2014년 11월 전자공학회 논문지 제 51권 제 11 호
Journal of The Institute of Electronics and Information Engineers Vol. 51, NO. 11, November 2014 http://dx.doi.org/10.5573/ieie.2014.51.11.073

논문 2014-51-11-10

LM(Levenberg-Marquardt) 알고리즘의 FPGA 구현

(FPGA Implementation of Levenverg-Marquardt Algorithm)

이 명 진*, 정 용 진**

(Myung-Jin Lee and Yong-Jin Jung©)

요 약

LM 알고리즘은 비선형 시스템의 least square problem을 풀기위해 사용되는 것으로, 다양한 분야에서 활용되고 있는 중요한 알고리즘이다. 하지만 응용 분야의 목적 함수가 복잡하고 고차원인 경우, 목적 함수의 연산 횟수가 많아지고, 내부에서 연산되는 행렬 및 벡터 연산에 시간이 많이 소요되어, 임베디드 환경에서의 실시간 동작을 위해서는 하드웨어 가속기 설계가 불가피하다. 본 논문에서는 LM 알고리즘을 하드웨어로 설계하였으며, 반복되는 목적 함수 연산을 파이프라인 처리 하고, 행렬 및 벡터 연산은 데이터 입력 주기를 줄여 속도를 향상시켰다. 설계한 LM 알고리즘의 하드웨어 성능을 측정하기 위해, 응용 분야로 3D reconstruction의 한 부분인 refining fundamental matrix(RFM)를 적용하였다. 실험 결과 소프트웨어와 비슷한 정확도를 가지면서, 최대 74.3배의 속도 향상을 볼 수 있었다.

Abstract

The LM algorithm is used in solving the least square problem in a non linear system, and is used in various fields. However, in cases the applied field's target functionis complicated and high-dimensional, it takes a lot of time solving the inner matrix and vector operations. In such cases, the LM algorithm is unsuitable in embedded environment and requires a hardware accelerator. In this paper, we implemented the LM algorithm in hardware. In the implementation, we used pipeline stages to divide the target function operation, and reduced the period of data input of the matrix and vector operations in order to accelerate the speed. To measure the performance of the implemented hardware, we applied the refining fundamental matrix(RFM), which is a part of 3D reconstruction application. As a result, the implemented system showed similar performance compared to software, and the execution speed increased in a product of 74.3.

Keywords : Levenberg-Marquardt, FPGA, Hardware, 3D reconstruction, Fundamental matrix

* 학생회원, ** 정회원, 광운대학교 전자통신공학과
(Department of electronics and communication engineering, Kwangwoon University)
© Corresponding Author(E-mail: yjjeong@kw.ac.kr)
※ This work was supported by the industrial Core Technology development Program(10049192, Development of a smart automotive ADAS SW-Soc for a self-driving car) funded By the Ministry of Trade, industry & Energy and supported by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the Human Resource Development Project for SoC support program(NIPA-2014-H0601-14-1001) supervised by the NIPA(National IT Industry Promotion Agency).
접수일자: 2014년06월12일, 수정일자: 2014년09월26일,
게재확정: 2014년10월27일

Ⅰ. 서 론

Least square problem을 푼다는 것은 방정식 해의 근사값을 찾는 것이며, 방정식은 선형 시스템과 비선형 시스템으로 나눌 수 있다. 선형 시스템은 푸는 방법이 정형화 되어 있지만[1], 비선형 시스템을 풀기 위한 방법은 Gauss-Newton 방법[2], LM(Levenberg-Marquardt) 알고리즘[3~4], Powell's Dog Leg 방법[5] 등 다양한 방법이 존재한다. 그 중 LM 알고리즘은 curve fitting, neural network, camera calibration, 3D

reconstruction[6] 등 다양한 분야에서 활용되고 있는 중요한 알고리즘이다. 하지만 목적 함수의 차원이 높고, 연산이 복잡한 응용 분야의 경우, 연산 시간이 오래 걸려 임베디드 환경에 부적합하기 때문에, 하드웨어 가속을 통해 속도를 향상시켜야 한다.

본 논문에서는 LM 알고리즘의 하드웨어 설계에 대해 연구하였으며, 하드웨어 검증을 위해 3D reconstruction의 한 부분인 refining fundamental matrix(RFM)를 응용 분야로 적용하였다. RFM은 LM 알고리즘의 중요한 응용 분야중 하나로, 내부 연산이 매우 복잡하고, 입력에 따라 내부 행렬 및 벡터의 크기가 매우 커지기 때문에 임베디드 환경에 부적합하다는 한계를 가지고 있어, 하드웨어 가속이 적절한 응용 분야이다.

각 장의 내용은 다음과 같다. Ⅱ장에서는 LM 알고리즘에 대해 간단히 서술하고, Ⅲ장에서는 제안한 LM 알고리즘의 하드웨어에 대해 서술하였다. Ⅳ장에서는 응용 분야인 RFM, Ⅴ장은 실험 결과 및 분석, 끝으로 Ⅵ장에서 결론을 서술한다.

Ⅱ. LM 알고리즘

비선형 시스템의 least square problem은 미분을 이용하여 반복적으로 해의 근사값을 구하며, Newton-Rapson 방법과 흐름이 유사하다.

그림 1은 Newton-Rapson 방법으로, k+1 단계를 반복적으로 구하면서 해의 근사값을 구하는 방법을 나타낸다. 이와 유사한 방식으로, 비선형 시스템의 least square problem은 $F: R^n \rightarrow R^m$인 함수가 있을 때, $\mathbf{x} \in R^n$에 대한 $\| F(\mathbf{x}) \|$의 값을 최소화 시키는 $\mathbf{x}$를 찾는 것을 말하며, k+1 단계는 $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{h}$로, 식(1)을 기반으로 $\mathbf{h}$를 구한다.

