군에서 낮은 대비를 갖는 점, 에지성분이 강한 키포인트와 대비(contrast)가 불안정한 키포인트를 후보군에서 제외하고 최종적으로 안정적인 키포인트를 특징점으로 결정한다. 세 번째로 정해진 특징점의 주변 영역을 계산하여 기울기방향(gradient orientation)에 따른 히스토그램을 구한다. 각각 10도씩 36단계로 이루어진 히스토그램을 구하는데 그 중 기준(threshold)을 넘는 방향이 그 특징점의 방향(Orientation)이 되며, 그때의 크기를 특징점의 영역으로 결정한다. 만약 기준을 넘는 방향이 두 개 이상인 경우 기준을 넘는 개수에 맞추어 특징점을 확장한다. 마지막으로 [그림 1]과 같이 특징점의 방향에 맞추어 결정된 영역에 있는 픽셀들을 4x4 배열에 할당한 후 각 배열을 8방향으로 재구성한다. 재구성된 128개의 값(128 bin)이 그 특징점의 descriptor vector가 된다. 모든 특징점에 대해서 각각 descriptor vector를 구하는 작업이 SIFT descriptor이다.

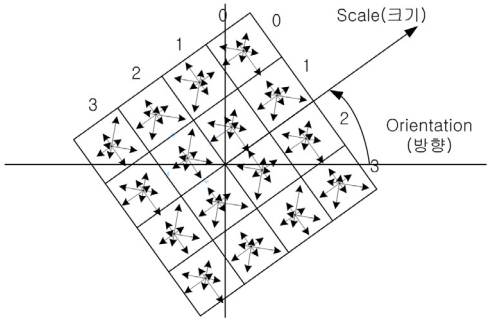

그림 1. 특징점에 대한 4x4 description

이와 같은 네가지 과정을 통해 특징점을 추출하게 되는데 영상이 커지거나 복잡해지면 반복되는 연산이 많아 실시간 처리가 어려워지는 단점을 갖는다. 그러므로 하드웨어를 통해 실시간 처리를 할 수 있다.

이전 연구들에서는 네 번째 과정의 연산에서 각 특징점의 기울기(gradient)와 픽셀의 기울기를 맞추기 위해서 삼각함수 연산이 필요하여 하드웨어로 구현하지 않고 소프트웨어로 연산을 수행하였다. 그러나 각 특징점의 개수에 따른 연산시간을 측정해본 결과 [그림 2]와 같이 descriptor 부분이 모든 연산의 대략 65% 이상을 차지함을 볼 수 있다.

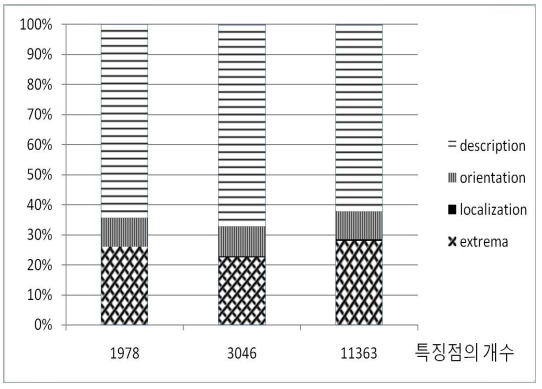

그림 2. 특징점의 개수에 따른 SIFT 연산시간

descriptor 연산은 [그림 1]과 같이 특징점 주변의 픽셀들이 특징점 영역의 크기(Scale)에 포함되는지 판단한 후 각 픽셀의 기울기(Orientation)를 특징점의 기울기와 동일하게 맞추고 4x4 배열의 128bin에 각 픽셀의 값을 추가하여야 하는 연산으로 [표 1]은 각각의 연산이 차지하는 비율을 나타낸다. 연산중에서 가장 많은 시간을 차지하는 연산이 회전변환이다. 회전변환의 처리를 위해서 사인과 코사인 연산이 필요하다. descriptor의 전체 연산에서 사인과 코사인 연산이 연산시간의 98% 이상을 차지한다.

