CPU와 GPU의 병렬 처리를 이용한 고속 물체 인식 알고리즘 구현

The Implementation of Fast Object Recognition Using Parallel Processing on CPU and GPU

김 준 철, 정 용 한, 박 은 수, 최 학 남, 김 학 일*, 허 욱 렬
(Jun-Chul Kim, Young-Han Jung, Eun-Soo Park, Xuenan Cui, Hak-Il Kim, and Uk-Youl Huh)

Abstract: This paper presents a fast feature extraction method for autonomous mobile robots utilizing parallel processing and based on OpenMP, SSE (Streaming SIMD Extension) and CUDA programming. In the first step on CPU version, the algorithms and codes are optimized and then implemented by parallel processing. The parallel algorithms are debugged to maintain the same level of performance and the process for extracting key points and obtaining dominant orientation with respect to key points is parallelized. After extraction, a parallel descriptor via SSE instructions is constructed. And the GPU version also implemented by parallel processing using CUDA based on the SIFT. The GPU-Parallel descriptor achieves an acceleration up to five times compared with the CPU-Parallel descriptor, but it shows the lower performance than CPU version. CPU version also speed-up the four and half times compared with the original SIFT while maintaining robust performance.

Keywords: parallel processing, OpenMP, SSE, CUDA, SIFT, SURF

I. 서론

자율 주행 이동 로봇에서 물체 인식 기술은 노출된 환경에서 이동하면서 주변 물체를 인식해야 하기 때문에 다양한 어려운 문제가 존재하게 된다. 일반적으로 카메라의 위치 변화, 조명의 변화, 물체의 회전, 복잡한 배경, 다른 물체의 의한 가려짐과 같은 외부적인 요인 때문에 인식 성능이 떨어진다[1]. 이러한 문제점을 해결하기 위해 다양하게 변하는 외부 환경 속에서 강인하게 물체를 인식할 수 있도록 다양한 연구가 진행되고 있다[2-8].

일반적으로 물체 인식의 방법은 크게 세가지로 나누어 접근 할 수 있다. 먼저 전역 정보나 지역 정보를 이용하여 인식에 필요한 특징점을 어떻게 추출(representation) 할 것인가의 문제를 고려해야 한다. 그 다음 추출된 특징점을 바탕으로 데이터를 어떻게 분류할 것인가를 고려하게 된다[5]. 이러한 학습(learning)의 과정이 이루어지고 나면 새로운 데이터를 어떻게 정합하여 인식(recognition)할 것인가의 문제가 남게 된다. 표현과 학습, 그리고 인식의 과정은 독립적이기 보다는 서로 연계되어 강인한 인식 성능을 보안하기 위한 방향으로 연구되고 있다[3-7].

특히 특징 기반 물체인식 연구 가운데 영상의 크기와 회전에 강인한 특성을 지닌 SIFT (Scale Invariant Feature Transform) [2] 알고리즘이 제안되었고, Mikolajczyk[9,10]의 실험에서 SIFT 알고리즘을 기본으로 하는 다양한 방법들이 우수한 성능을 가짐을 보여 주었다. 또한 SIFT 알고리즘의 성능을 보안하고 속도를 개선하기 위하여 PCA-SIFT[3], SURF (Speed up Robust Features) [4]나 전역정보를 이용한 방법[6]등 다양한 연구가 진행되고 있다. 이러한 SIFT 서술자(descriptor)를 기반으로 하는 방법은 높은 차원의 벡터 성분을 가지고 있기 때문에 서술자 생성과 정합(matching)의 과정에서 많은 시간을 소요하게 된다. 따라서 실시간 인식을 필요로 하는 이동 로봇 환경에서는 안정된 성능과 고속화된 알고리즘이 요구된다.