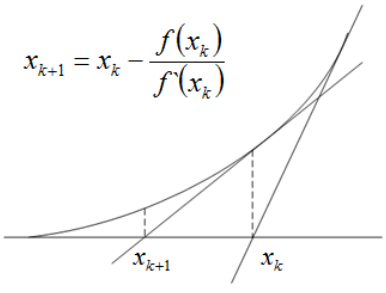


그림 1. Newton-Rapson 방법
Fig. 1. Newton-Rapson method.

$$F(\mathbf{x}_k + \mathbf{h}) \approx F(\mathbf{x}_k) + F'(\mathbf{x}_k)\mathbf{h} = \mathbf{f}_k + \mathbf{J}_k\mathbf{h} \quad (1)$$

$$\mathbf{J}_k^{\mathrm{T}}\mathbf{J}_k\mathbf{h} = -\mathbf{J}_k^{\mathrm{T}}\mathbf{f}_k \quad (2)$$

$$(\mathbf{J}_k^{\mathrm{T}}\mathbf{J}_k + \mu\mathbf{I})\mathbf{h} = -\mathbf{J}_k^{\mathrm{T}}\mathbf{f}_k \quad (3)$$

$$j_{ij} = \frac{\partial f_i}{\partial x_j}(\mathbf{x}),\ (x_j \in \mathbf{x}, f_i \in F(\mathbf{x}), j_{ij} \in \mathbf{J}) \quad (4)$$

$F(\mathbf{x}_k + \mathbf{h})$는 테일러 전개를 통해 식(1)과 같이 근사화할 수 있으며, $\mathbf{J}$는 $F(\mathbf{x})$의 도함수로 식(4)와 같이 구한다. 식(1)을 선형 시스템의 least square problem을 풀기 위한 형태로 나타낸 것이 식(2)이며, 이를 반복적으로 풀어 해의 근사값을 구하는 방법이 Gauss-Newton 방법이다. LM 알고리즘은 식(2)를 기반으로 해에 더 빠르게 수렴할 수 있도록 댐핑 파라미터(damping parameter) μ를 조절하며 식(3)과 같은 방법으로 $\mathbf{h}$를 구한다. 댐핑 파라미터는 $\| F(\mathbf{x}) \|$가 0에 얼마나 근접했는지, 올바른 방향으로 진행되고 있는지 판단하는 척도이다. 아래 설명은 LM 알고리즘의 흐름으로 stage는 LM 알고리즘의 세부 단계를 의미하고, step은 k 번째 반복을 의미한다.

LM 알고리즘의 주요 변수에 대한 설명은 표 1과 같으며, $F(\mathbf{x})$는 응용 분야에 따라 달라진다. LM 알고리즘의 흐름[8]은 그림 2와 같이 8 stage에 걸쳐 수행되며, 표 2는 각 stage에서 어떤 연산이 수행되는지 나타낸다. Stage 1은 초기 step인지 판단하고 함수를 계산하는 단계로 현재 step 또는 다음 step의 함수를 계산한다. Stage 2는 현재 step이 유효한지 판단하는 stage로, ρ 값을 구하여 0보다 큰 경우 유효하다고 판단한다. Stage 3에서는 현재 step이 유효할 경우, $\mathbf{x}$와 $\mathbf{f}$를 업데

표 1. LM 알고리즘의 변수 설명
Table 1. Description of LM algorithm variables.

- $\mathbf{x} \in R^n$(현재 step), $\mathbf{x}_{new} \in R^n$(다음 step)
- $F(\mathbf{x}) = \mathbf{f} \in R^m$(현재 step), $F(\mathbf{x}_{new}) = \mathbf{f}_{new} \in R^m$(다음 step)
- $\mathbf{J}$: $m \times n$ Jacobi 행렬
- $\epsilon = 10^{-8}, \tau = 10^{-3}$
- $\rho = \dfrac{\| \mathbf{f} \|^2 - \| \mathbf{f}_{new} \|^2}{\mathbf{h}(\mu\mathbf{h} - \mathbf{g})}$
- μ : damping parameter(LM parameter)
- $\max iter = 200$

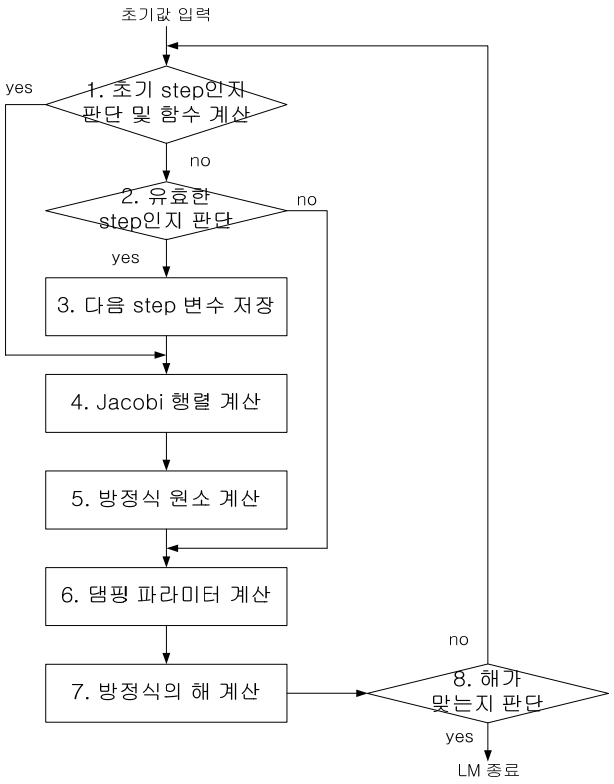


그림 2. LM 알고리즘의 흐름
Fig. 2. LM algorithm flow.

표 2. LM 알고리즘의 각 stage 별 연산
Table 2. Operations in each stage of LM algorithm.

단계(stage)		연산
1	초기 step인지 판단 및 함수 계산	$\mathrm{if}(iter==0)$ $\mathbf{f}=F(\mathbf{x})$ else $\mathbf{x}_{new}=\mathbf{x}+\mathbf{h}$ $\mathbf{f}_{new}=F(\mathbf{x}_{new})$
2	유효한 step인지 판단	$\rho=\frac{\|\mathbf{f}\|^2-\|\mathbf{f}_{new}\|^2}{\mathbf{h}(\mu\mathbf{h}-\mathbf{g})}$
3	다음 step 변수 저장	$\mathbf{x}=\mathbf{x}_{new},\ \mathbf{f}=\mathbf{f}_{new}$
4	Jacobi 행렬 계산	$\mathbf{J}$(식(4))
5	방정식 원소 계산	$\mathbf{A}=\mathbf{J}^T\mathbf{J},\ \mathbf{g}=\mathbf{J}^T\mathbf{f}$
6	댐핑 파라미터 계산	$\mathrm{if}(iter==0)$ $\mu=\tau\times\max\{\mathrm{diag}(\mathbf{A})\};$ else $\mathrm{if}(\rho>0)$ $\mu=\mu\times\max\{1/3,1-(2\rho-1)^3\};$ $\nu=2;$ else $\mu=\mu\times\nu;$ $\nu=2\nu;$
7	방정식의 해 계산	row reduction[7]
8	해가 맞는지 판단	$\|\mathbf{h}\|\le\epsilon(\|\mathbf{x}\|+\epsilon)$ or $iter==200$

이트 시킨다. Stage 4는 Jacobi 행렬을 계산하고, stage 5에서는 방정식을 풀기 위하여 식(3)의 각 원소들을 계산한다. Stage 6에서는 댐핑 파라미터를 계산하고, stage 7에서 식(3)을 풀어 $\mathbf{h}$를 구한다. 마지막으로 stage 8은 LM 알고리즘의 종료 조건을 판단하며 두 가지 조건 중 하나를 충족하면, LM 알고리즘을 종료시킨다. 첫 번째 조건은, $\|\mathbf{h}\|\le\epsilon(\|\mathbf{x}\|+\epsilon)$를 만족하여 해가 구해졌다고 판단하는 경우이고, 두 번째는, 최대 반복 횟수에 도달할 경우이다. 최대 반복 횟수가 정해져 있는 이유는 무한 반복을 방지하기 위함이다.