따라서, 본 논문에서는 descriptor 연산을 빠르게 연산하기 위해서 가장 많은 연산시간을 차지하는 사인과 코사인 연산에 대한 하드웨어를 제안한다.

표 1. Descriptor 연산 수행 처리상의 각 단계별 연산시간 비율

특징점 개수 / 연산단계	1978	3046	11363
영역판단	0.00%	0.00%	0.07%
회전변환	99.27%	98.83%	98.97%
128 bin 추가	0.73%	1.17%	0.96%

III. 사인(Sine)/코사인(Cosine) 알고리즘

본 논문에서는 descriptor의 회전변환을 위해서 테일러 급수를 이용한 사인과 코사인 함수를 사용한다. Maclaurin 시리즈 확장을 사용하여 사인과 코사인 함수를 나타내면 식(1)과 식(2)와 같다 [9].

$$\sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots, \text{for } all\ x \quad (1)$$

$$\cos(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \ldots, \text{for } all\ x \quad (2)$$

위의 식에서 허용오차 범위 내에서 적당한 n을 구하는 것이 무엇보다 중요하다.

본 논문에서는 함수의 출력 오차와 descriptor의 벡터 오류를 가지고 n을 결정하였다. 이 두 가지는 SIFT 결과에 직접적인 영향을 주는 요소이다.

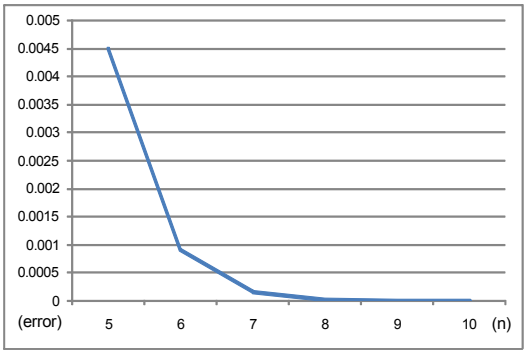

그림 4. n에 따른 연산의 최대 오차($-\pi \sim \pi$)

우선 첫 번째로 n에 따른 함수의 최대 오차를 계산하였는데 [그림 4]와 같다. x축은 n을 나타내며, y축은 최대 오차값이다. 여기서 입력은 모든 각도의 계산을 위해서 $-\pi \sim \pi$까지의 값을 입력하였다. 함수의 최대 오차는 n이 5일때 0.0045가 발생하였으며, n이 10일때는 0.00000046475가 발생하였다.

두 번째로 특징점의 개수가 다른 여러장의 이미지들에서 OpenCV 소프트웨어의 삼각함수를 통한 descriptor와 식(1)과 식(2)를 사용하였을 때의 descriptor를 구하여 두 벡터의 각 bin들 간의 차를 구하였다. 그림 5는 특징점의 개수가 11363인 이미지상에서 descriptor 벡터의 각 bin의 오류를 보여주고 있다. n이 8일때는 bin들의 오차가 -1인 bin의 개수가 3개이며 +1인 bin의 개수가 7개이다. n이 10일때 descriptor의 오류가 없음을 알 수 있다. 다른 이미지 상에서도 n이 10일 때 descriptor의 오류가 발생 하지 않았다.