본 논문에서는 OpenMP[11,12], SSE (Streaming SIMD Extension) [13-16]와 CUDA (Compute Unified Device Architecture)[17]을 이용하여 움직이는 로봇 환경에서 적합한 고속 특징점 추출 알고리즘을 구성하여 비교 분석하였다. 기본적으로 CPU상의 특징 추출은 안정된 성능을 보이는 Hessian[9,10]을 바탕으로 구성하였다. 병렬 처리로 구현된 스케일(scale) 공간상에서 후보 점을 찾고 불안정한 점은 박스 필터를 적용하여 제거하였고, 이러한 과정 또한 OpenMP를 통한 병렬처리로 구현하였다. 선택된 특징점을 바탕으로 Haar wavelet 을 이용하여 영역의 방향 성분을 구하고 방향에 따라 회전된 영역에서 서술자를 구하였다. 서술자는 선택된 영역의 Haar wavelet을 적용하고, 반응 값을 더하여 벡터를 구성하게 된다. 이러한 과정은 128 비트 레지스터(register)를 갖는 SIMD 명령어를 사용하기 위하여 영상 데이터의 재구성이 이루어졌다.

GPU상의 특징 추출은 SIFT 서술자 생성 방법을 바탕으로 CUDA를 이용하여 구성되었다. 먼저 DOG (Difference of Gaussian) 피라미드를 형성하여 후보점을 추출하고, 추출된 점을 바탕으로 그래디언트(gradient) 변화에 따라 방향과 서술자를 구성하였다.

II. OpenMP, SSE와 CUDA를 이용한 병렬처리

1. OpenMP를 이용한 병렬처리 시스템 구성

OpenMP는 공유 메모리(shared memory) 환경에서 멀티 스레드(thread) 생성을 위하여 범용적으로 사용되는 응용 프로

* 책임저자(Corresponding Author)
논문접수: 2009. 1. 30., 채택확정: 2009. 2. 25.
김준철, 정용한, 박은수, 최학남, 김학일: 인하대학교 정보공학과
(jckim@vision.inha.ac.kr/yhjung@vision.inha.ac.kr/espark@vision.inha.ac.kr/cncui@vision.inha.ac.kr/hikim@inha.ac.kr)
허욱렬: 인하대학교 전기공학과(uyhuh@inha.ac.kr)
※ 본 연구는 과학재단 특정기초 연구 지원 사업의 연구비 지원으로 연구 되었음.

그램 인터페이스 (API: Application Program Interface)이다. 구성은 크게 컴파일러 지시어(compiler directives), 실행시간 라이브러리(run-time library) 그리고 환경 변수(environmental variable)로 되어 있다. 간단한 지시어의 삽입으로 프로그램내의 루프를 병렬로 실행하는 것이 가능하지만 OpenMP를 지원하는 컴파일러가 모든 문제를 분석해서 자동으로 병렬화 해주는 것은 아니다. 순차 프로그램을 사전에 철저히 분석해서 병렬화에 의해 야기되는 동기화, 데이터 의존성 등의 문제들을 직접 처리해야만 한다[12].

OpenMP를 이용한 병렬처리에서 데이터 의존성이 많은 연산은 다른 동작이 완전하게 수행 될 때까지 기다려야 한다. 예를 들어, 서술자를 생성하기 위해서는 ROI (Region Of Interesting) 영역의 방향 정보가 필요하기 때문에 그 다음 단계는 방향 정보를 구하는 연산이 완료 되지 않으면 실행 할 수 없게 된다. 또한 공유 메모리에 의존적인 연산을 병렬 처리할 때, 메모리를 읽고 쓰는 연산이 동시에 실행되려고 하는 데이터 충돌(data collision) 이 발생하여 부정확한 결과를 발생하기도 한다. 이러한 문제를 해결하기 위해서는 특징점 추출과 서술자 생성 과정은 데이터 의존성을 최소화 하여 독립적으로 수행될 수 있도록 구성되어야 한다.

본 논문에서 각각의 스레드는 다른 레벨의 분산으로 가우시안 필터 된 영상을 입력으로 받고, 다른 크기와 값을 갖는 박스 필터를 적용하여 스케일 공간의 피라미드를 동시에 구성하게 된다. 후보 점에 대한 보정과 방향을 구하는 과정 또한 각 점에서 독립적으로 수행 되기 때문에 병렬적으로 처리하였다.