Ⅲ. 하드웨어

1. 하드웨어 설계

LM 알고리즘을 하드웨어로 설계하기 위해서는 고정 소수점 방식으로 연산할 것인지 부동 소수점 방식으로 연산할 것인지 판단해야 하며, 연산이 어떤 부분에 집중되는지 파악하여 연산 효율을 높여야 한다. LM 알고리즘은 사용되는 수의 범위가 넓고 랜덤하며, 반복 연산을 통해 해를 구한다는 특징이 있다. 이 같은 경우 고정 소수점으로 연산을 하게 되면 오차의 누적으로 인해 정확도가 떨어지게 되며, 높은 정확도를 요구하는 분야에서는 활용할 수 없게 된다. 이와 같은 이유로 연산의 대부분이 IEEE 754 floating-point format으로 수행된다. 연산을 위해서 Xilinx ISE에서 제공하는 FPU 5.0을 사용하였다. FPU는 매 클럭마다 데이터를 입력할 수 있으나, 출력에 2 클럭 이상 소요되기 때문에, 연산이 매 클럭마다 이루어질 수 없는 부분이 존재하며, 속도 향상을 위해 이를 고려하여 하드웨어를 설계해야 한다.

LM 알고리즘(그림 2)의 각 stage에서 연산 시간이 오래 걸리는 부분은 크게 두 부분으로 나눌 수 있으며,

표 3. Stage 4의 pseudo 코드
Table 3. Pseudo code of stage 4.

```
for i=0: n-1 {
  deltax = memory_copy(x);
  delta = eps*x[i];
  deltax[i] = x[i]+delta;
  wa = compute_function(deltax);
  for j=0: m-1 {
    Jacobi[i*m+j] = (wa[j]-f[j])/ delta;
  }
}
```

stage 4 Jacobi 행렬 계산과 stage 5 방정식 원소 계산이다. Jacobi 행렬의 크기는 $m \times n$으로, 식(4)와 같이 $F(\mathbf{x})$를 $\mathbf{x}$의 각 원소에 대해 편미분한 행렬이다. Pseudo 코드는 표 3과 같으며, 이 부분에서 연산 시간이 오래 걸리는 이유는 입력을 변화시키면서 n번 함수 연산을 수행하기 때문이다.

반복적인 함수 연산은 파이프라인 처리를 통해 속도를 향상시킬 수 있다. 파이프라인 처리를 위해서는 함수를 연산할 때, 단계별로 나누어 수행시킬 수 있도록 연산기를 분리해 설계해야하고, Jacobi 행렬의 원소를 계산할 때는 함수의 결과값을 저장하기 위한 메모리를 추가로 두어 더블 버퍼링 방법을 활용해야 한다. 함수

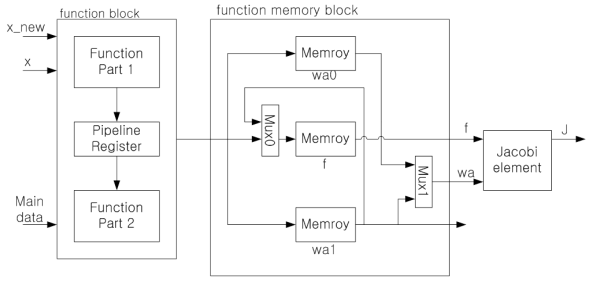


그림 3. function block과 function memory block의 구조
Fig. 3. Structure of function block and function memory block.

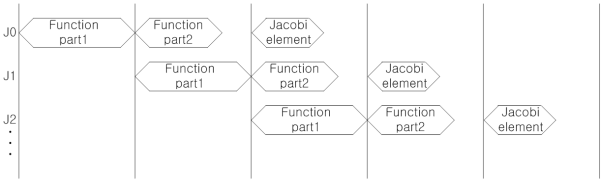


그림 4. Jacobi 행렬을 위한 파이프라인 처리
Fig. 4. Pipeline processing for obtaining the Jacobian matrix.

표 4. Stage 5의 pseudo 코드
Table 4. Pseudo code of stage 5.

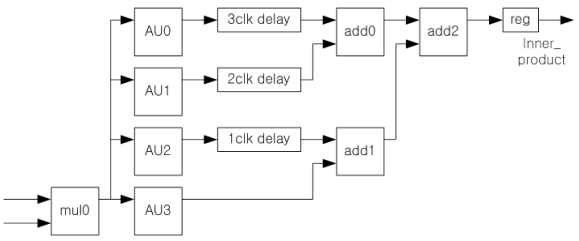
```
for row=0: n-1 {
  for col=row: n-1 {
    A[row*n+col] = inner_product(&J[row*m],
                   &J[col*m], m);
      if(row != col)
        A[col*n+row] = A[row*n+col];
  }
}

for row=0: n-1{
  g[row] = inner product(&J[row*m], f);
}
```

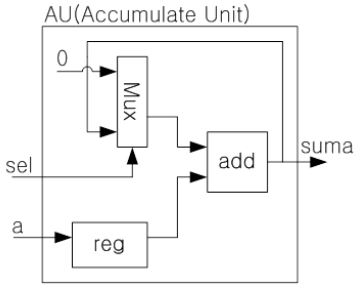
연산을 두 개의 연산기로 나누어 설계한 구조와 메모리 블록의 구조는 그림 3과 같고, 그림 4와 같이 3단계의 파이프라인 연산 타이밍을 가진다.

Stage 5 방정식 원소 계산은 $\mathbf{J}^T\mathbf{J}$와 $\mathbf{J}^T\mathbf{f}$를 계산하며, $\mathbf{J}^T\mathbf{J}$를 구하는 과정에서 $m \times 1$벡터 내적을 ${}_{n+1}C_2$번 수행해야 하고, $\mathbf{J}^T\mathbf{f}$를 구하는 과정에서 $m \times 1$벡터 내적을 n번 수행한다. 내적의 수행 횟수가 많기 때문에 연산 시간이 오래 걸리며, pseudo 코드는 표 4와 같다.