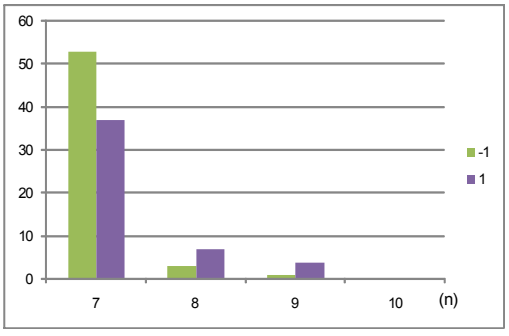

그림 5. Descriptor의 각 bin의 오류

위의 두가지 실험을 통해 본 논문에서는 descriptor의 오류가 발생하지 않고 사인/코사인 함수의 오차를 허용할 수 있는 범위로 n의 값을 10으로 정하였다. 식(3)은 n이 10일 때의 테일러 급수를 사용한 코사인 함수를 나타내며, 식(4)는 코사인 함수를 사용한 사인 함수의 변환을 나타낸다.

$$\cos(x) \approx 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} \tag{3}$$

$$\sin(x) = \cos\left(-\frac{\pi}{2} + x\right) \tag{4}$$

IV. 제안한 사인/코사인 프로세서의 구조

1. 사인/코사인 프로세서 구조

제안한 사인/코사인 프로세서는 함수의 대칭성을 사용하여 $0 \sim \pi/2$의 코사인 연산으로 사인 연산과 코사인 연산을 실행하도록 구현하였다.

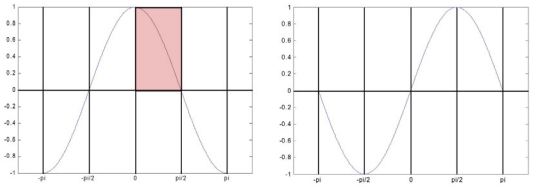

그림 6. 사인/ 코사인 함수의 출력

$-\pi \sim \pi$의 입력에 따른 각 함수의 출력이 [그림 6]과 같다. 색칠되어 있는 $0 \sim \pi/2$의 코사인 출력값으로 나머지 부분의 출력값을 구하기 위해서 식(5)와 식(6)과 같이 식을 변형하였다.

$$\cos(x) = \begin{cases} \cos(\pi + x) & (-\pi \le x < -\frac{\pi}{2}) \\ -\cos(-x) & (-\frac{\pi}{2} \le x < 0) \\ \cos(x) & (0 \le x < \frac{\pi}{2}) \\ -\cos(\pi - x) & (\frac{\pi}{2} \le x \le \pi) \end{cases} \tag{5}$$

$$\sin(x) = \begin{cases} -\cos(-\frac{\pi}{2} - x) & (-\pi \le x < -\frac{\pi}{2}) \\ -\cos(\frac{\pi}{2} + x) & (-\frac{\pi}{2} \le x < 0) \\ \cos(\frac{\pi}{2} - x) & (0 \le x < \frac{\pi}{2}) \\ \cos(-\frac{\pi}{2} + x) & (\frac{\pi}{2} \le x \le \pi) \end{cases} \tag{6}$$

식(5)와 식(6)을 코사인 함수의 입력, 코사인 프로세서, 그리고 출력값의 보정을 위한 모듈로 나누어서 구현하였다.

제안한 프로세서의 전체 시스템 구조가 [그림 7]과 같다. 첫 번째 모듈은 영역을 결정하는 모듈이다. 입력된 데이터가 $-\pi \sim \pi$사이의 네 부분 중 어느 부분에 속하는지 판단하여 식(5)와 식(6)과 같이 $0 \sim \pi/2$ 사이의 값으로 변환한다.

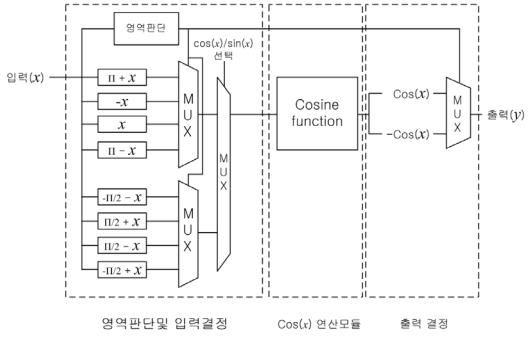

그림 7. 제안한 사인/코사인 프로세서

두 번째로 입력된 값을 식(3)의 테일러 급수를 통해 계산한다. 마지막으로 계산된 결과값을 첫 번째 모듈에서 결정한 영역에 따라 음수 또는 양수로 바꾸어 출력한다.