2. SSE를 이용한 병렬처리 시스템 구성

Intel Pentium II 프로세서에서 MMX 명령어를 지원하기 시작하여, Intel Pentium4 프로세서 및 듀오코어 환경에 이르러서는 MMX 명령어의 확장인 SSE, SSE2, SSE3, SSSE3 명령어를 지원한다. MMX명령어에서는 64비트의 레지스터(register)를 이용하여 8개의 8비트 정수 데이터를 동시에 처리할 수 있고, SSE 에서는 128비트 레지스터를 이용하여 그림 1에서처럼 4개의 SIMD-FP(Floating Point) 연산을 할 수 있다. SSE2는 16개의 8비트 정수 데이터를 동시에 처리하는 SIMD 명령어들과 2개의 실수(double precision) 실수 데이터를 처리하는 명령어를 포함하고 있다. 90nm 프로세서 기술 환경에서 포함된 SSE3 명령어에서는 13개의 명령어를 추가하여 SSE, SSE2, 및 x87-FP 기능을 강화 하였다. 또한 SIMD 정수 연산 기능을 강화하기 위한 32개의 명령어가 SSSE3 명령어에 포함되었다

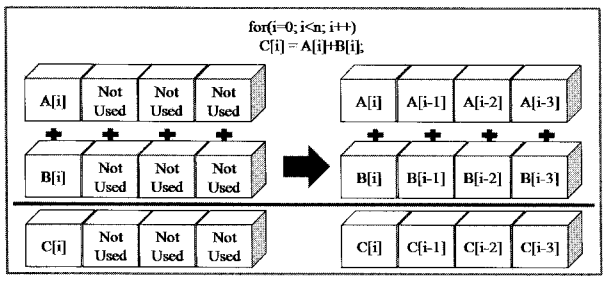


그림 1. SSE 명령어를 이용한 병렬 연산 예.

Fig. 1. The example of parallel computing using SSE instruction.

[14,15].

SIMD (Single Instruction Multiple Data) 명령어를 사용하기 위한 방법은 Inline Assembler, Automatic Vectorization, Intrinsic 함수 등이 있다[14,15]. Inline Assembler는 SSE 명령어를 직접 이용하여 구현하므로 가장 좋은 속도 개선을 이룰 수 있지만, 프로세서 구조에 따른 유동성이 떨어지고 개발 시간이 오래 걸리는 단점이 있다. Automatic Vectorization은 Intel C/C++ 컴파일러를 통해 성능을 개선하고자 하는 루프를 자동적으로 SSE를 사용하여 구현하게 되는 가장 쉬운 방법이다. 하지만 이 방법은 데이터 의존성이 강한 루프의 경우는 적용하기 어렵다. Intrinsic 함수는 Inline Assembler와 Auto Vectorization 의 중간 방법으로써 Intel 사에서 SSE 명령어를 사용하기 쉽도록 내재하여 지원하고 있다. Intrinsic 함수를 이용하면 SIMD 컴파일러가 지원되는 Intel 프로세서 모든 구조에 적합하고, Assembler와 비슷한 성능을 가져 올 수 있다. 따라서 본 연구에서는 Intrinsic 함수에 적합하도록 알고리즘을 변환하고 데이터를 재배열하여 고속 물체인식을 알고리즘을 구현하였다.

하지만, 데이터 의존도가 강한 루프(loop)는 OpenMP를 이용한 병렬처리처럼, SSE 명령어를 적용하기 어렵고 성능 개선에 제약을 갖게 된다. 실수(float)를 많이 포함하는 연산 또한 최적의 성능을 달성하기가 어려워 진다. 특징점 추출의 과정은 많은 루프를 갖고 있지만, 복잡도가 높아 Intrinsic 함수를 사용하기 어렵다. 하지만, 서술자를 생성하는 과정은 그림 1과 같이 고정된 벡터 형태를 주로 사용하게 되고 각각의 반응 값들은 독립적으로 수행 되기 때문에 SIMD 구조에 적합한 특성을 갖고 있다. 따라서, 서술자를 구하는 과정을 SSE 명령어를 이용한 데이터 재배열을 통해 수행 속도를 줄일 수 있다.