Stage 5의 주된 연산은 벡터의 내적이며, 내적은 곱셈의 결과가 누적되는 형태이기 때문에 덧셈기의 latency에 따라 속도가 결정된다. Latency는 입력 데이터에 대한 결과가 출력되는데 소요되는 클럭을 의미한다. 200MHz 동작시, 덧셈기의 latency는 4이며, 데이터 입력 주기가 4 클럭으로 제한되기 때문에 연산 시간이 오래 걸리게 된다. 연산 속도를 향상시키기 위해 그림 5와 같이 누적 연산을 병렬 처리를 하면, 그림 6과 같이 매 클럭마다 데이터를 입력할 수 있어 처리 속도를 향



(a) Structure of inner product unit



(b) Accumulate unit

그림 5. 내적 연산기와 누적 연산기의 구조
Fig. 5. Structure of inner product and accumulate unit.

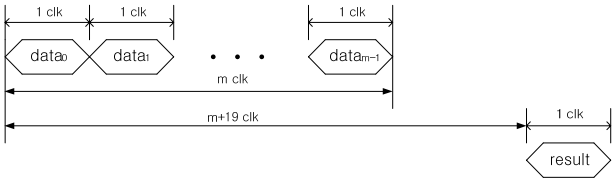


그림 6. 내적 연산 타이밍
Fig. 6. Timing of inner product.

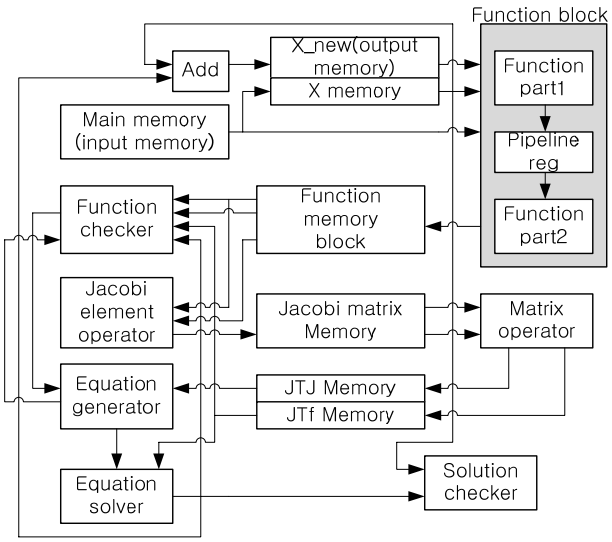


그림 7. LM 알고리즘의 하드웨어 구조
Fig. 7. Hardware architecture of LM algorithm.

상시킬 수 있다.

LM 알고리즘의 흐름(그림 2, 표 2)을 보면 주요 변수들의 연산은 종속성이 존재하며, 행렬 및 벡터 형태로 크기가 큼을 알 수 있다. 또한 경우에 따라서 주요 변수들을 다시 활용되는 경우도 있으므로, 데이터들은 메모리를 통해 이동해야하며, 그림 7과 같은 하드웨어 구조를 가진다.

2. LM 하드웨어 각 연산기별 동작

그림 7에서 Function block은 응용 분야에 맞도록 설계해야 하며, 각 연산기 및 메모리의 역할은 아래와 같다. Main memory는 데이터가 입력되는 메모리로 $\mathbf{x}$의 초기값, 함수의 입력 차원 수(n), 출력 차원수(m)와 함수 연산에 활용될 데이터가 저장된다. Function block은 Jacobi 연산을 위해 두 연산기로 나누었으며, 두 연산기 사이에는 파이프라인 레지스터가 존재한다. Function memory block은 3개의 메모리로 이루어져 있으며, 메모리 중 2개는 Jacobi 행렬을 구하기 위한 더블 버퍼링을 위해 활용되며, 나머지 1개는 특정 조건이 발생하기 전까지 데이터를 저장하고 있는 메모리이다. Function checker는 표 1의 ρ를 구하는 연산기로 결과값에 따라 Jacobi 행렬을 구하고 Matrix operator를 동작시킬 것인지, 댐핑 파라미터만 계산할 것인지 결정된다. Matrix operator은 벡터 내적을 위한 연산기로 $\mathbf{J^TJ}$와 $\mathbf{J^Tf}$를 연산하기 위한 연산기이다. Equation generator는 댐핑 파라미터(μ)를 구하고 $\mathbf{J^TJ}+\mu\mathbf{I}$를 구하는 연산기이다. Equation solver는 식(3)을 풀기위한 연산기이다. Solution checker는 LM 알고리즘의 해가 구해졌는지 판단한다. Equation solver의 결과인 $\mathbf{h}$는 $\mathbf{x}$와 더해져 X_new memory에 저장된다.

Ⅳ. LM 하드웨어의 응용

1. Fundamental 행렬

3D reconstruction은 다수의 영상을 입력받아 3차원 영상으로 재생성하는 것으로 참고문헌 [6]에 자세히 서술되어있으며, fundamental 행렬은 3차원 좌표를 추출하기 위해 필요한 행렬이다.

그림 8에서 $\mathbf{M}_i$은 실제 3차원에서의 좌표이며, $\mathbf{m}_i$과 $\mathbf{m'}_i$은 각각 $\mathbf{M}_i$이 image0와 image1에 맺힌 상이다. 이 때 $\mathbf{M}_i$, $\mathbf{m}_i$, $\mathbf{m'}_i$ 세 점으로 이루어진 평면을 epipolar plane이라하고, epipolar plane과 각 영상이 만나는 직선을 epiplolar line($\mathbf{e}_i$, $\mathbf{e'}_i$)이라고 하며, 식 (5), (6)을 만족시키는 rank 2 행렬 $\mathbf{X}$를 fundamental 행렬이라고 한다.

$$\mathbf{m'}_i\mathbf{X}=\mathbf{e}_i,\ \mathbf{X}\mathbf{m}_i=\mathbf{e'}_i(\mathbf{X}:3\times3\,\mathrm{rank}2) \tag{5}$$

$$\mathbf{m'}_i\mathbf{X}\mathbf{m}_i=0 \tag{6}$$

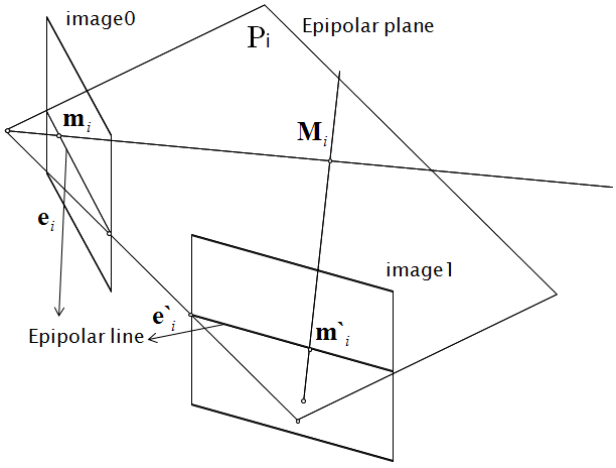


그림 8. 두 영상의 매칭점과 epipolar plane과의 관계
Fig. 8. The relationship between matched point of two images and epipolar plane.