2. 영역판단 및 입력 결정

영역 판단 모듈은 cos(x) 연산모듈의 입력을 선택하는 모듈로써 $-\pi \sim \pi$사이의 값을 $\pi/2$간격으로 나누어진 네 부분 중 어디에 속하는지 결정한다. 결정한 후 식(5)와 식(6)과 같이 cos(x)의 입력을 결정하기 위해서 [그림 7]에서 연산된 8개의 입력중 하나를 선택한다. 결정된 입력은 $0 \sim \pi/2$사이의 값이 된다. 또한 사인과 코사인 연산을 선택하게 된다.

결정된 입력값이 양수이므로 제안한 cos(x) 연산 모듈은 음수에 대한 연산은 고려하지 않는다. 그러므로 본 논문에서는 복잡한 음수곱셈이 필요 없다.

3. cos(x) 연산 모듈

cos(x) 연산모듈은 영역판단에서 입력된 값을 식(3)에 의하여 연산하는 모듈이다. 각 항의 중복된 연산을 간소화하기 위해서 식(7)부터 식(11)과 같이 분해하여 연산하였다.

$$\frac{x^2}{2!} = \frac{x^2}{1 \cdot 2} = x^2 \cdot \frac{1}{2^1} \tag{7}$$

$$\frac{x^4}{4!} = \frac{x^{2+2}}{1 \cdot 2 \cdot 3 \cdot 4} = x^2 \cdot \frac{1}{3} \cdot x^2 \cdot \frac{1}{2^3} \tag{8}$$

$$\frac{x^6}{6!} = \frac{x^{4+2}}{1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6} = x^2 \cdot x^2 \cdot \frac{1}{3} \cdot x^2 \cdot \frac{1}{15} \cdot \frac{1}{2^4} \tag{9}$$

$$\frac{x^8}{8!} = \frac{x^{2+2+4}}{3 \cdot 7 \cdot 15 \cdot 2^7} = x^2 \cdot x^2 \cdot \frac{1}{3} \cdot x^2 \cdot \frac{1}{7} \cdot x^2 \cdot \frac{1}{15} \cdot \frac{1}{2^7} \tag{10}$$

$$\frac{x^{10}}{10!} = \frac{x^{8+2}}{3 \cdot 7 \cdot 15 \cdot 45 \cdot 2^8} = x^2 \cdot x^2 \cdot \frac{1}{3} \cdot x^2 \cdot \frac{1}{7} \cdot x^2 \cdot \frac{1}{15} \cdot x^2 \cdot \frac{1}{45} \cdot \frac{1}{2^8} \tag{11}$$

식(7)부터 식(11)을 구현한 하드웨어 구조가 [그림 8]과 같다. [그림 8]과 같이 2^n의 나눗셈은 배럴 쉬프트(barrel shift)상에서 오른쪽 쉬프트(shift)를 사용하여 구현하였는데, Rshift의 숫자는 쉬프트의 비트수를 나타낸다. 상수나눗셈은 $1/n$의 상수곱셈으로 바꾸어 연