3. CUDA를 이용한 병렬처리 시스템 구성

GPU는 그래픽스 작업으로 인해 생기는 병목 현상을 해결하기 위해 설계된 특수한 목적의 처리 장치이다. 현재의 GPU 하드웨어 구조는 높은 트랜지스터 집적도와 SIMD 병렬 구조로 인해 그래픽 분야뿐 아니라 범용 처리 장치로써 사용하기 위한 다양한 연구가 진행 되어 왔다[19,21]. 일반적으로 GP-GPU (General Purpose Computation on GPUs)라 불리는 이 기술은 특히 빠른 속도로 대량의 데이터를 처리해야 하는 비디오 인코딩이나 컴퓨터비전 분야 등에서 실시간적인 성능 보장을 위해 사용되고 있다[18-20].

CUDA는 C언어를 이용하여 GPU에서 범용 컴퓨팅을 위해 설계된 하드웨어와 소프트웨어 구조를 말한다. NVIDIA에서 드라이버 및 소프트웨어 개발 툴킷을 제작하여 배포하고 있으며 자사의 Geforce 8 series, Quadro NVS 130M, Tesla 이상의 하드웨어에서 사용이 가능하다. CUDA는 그래픽스 하드웨어를 하나의 독립적인 플랫폼으로 간주하여 프로그래밍 할 수 있는 환경을 제공하며, 사용자의 그래픽스 파이프라인에 대한 이해의 필요성을 최소화 시켜준다. 또한 CPU가 GPU에서 동작되는 프로세스를 호출하는 방식으로 그림 2와 같이CPU와 GPU가 동시에 작동 될 수 있는 복합 시스템 개발을 가능하게 한다[17].

CUDA를 이용한 특징점 추출의 과정은 그림 2와 같은 모델에 의해 수행된다. CUDA에서 실행될 커널(kernel) 함수가 준비되면, 커널을 호출하기 전 그리드(grid)를 구성해야 한다.

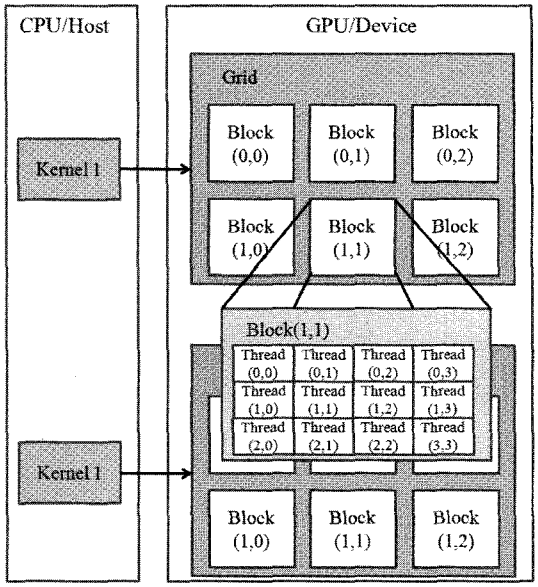


그림 2. CUDA의 프로그래밍 모델(reprinted[17]).
Fig. 2. Heterogeneous programming model with CUDA.

그리드는 다시 각각의 블록(block)으로 구성되고, 하나의 블록은 많은 수의 스레드를 생성하고 서로 데이터를 공유하며 병렬처리를 수행하게 된다. 스레드별로 실행 함수 단위인 커널(kernel)을 수행하게 되며, 커널의 구성에 따라 수행 속도의 큰 차이를 보일 수 있다.