2. RFM(Refining fundamental matrix)

RFM은 fundamental 행렬을 정확하게 재조정하는 부분으로, 정확한 3차원 좌표를 생성하기 위해 반드시 수행해야하는 부분이다. 그림 8의 $\mathbf{m}_i$과 $\mathbf{m'}_i$은 두 영상의

매칭점이며, 각 영상에서 SIFT[9]를 활용하여 특징점을 추출한 뒤, ANN매칭[10]을 통해 추출한 좌표이다. 이 매칭점을 기반으로 $\mathbf{X}$를 구하며, 모든 매칭점의 쌍에 대해 식 (5), (6)을 만족해야한다. 하지만 잘못된 매칭점 쌍과 컴퓨터 계산으로 인한 오차가 존재하기 때문에, 가장 정확한 $\mathbf{X}$를 찾아야하며, LM 알고리즘을 통해 해결할 수 있다.

3. 목적 함수

Fundamental 행렬의 오차는 $F(\mathbf{x})$로 나타낼 수 있으며, 이것이 LM 알고리즘의 목적 함수가 된다. $\mathbf{X}$는 $\mathbf{x}$의 각 원소로 이루어져 있는 행렬로 식(7)과 같다.

$$\mathbf{x}=\begin{bmatrix} x_1 \\ \bullet \\ \bullet \\ \bullet \\ x_9 \end{bmatrix} \Rightarrow \mathbf{X}=\begin{bmatrix} x_1 x_2 x_3 \\ x_4 x_5 x_6 \\ x_7 x_8 x_9 \end{bmatrix} \tag{7}$$

Fundamental 행렬의 오차는 correction, residual 두 단계에 걸쳐 계산할 수 있다. Correction 연산은 fundamental 행렬의 특성인 rank 2로 변환시키는 단계로, SVD[11]를 활용하여 식(8)과 같이 $\mathbf{X}$를 분해한 후, 가장 작은 singular value를 0으로 고정하여 식(9)와 같이 $\mathbf{X}'$을 구한다.

$$\mathbf{X}=\mathbf{U}\begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix}\mathbf{V}^{\mathrm{T}} \ (\sigma_1 > \sigma_2 > \sigma_3) \tag{8}$$

$$\mathbf{U}\begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & 0 \end{bmatrix}\mathbf{V}^{\mathrm{T}}=\mathbf{X}' \ (\sigma_1 > \sigma_2 > 0) \tag{9}$$

Residual 연산은 fundamental 행렬의 오차를 구하는 단계로 두 영상의 epipolar line과 매칭점이 얼마나 떨어져 있는지에 대한 척도를 의미한다. 매칭점의 개수가 m일 때, i번째 매칭점에 대한 residual은 식(10), (11) 연산을 통해 구할 수 있다. 이를 모든 매칭점에 대해 구하면 식(12)와 같은 형태인 목적 함수가 되며, 그림 2와 같은 과정을 거쳐 $\| F(\mathbf{x}) \|$를 최소화시키는 $\mathbf{x}$를 찾는다.

$$\begin{matrix} \mathbf{m}'_i\mathbf{X}'=\mathbf{e}_i \\ \mathbf{X}'\mathbf{m}_i=\mathbf{e}'_i \end{matrix}, \left(\mathbf{e}_i=\begin{bmatrix} e_{ix} \\ e_{iy} \\ e_{iz} \end{bmatrix}, \mathbf{e}'_i=\begin{bmatrix} e'_{ix} \\ e'_{iy} \\ e'_{iz} \end{bmatrix} \right) \tag{10}$$

$$f_i=\sqrt{\left(\frac{1}{e_{ix}^2+e_{iy}^2}+\frac{1}{e'^2_{ix}+e'^2_{iy}}\right)\times(\mathbf{m}'_i\mathbf{X}'\mathbf{m}_i)^2} \tag{11}$$

$$F(\mathbf{x})=\mathbf{f}=\begin{bmatrix} f_1 \\ \bullet \\ f_i \\ \bullet \\ f_m \end{bmatrix} \tag{12}$$

4. 목적 함수 하드웨어

목적 함수는 Correction operator와 Residual operator로 나누어 설계할 수 있으며, 그림 9와 같다. 두 연산기 사이에는 파이프라인 레지스터가 있으며, 이는 Jacobi 행렬을 계산할 때, 파이프라인 처리를 통해 속도를 향상시키기 위함이다.

Correction operator는 식(8), (9)를 계산하기 위한 연산기로, 그림 10과 같은 연산 타이밍을 가진다. 여기서 Main block은 회전을 수행하는 부분으로, sin·cos값을 구하는 과정에서 나눗셈과 루트 연산이 포함되어 있고, 내부에서 반복이 되기 때문에 시간이 오래 걸린다. Correction의 연산은 평균 2975 clock이 소요되며, 소요되는 시간이 일정하다.

Residual operator는 식(10), (11)을 계산하기 위한 연

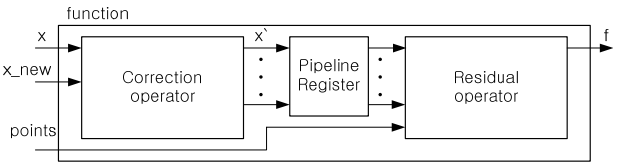


그림 9. RFM 목적 함수 연산기
Fig. 9. Object function operator of RFM.

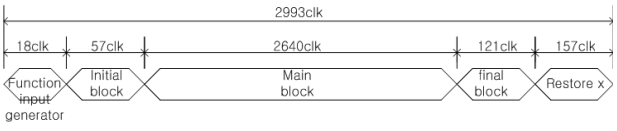


그림 10. Correction operator의 연산 타이밍
Fig. 10. Operation timing of correction operator.