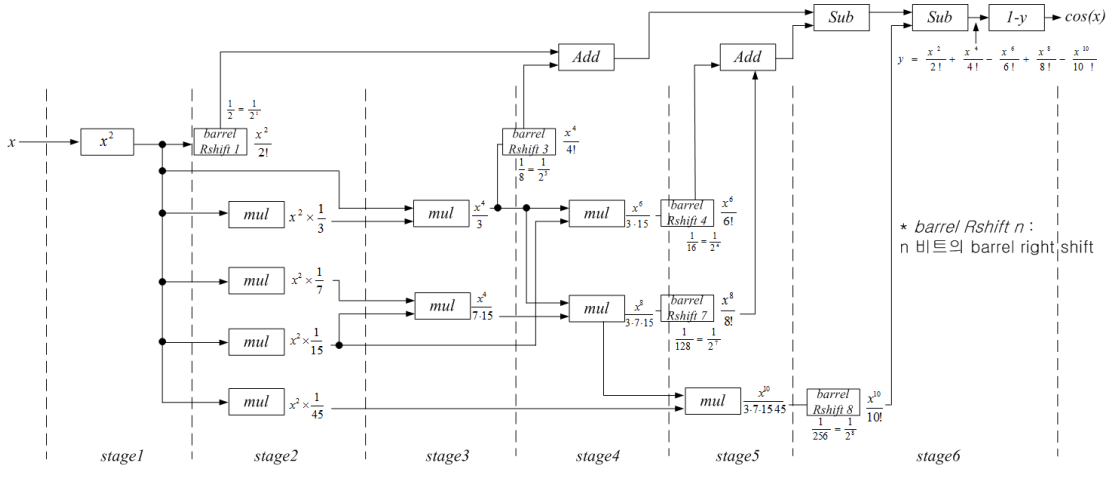

그림 8. 제안한 cos(x)의 하드웨어 구조

산함으로써 복잡한 나눗셈연산을 실행하지 않도록 구현하였다.

또한 프로세서의 연산 속도를 고려하여 6단계의 파이프라인 구조를 사용하였다. 스테이지1에서는 x^2의 연산만을 실행한다. 스테이지2에서 식(7)을 구하기 위한 1비트 배럴 쉬프트와 식(8)에서 식(11)까지의 연산을 위한 상수 곱셈을 구하게 된다. 스테이지3은 식(8)의 $x^4/3$ 부분과 식(10)의 $x^4/(7 \cdot 15)$ 부분을 연산하며 3비트 배럴 쉬프트를 통해 식(8)을 구한다. 스테이지4에서는 스테이지2와 스테이지3에서 구한 식(7)과 식(8)의 결과를 더함으로써 식(3)의 두 번째항과 세 번째항의 부분합(partial sum)을 계산하며, 식(9)와 식(10)을 계산한다. 스테이지5에서는 식(11)을 구하고, 식(9)와 식(10)을 더하여 식(3)의 네 번째항과 다섯 번째항의 부분합을 구하게 된다. 스테이지6에서 식(3)의 두 번째 항에서 여섯 번째 항까지의 부분합을 통합하고, 식(3)의 첫 번째 항인 1에서 식(7)에서 식(11)까지의 부분합([그림 8]에서 y)을 빼서 최종적으로 식(3)의 $\cos(x)$를 구하게 된다. 파이프라인의 스테이지는 곱셈의 최대시간을 고려하여 스테이지를 구분하였으며 배럴 쉬프트의 연산은 스테이지와 스테이지 사이에 실행함으로써 연산시간을 최대한 줄일 수 있다.

4. 출력 판단 모듈

출력 판단 모듈은 $0 \sim \pi/2$까지의 입력으로 계산된 사인/코사인 값을 올바른 값으로 변환하는 것으로써 식(5)와 식 (6)에 음수를 취하는 부분을 나타낸다. cos(x)는 $-\pi \sim -\pi/2$ 그리고 $\pi/2 \sim \pi$의 영역에서 출력에 음수를 sin(x)는 $-\pi \sim 0$까지의 영역에서 출력을 음수로 치환해 줌으로써 올바른 결과를 얻을 수 있다.

V. 구현 및 시뮬레이션

본 논문은 Verilog-HDL과 Open CV를 사용하여 모델링 하였으며, visual studio, modelsim, active-HDL, 그리고 matlab을 연동하여 시뮬레이션 및 검증을 수행하였다. 또한 Xilinx ISE 9.2i를 사용하여 합성을 하였고, Spartan 2E(XC2S200E-PQ208-6)를 사용하여 구현하였다.