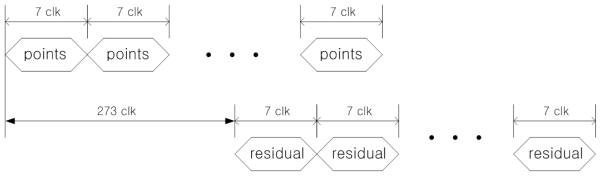


그림 11. Residual의 연산 타이밍
Fig. 11. Operation timing of residual operator.

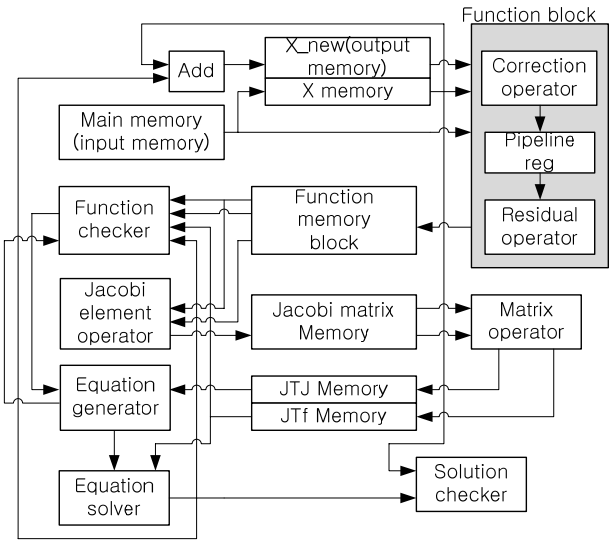


그림 12. RFM을 적용한 LM 하드웨어
Fig. 12. LM hardware applied RFM.

산기로, 그림 11과 같이 파이프라인 처리된 연산 타이밍을 가진다. 입력 데이터는 7 clock 주기로 입력되며, 입력에 대한 결과값은 273 clock 후 부터 7 clock 단위로 출력된다. Residual의 수행 시간은 매칭점 개수에 의해 결정된다.

설계한 RFM의 목적 함수를 LM 알고리즘의 하드웨어에(그림 7) 적용하면, 그림 12와 같은 구조가 되며, 각 연산기에 대한 설명은 3.2에서 설명하였다.

V. 실험 결과 및 분석

1. 실험 환경 및 내용

RFM을 적용한 LM 알고리즘의 성능을 측정 하였으며, 수행 시간 및 정확도를 측정하였다. 수행 시간 비교를 위해 소프트웨어는 Exynos 4412(ARM Cortex-A9 1.4 GHz)를 사용하였고, 하드웨어는 Xilinx 사의 KC 705(Kintex 7XC7K325T FPGA)에서 200MHz로 동작할 수 있도록 설계하였다. RFM을 적용한 LM 알고리즘의 입력은 fundamental 행렬의 초기값, 매칭점 쌍의 수, 매칭점의 좌표이고, 출력은 보정된 fundamental 행렬이다.

2. FPGA 자원 활용율

KC 705에는 Kintex-7 XC7K325T FPGA가 내장되어 있으며, 설계된 하드웨어의 자원에 대한 활용도는 표 5와 같다. 본 논문의 LM 하드웨어는 부동 소수점 연산으로 인해 많은 LUT와 Register를 사용함을 알 수 있다. KC 705의 BRAM은 1개당 36Kbit로, 입력 메모리에 2개의 BRAM을 사용하여 최대 576개의 데이터가 입력될 수 있다. 또한 Function memory block에서 3개, Jacobi matrix memory에서 8개를 사용하여 총 13개의 BRAM을 사용한다. 나머지 메모리는 크기가 작아 LUT를 활용하여 메모리를 생성하였다.

표 5. 디바이스 활용도
Table 5. Device utilization.

구분	LUT	Register	DSP	BRAM
사용량/ 총 자원 (%)	77891/ 203800 (38%)	67390/ 407600 (16%)	519/ 840 (61%)	13/ 445 (2%)

3. 매칭점 수에 대한 연산 시간

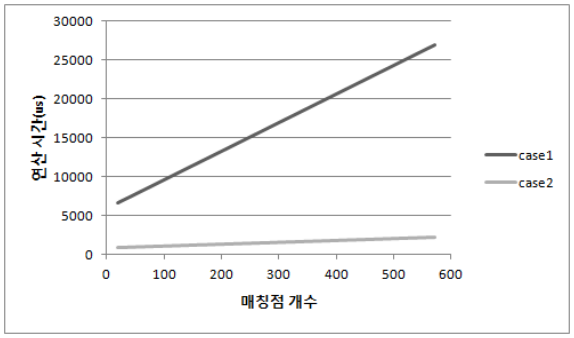
RFM을 적용한 LM 알고리즘의 수행 시간 입력되는 매칭점의 수에 따라 연산 시간이 가변적이다. 매칭점의 수가 m개라고 한다면, $\mathbf{f}$의 차원수는 m이고, $\mathbf{J}$행렬의 크기는 m×8이 된다. 따라서 목적 함수 $\mathbf{f}$, Jacobi행렬 $\mathbf{J}$, 방정식의 원소 $\mathbf{J}^T\mathbf{J}$와 $\mathbf{J}^T\mathbf{f}$ 계산에 소요되는 시간이 변하게 되어, 매칭점 수에 따른 연산 시간을 분석해야 한다. 또한 LM 알고리즘은 매 반복마다 데이터의 흐름이 달라지기 때문에, 흐름에 따라 분석을 진행해야하며, 그림 2에 근거하여 표 6과 같이 두 방향으로 나눌 수 있다.

표 6에서 case 1은 $\mathbf{J}$ 계산 후 $\mathbf{J}^T\mathbf{J}$, $\mathbf{J}^T\mathbf{f}$를 계산하는 경우이고, 이를 계산하지 않는 경우가 case 2이며, 매 반복에서 두 case 중 하나가 수행된다. 그림 13은 매칭점 수에 대한 각 case의 연산 시간을 나타낸다.

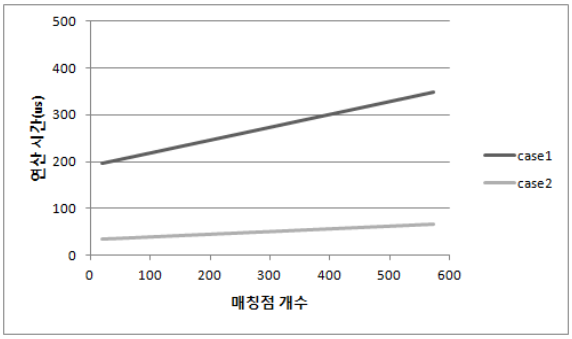
그림 13에서 하드웨어가 소프트웨어에 비해 case 1일 경우, 33.5~76.9배 속도가 빠르고, case 2일 경우 23.1~33.9배 속도가 상승하였다. Case 1이 case 2에 비해 속도 상승률이 높은 이유는 Jacobi 행렬 계산 시 파이프라인 처리하고, $\mathbf{J}^T\mathbf{J}$, $\mathbf{J}^T\mathbf{f}$ 연산을 병렬 처리로 하기 때문이며, 입력 데이터의 수가 많아질수록 Jacobi 행렬과 $\mathbf{J}^T\mathbf{J}$, $\mathbf{J}^T\mathbf{f}$ 계산 효율이 높아지게 되어 속도 상승률이 높아지게 된다.

표 6. Case 별 LM 알고리즘 흐름
Table 6. LM algorithm flow in each case.

구 분	stage 흐름
case 1	1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9
case 2	1 → 2 → 6 → 7 → 8 → 9



(a) 매칭점 수에 대한 case 1, 2의 연산 시간(Exynos 4412)



(b) 매칭점 수에 대한 case 1, 2의 연산 시간(KC 705)

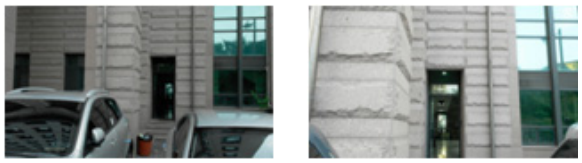
그림 13. Exynos 4412와 KC 705에서의 연산 시간
Fig. 13. Operation time on Exynos 4412 and KC 705.

4. 전체 동작 시간 및 정확도

5.3절은 매칭점 수에 대한 반복 1회의 수행 시간에 대해 기술하였으며, 입력되는 매칭점 수에 따라 연산 시간이 증가함을 알 수 있다. 하지만 입력되는 매칭점의 수가 같더라도 매칭점 좌표와 fundamental 행렬의 초기값이 다른 경우, 반복 횟수가 다를 수 있으므로, 각 case가 반복 되는 횟수 및 전체 반복 횟수와 연산 시간을 측정해야한다. 전체 연산 시간 측정을 위해, 56개의 영상(1280720)을 촬영하였으며, 각 영상의 쌍에서 매칭점과 fundamental 행렬을 추출하여 데이터 베이스를 구축하였다. 데이터 베이스의 매칭점 평균 개수는 96개이고, 최소 20개, 최대 573개이며, 표 7과 같이 입력 매칭점 수가 최소일 경우를 EX 1, 평균일 경우를 EX 2, 최대일 경우를 EX 3으로 나누어 전체 연산 시간에 대한 결과를 서술하였다.

그림 14는 실험을 위해 선별한 영상으로 (a), (b), (c)

(a) EX 1의 입력 영상



(b) EX 2의 입력 영상

(c) EX 3의 입력 영상

그림 14. 실험 항목의 입력 영상
Fig. 14. Input images in each experiment.

표 7. 각 실험 항목의 데이터 개수
Table 7. Number of data in each experiment.

구분	EX 1	EX 2	EX 3
입력 데이터 수	20	96	573

표 8. 전체 연산 시간 비교
Table 8. Comparison total operation time.

구분		case1 반복	case2 반복	전체 반복	Time
단위		횟수	횟수	횟수	ms
EX 1	Exynos 4412	22	12	34	155.526
	KC 705	22	12	34	4.768
EX 2	Exynos 4412	20	13	33	201.346
	KC 705	20	14	34	4.919
EX 3	Exynos 4412	17	12	29	483.955
	KC 705	16	14	30	6.860

에서 추출한 매칭점 수는 각각 20개, 96개, 573개이다. 각 영상에서 fundamental 행렬과 매칭점을 추출하여 소프트웨어와 하드웨어에서의 LM 알고리즘의 속도를 측정하였다. 표 8은 그림 14의 각 실험 항목에 대한 전체 연산 시간을 측정한 결과로, case1과 case2가 수행되는 횟수와 전체 연산 시간을 나타낸다. 실험의 각 항목에서는 약 32.6~74.3배의 속도 향상을 보이고 있다. 표 8을 보면, 소프트웨어와 하드웨어의 반복 횟수가 다른 항목이 존재한다. 반복 횟수가 다른 이유는 floating-point 연산의 결과가 다르기 때문이다. Floating-point의 연산 결과 오차는 FPU의 반올림 처리 방식(rounding mode)으로 인해 나타나며, Exynos 4412의 FPU가 반올림 처리를 여러 방식[12]으로 해주는데 비해

표 9. 정확도(RMS) 비교
Table 9. Comparison accuracy(RMS) value.

구분	Exynos 4412	KC 705
EX 1	111.1050	111.1596
EX 2	41.8177	41.8147
EX 3	2.4316	2.4318

Xilinx에서 제공하는 FPU의 반올림 처리 방식[13]은 기본 방식만 사용하기 때문이다. 여기서 발생한 오차가 반복이 진행되면서 계속 누적되고, 소프트웨어와 하드웨어에서 계산한 중간 데이터 값에 차이가 발생한다. 이로 인해 데이터의 흐름을 결정하는 ρ값(표 1)도 변하게 되며, 각 case가 수행되는 횟수 및 전체 반복 횟수 또한 달라지게 된다. 이와 같은 오차는 최종 결과값에도 영향을 미치기 때문에 정확도 측정이 필요하다. 표 9는 RFM 수행 후, 얻어진 fundamental 행렬의 정확도를 비교한 결과이다.

Ⅳ장 3절에서 RFM의 목적 함수는 fundamental의 정확도 $F(\mathbf{x})$임을 기술하였다. 그러므로 정확도는 $F(\mathbf{x})$의 RMS값으로 나타낼 수 있다. 표 8과 같이 반복 횟수의 차이가 있어도, 표 9와 같이 Exynos 4412에서 측정한 정확도와 KC 705에서 측정한 정확도는 거의 동일함을 알 수 있다.

5. 다른 구현 사례와의 비교

본 논문에서 구현한 LM 알고리즘의 하드웨어는 높은 정확도를 위해 내부 연산을 IEEE-754 floating format으로 연산하며, 응용 분야가 매우 복잡하다는 특징을 가지고 있다. 이와 유사한 사례로 fitting 문제에 대해 LM 알고리즘을 하드웨어로 구현한 사례[14]가 존재하며, floating format으로 내부 연산을 수행한다. 응용 분야가 다르기 때문에 정확한 비교 대상이 될 수 없지만, 표 10, 11을 통해 구현한 하드웨어에 사용된 연산기 개수와 HW 자원을 얼마나 사용했는지 비교하였다.