구현한 하드웨어는 16비트의 입출력을 사용하며, 고정소수점으로 구현하였다. 고정소수점은 정수부 3비트와 소수부 13비트로 나누어 표현한다.

Lowe[4]의 SIFT를 소프트웨어로 구현한 사인/코사인의 결과와 제안한 프로세서를 사용한 사인/코사인의 결과가 [그림 10]과 같다. 왼쪽 그림(a)는 소프트웨어를 사용한 연산결과이고 중간 그림(b)는 제안한 프로세서를 사용한 연산결과이며 오른쪽 그림(c)는 두 개의 연산결과의 오차를 나타낸다.

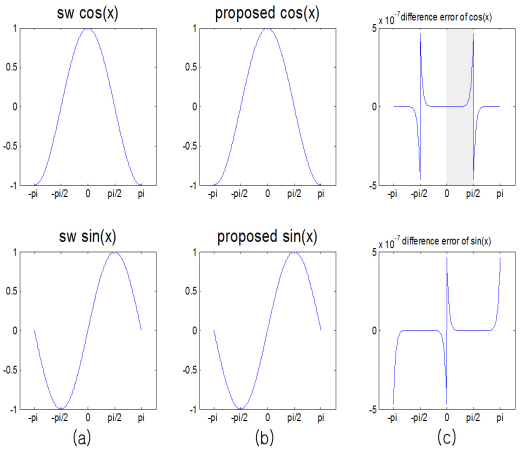

그림 10. 제안한 sin/cos 함수의 오차율, (a)소프트웨어 연산결과, (b)제안한 하드웨어의 연산결과, (c)두 연산결과의 오차(a)-(b)

[그림 10]에서 두 결과를 비교하였을 때, 제안한 프로세서는 식(3)의 테일러 시리즈에 의해 구현하였으므로 입력값이 커질수록 오차가 증가하여 입력의 최대값인 $\pi/2$에서 가장 큰 연산 오류 0.000000046475를 나타내었다. 또한 제안한 프로세서는 [그림 6]과 같이 cos(x)와 sin(x)를 각각 네 부분으로 나누어 함수의 대칭성을 사용하여 구현하였으므로 [그림 10]에서 cos(x)와 sin(x)의 오차가 $0 \sim \pi/2$사이의 cos(x)오차와 대칭을 이루고 있다. 이 결과를 사용하여 descriptor에 적용하였을 때 descriptor vector의 오류는 발생하지 않아 문제없이 동작함을 확인하였다.

표 2. Spartan XC2S200E DEVICE상에서 Sine/Cosine 프로세서의 Slice Delay 비교 (Word Length= 16bit)

평가항목 / 알고리즘	LUTs	Slices	max Frequency MHz	Slice Delay Product
Algorithm-I [12]	350	186	54.35	3.42
Algorithm-II[12]	378	203	60.80	3.34
scaling-Free CORDIC[11]	1658	945	52.54	17.99
Xilinx Core	1165	702	58.28	12.04
CORDIC[10]	359	191	59.766	3.19
Proposed	233	149	60.01	3.10

[표 2]는 본 논문에서 설계한 사인/코사인 하드웨어와 비교한 결과를 나타낸다. [12]의 Algorithm-I에서는 ROM과 여러개의 배럴 쉬프터(barrel shifter)를 사용하여 설계는 방법으로 2개의 가감산기와 73개의 플립플롭, 그리고 ROM을 통해 구현하였으며, Algorithm-II는 Algorithm-I을 개선하여 배럴 쉬프터를 제거한 방법으로 13개의 16비트 덧셈기, 7개의 18비트 뺄셈기와 215개의 플립플롭을 사용하여 구현한 것이다. [10]에서는 테일러급수를 변형된 CORDIC을 사용하여 FPGA상에 구현하였다. 본 논문은 Xilinx FPGA의 Spartan 2(XC2S200E)에 16bit의 입출력으로 배럴 쉬프터(barrel shifter)와 상수곱셈을 사용하여 구현한 구조로 10개의 곱셈기, 5개의 배럴 쉬프터, 5개의 덧셈기를 사용하였으며, 149 Slices에 233 LUTs(Look Up Table)가 소모되었다. 최대 주파수는 60.01MHz로 동작하였다. 이전 논문과 비교한 결과 [10]보다 크기는 36% 줄어들었으며 CORDIC 알고리즘 대신 테일러 급수 기법을 사용하여 동작속도가 3%정도 빨라짐을 알 수 있다. 또한 전체적으로 모든 비교 논문보다 성능이 향상됨을 알 수 있다.