표 10, 11은 [14]와 본 논문에서 사용한 연산기의 수

표 10. 연산기의 수 비교
Table 10. Comparison number of operators.

operator	[14]		본 논문	
	32bit	64bit	32bit	64bit
Adders	7	23	23	27
Multipliers	25	43	17	31
Dividers	7	10	1	5
Root operators	–	–	2	5

표 11. 사용한 하드웨어 자원 비교
Table 11. Comparison platform and HW utilization.

구분	Platform	Occupied Slices	DSP Blocks
[14]	Vertex 6	37680	768
본 논문	Kintex 7	26517	519

표 12. 연산 속도의 향상 비교
Table 12. Comparison improvement of calculation velocity.

구분	[14]		본 논문	
	SW	HW	SW	HW
Platform	Mirco blaze	Vertex 6	Cortex A9	Kintex 7
동작속도	150MHz	100MHz	1.4GHz	200MHz
속도향상	최대 40배		최대 74배	

와 사용한 하드웨어 자원인 occupied slice 및 DSP block을 나타낸다. 각 platform은 DSP48E1이라는 동일한 DSP block을 사용한다. Adder와 multiplier에 DSP block이 사용되며, 특히 64bit multiplier는 다른 연산기에 비해 많은 DSP block이 사용된다. 이와 같은 이유로 [14]가 본 논문 보다 많은 DSP를 사용하고 있는 것으로 예측할 수 있다. 또한 Impulse C[15]를 통해 하드웨어 구현을 했다고 언급하고 하였으며 구현된 구조에 대한 언급이 없기 때문에 구조에 대한 비교가 힘들다. 이와 같은 이유로 단순 크기에 대한 비교를 위해 표 11을 보면, [14]의 하드웨어 크기가 더 큼을 알 수 있다.

마지막으로 [14]와 본 논문의 속도 향상에 대한 결과는 표 12와 같다. SW 동작을 위한 프로세서의 속도가 같다고 가정한다면, 본 논문이 [14]에 비해 속도 개선이 더 잘 되었음을 알 수 있다.

Ⅵ. 결 론

본 논문에서는 설계한 LM 알고리즘의 하드웨어는 연산의 집중도가 높은 Jocobi 행렬 연산의 파이프라인 처리와 내적 연산의 누적을 병렬처리 하여 속도를 향상시켰다. 설계한 하드웨어의 성능을 측정하기 위해 RFM을 응용 분야로 활용하였으며, 임베디드 프로세서 Exynos 4412(ARM Cortex-A9 1GHz)에 비해 32.6~74.3배 향상되었음을 알 수 있었다. 또한 기존에 설계된 [14]보다 FPGA 크기가 작고 속도 개선이 더 잘 되었음을 알 수 있었다. 하지만 연산들이 부동 소수점 방식으로 이루어지고 목적 함수의 복잡도가 높기 때문에, 로직 사이즈가 매우 크다는 단점이 있음을 알 수 있었다.

LM 알고리즘은 다양한 분야에서 활용되고 있지만 하드웨어에 관한 연구가 최근에서야 시작이 되고 있다. 본 논문처럼 목적 함수가 복잡하고 고차원인 응용 분야의 경우, 본 논문에서 제안한 하드웨어의 구조를 참고하여 다른 응용 분야에 대한 가속 엔진 개발에 많은 도움이 될 것으로 기대된다.

REFERENCES

[1] Golub, G. "Numerical methods for solving linear least squares problems." Numerische Mathematik 7.3 (1965): 206-216.

[2] Gratton, Serge, Amos S. Lawless, and Nancy K. Nichols. "Approximate Gauss-Newton methods for nonlinear least squares problems." SIAM Journal on Optimization 18.1 (2007): 106-132.

[3] Levenberg, Kenneth. "A method for the solution of certain problems in least squares." Quarterly of applied mathematics 2 (1944): 164-168.

[4] Marquardt, "An Algorithm for Least-Squares Estimation of Nonlinear Parameters", Journal of the society for Industrial and Applied Mathematics, Vol. 11, No. 2(Jun.,1963), 431-441.

[5] Powell, Michael JD. "A hybrid method for nonlinear equations." Numerical methods for nonlinear algebraic equations 7 (1970): 87-114.

[6] R. Hartley and A. Zisserman, "Multiple View Geometry in Computer Vision", Cambridge University Press, 2000.

[7] David C. Lay, "Linear Algebra and Its Applications, Third Edition", Pearson, 14-25.

[8] Madsen, Kaj, Hans Bruun Nielsen, and Ole Tingleff. Methods for non-linear least squares problems. 1999.

[9] Lowe, David G. "Object recognition from local scale-invariant features." Computer vision, 1999. The proceedings of the seventh IEEE international conference on. Vol. 2. Ieee, 1999.

[10] Mount, David M. ANN programming manual. Technical report, Dept. of Computer Science, U. of Maryland, 1998.

[11] Golub, Gene H., and Christian Reinsch. "Singular value decomposition and least squares solutions." Numerische Mathematik 14.5 (1970): 403-420.

[12] ARM, "Cortex-A9 Floating-Point Unit (FPU) Technical Reference Manual(ARM DDI 0408), Revision :r2p2", : 44.

[13] Xilinx, "floating_point_ds335.pdf",(March, 2011): 5.

[14] Blasco, José M., et al. "Maximum Likelihood Estimation and Non-Linear Least Squares Fitting Implementation in FPGA Devices for High Resolution Hodoscopy." (2013): 1-7.

[15] ImpulseC, Impulse CoDeveloperTM [Online]. Available: http://www.impulseaccelerated.com

저 자 소 개

이 명 진(학생회원)
2012년 광운대학교 전자통신 공학과 학사 졸업
2014년 8월 광운대학교 전자통신 공학과 석사 졸업
<주관심분야 : 영상처리 및 인식, Soc 설계, 임베디드 시스템>

정 용 진(정회원)
1983년 서울대학교 제어계측 공학과 학사 졸업
1983년 3월 ~ 1989년 8월 한국전자통신연구원
1995년 미국 UMASS 전자전산 공학과 석사 및 박사 졸업
1995년 4월 ~ 1999년 2월 삼성전자 반도체 수석 연구원
1999년 3월 ~ 현재 광운대학교 전자통신공학과 정교수
<주관심분야 : 무선통신, 정보보호, Soc 설계, 영상처리 및 인식, 임베디드 시스템>