구현한 사인/코사인 구조를 사용하여 descriptor를 구현하였을 때의 시뮬레이션 결과는 [표 3]과 같다. 특징점의 개수에 따라 descriptor의 연산시간과 소프트웨어와 제안한 하드웨어로 구현하였을 때의 연산시간을 보여주고 있다. 사인/코사인 연산시간은 각 특징점 한 개당 소프트웨어로 구현하였을 때에는 약 1ms가 걸렸으며, 제안한 하드웨어로 구현하였을 때에는 약0.03ms가 소모됨을 알 수 있다. 그러므로 특징점의 개수가 늘어날수록 제안한 하드웨어로 구현하였을 때 연산시간이 줄어들 수 있으며, 소프트웨어로 구현하였을 때 보다 전체적으로 약 40배 정도 빨라짐을 알 수 있다.

표 3. Descriptor의 연산비교(단위 초)

특징점의 수 (단위 개수)	348	1070	1978	3046	11363
기존 descriptor의 소프트웨어 연산시간	0.483	1.232	2.342	3.000	13.499
기존 소프트웨어 Sine/Cosine 연산시간	0.477	1.222	2.325	2.965	13.360
제안한 하드웨어 Sine/Cosine 연산시간	0.012	0.0301	0.058	0.073	0.323

VI. 결론

본 논문은 SIFT를 위한 descriptor의 사인/코사인 함수를 하드웨어로 구현하였다. SIFT 알고리즘에서 descriptor가 연산의 65%정도를 차지하고 또한 descriptor의 95%이상이 사인/코사인 연산을 하는데 사용되므로 하드웨어로 구현이 필요하다.

사인/코사인 알고리즘을 구현하기 위해서 테일러 시리즈를 사용하였으며, 또한 $0 \sim \pi/2$사이의 cos(x) 연산을 통해서 사인/코사인의 모든 연산을 실행할 수 있도록 구현하였다.

구현을 위해서 Verilog-HDL을 사용하였으며, 소프트웨어 시뮬레이션과 하드웨어 시뮬레이션을 위해서 Matlab과 Modelsim, 그리고 Visual studio 2010을 사용하여 시뮬레이션을 수행하였다. 또한 Xilinx Spartan 2E상에 구현하여 검증하였다. 시뮬레이션 결과 소프트웨어와 하드웨어의 최대 오차는 $-\pi \sim \pi$에서 0.000000046475이였으며, descriptor vector의 오류가 발생하지 않았다. 또한 $0 \sim \pi/2$사이의 cos(x) 연산을 통해서 사인/코사인의 모든 연산을 실행함으로써 음수곱셈기를 없애고 양수곱셈만으로 연산을 실행함으로써 Spartan 2E 상에 구현하였을 때 [10]과 비교하여 크기

는 36% 줄어들었으며, 속도는 3%정도 빠르게 연산되었다. 실제 descriptor의 소프트웨어 연산 시간과 비교하였을 때 전체적으로 40배 정도 빠른 연산의 성능을 얻을 수 있었다.

향후 연구 과제로는 본 논문에서 구현한 descriptor를 사용하여 SIFT의 전체적인 연산을 효율적으로 실행 할 수 있는 구조에 대한 연구가 필요하다.

참 고 문 헌

[1] C. Stauffer and W. E. L Grimson, "Learning patterns of activity using real-time tracking," IEEE Trans. Pattern Anal. Mach. Intell., Vol.22, No.8, pp.747-757, 2000(8).

[2] U. Ozguner, C. Stiller, and K. Redmill, "Systems for safety and autonomous behavior in cars: The DARPA grand challenge experience," Proc. IEEE, Vol.95, No.2, pp.397-412, 2007(2).

[3] C. Y. Lin, P. C. Jo, and C. K. Tseng, "Multi-functional intelligent robot DOC-2," in Proc. IEEE-RAS Int. Conf. Humanoid Robots, pp.530-535, 2006(12).

[4] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," International Journal of Computer Vision, Vol.60, No.2, 2004.

[5] H. D. Chati, F. Muhlbauer, T. Braun, C. Bobda, and K. Berns, "Hardware/software co-design of a key point detector on FPGA," in Proc. Annual IEEE Symp. Field-Programmable Custom Computing Machines, pp.355-356, 2007(4).

[6] 박찬일, 이수현, 정용진, "임베디드 환경에서 SIFT 알고리즘의 실시간 처리를 위한 특징점 검출기의 하드웨어 구현," 전자공학회, 제46권, SD편, 제3호, pp.86-95, 2009.

[7] V. Bonato, E. Marques, and G. A. Constantinides, "A parallel hardware architecture for scale and rotation invariant feature detection," IEEE Trans. Circuits System. Video Tech., Vol.18, No.12, pp.1703-1712, 2008(12).

[8] Y. M. Lin, C. H. Yeh, S. H. Yen, C. H. Ma, P. Y. Chen, and C. C. J. Kuo, "Efficient VLSI design for SIFT feature description," International Symposium on Next-Generation Electronics (ISNE), pp.48-51, 2010.

[9] GholamAli Yaghoubi Sheramin, Shahram Babaie and Reza Kazemi Asl, "Data-Oriented Architecture of Sine and Cosine Functions," International Conference on Advanced Computer Theory and Engineering(ICACTE), pp.32-34, 2010.

[10] Supriya Aggarwal and Kavita Khare, "Hardware Efficient Architecture for Generating Sine/Cosine Waves," International Conference on VLSI Design, pp.57-61, 2012.

[11] K. Maharatna, A. Troya, S. Banerjee, and E. Grass, "Virtually scaling free adaptive CORDIC rotator," IEEE Proc.-Comp. Dig. Tech., Vol.151, No.6, pp.448-456, 2004(11).

[12] Leena Vachhani, K. Sridharan and Pramod K. Meher, "Efficient CORDIC Algorithms and Architectures for Low Area and High Throughput Implementation," IEEE Transactions on Circuit and Systems - I: Express Briefs, Vol.56, No.1, pp.61-65, 2009(1).

저 자 소 개

김 영 진(Young-Jin Kim) 정회원

- 1999년 2월 : 상지대학교 전산학과(이학사)
- 2001년 8월 : 경희대학교 전자계산공학과(공학석사)
- 2001년 9월 ~ 현재 : 경희대학교 전자계산공학과 박사과정
- 2010년 6월 ~ 현재 : 대양전자통신(주)책임연구원

<관심분야> : 컴퓨터 구조, 병렬처리, SoC, FPGA, 영상처리, 음성처리

이 현 수(Hyon Soo Lee) 정회원

- 1979년 2월 : 경희대학교 전자공학과(공학사)
- 1982년 4월 : 일본 게이오대학원 전기공학과(공학석사)
- 1985년 4월 : 일본 게이오대학원 전기공학과(공학박사)
- 2005년 ~ 2008년 : 경희대학교 전자정보대학 학장 및 정보통신대학원 원장
- 1985년 ~ 현재 : 경희대학교 컴퓨터공학과 교수

<관심분야> : 컴퓨터구조 및 VLSI, 병렬처리, SoC, 패턴 인식, 신경망, 음성처리