III. CPU 상의 병렬 특징 추출 알고리즘 구현

동적인 환경에서 물체 인식을 위해서는 영상의 크기, 회전, 잡음 또는 조명과 같은 외부 환경 변화에 강인하고 안정된 특징점 추출이 필요하다. 특징점 추출 방법에는 Harris, Harris-Laplace, Hessian-Laplace[9,10]등이 주로 사용되고 있다. Harris 코너 검출을 기반으로 하는 Harris-Laplace 방법은 조명변화와 시점변화에 강인한 장점을 가지고 있다. 하지만 스케일 공간에서 극 값을 검출하기 위해 가우시안 필터링 된 영상에 라플라시안 필터를 적용해야 하기 때문에 연산량이 많다.

Harris-Laplace 방법에 비해 Hessian-Laplace 방법은 가우시안 차분(DoG: Difference of Gaussian) 방법과 유사하고 강인한 성능을 유지하면서 연산량이 상대적으로 적다. 따라서 본 논문에서는 Hessian을 기반으로 특징점을 추출하고 박스 필터를 적용하여 안정성을 보장하였다. 특징점 추출의 과정은 가장 먼저 스케일 공간상의 후보점을 Hessian Matrix를 이용하여 추출하고, 추출된 후보에 대하여 테일러 급수를 이용하여 특징점의 위치를 세부 위치로 보정한다.

1. 병렬 스케일 공간 구성

영상의 Hessian 행렬식(determinant)을 이용하여 후보점을 추출하기 위해 스케일 공간 영상, $L(x,y,\sigma)$ 을 식 (1)에서처럼 가우시안 함수, $G(x,y,\sigma)$ 와 입력영상, $I(x,y)$ 과의 컨벌루션으로 구성된다

$$L(x,y,\sigma) = G(x,y,\sigma) * I(x,y) \tag{1}$$

$L(x,y,\sigma)$ 는 다른 레벨의 분산을 갖는 가우시안 함수에 의해 블러된 영상이다. 이러한 스케일 공간은 모든 가능한 스케일 영상에서 안정적인 점들을 검색하기 때문에 영상의 크기 변화에 대하여 강인한 성능을 보인다.

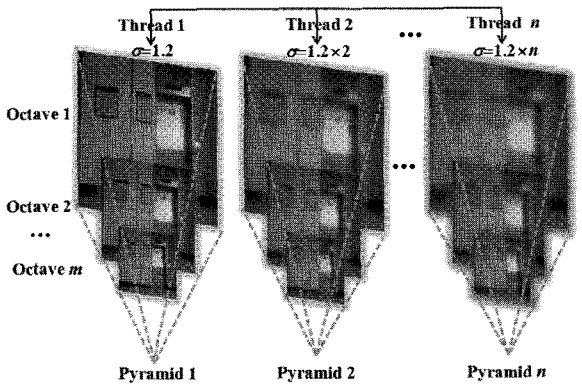


그림 3. 병렬로 구성된 스케일 공간 영상.
Fig. 3. Scale spaces image using parallel processing.

스케일 공간의 처리과정에서 발생하는 데이터 의존도는 병렬 처리를 위하여 데이터 재배열과 루프 내의 불변 코드의 재 배치 등을 통해 감소시켰다. 가우시안 함수에서 스케일 레벨이, σ_1, ..., σ_n과 같이 주어지면 m개의 옥타브 레벨을 갖는 피라미드를 구성하기 위해 그림 3과 같이 n개의 스레드가 필요하다. 각 피라미드의 입력은 다른 레벨의 가우시안 함수에 의해 블러 되고 동시에 다운셈플 되어 피라미드를 구성한다. n과 m의 값은 서로 독립적이며 둘 중에 하나의 값이라도 증가하면, 수행 시간과 성능의 두 가지 측면에서 trade-off 관계를 갖게 된다[2]. 또한 스레드의 개수는 프로세서의 개수보다 클 수 있지만, 이러한 경우 활성화 되지 않는 스레드는 활성화 된 스레드의 작업이 프로세서에서 완료 될 때까지 기다려야 하기 때문에 오버헤드가(overhead) 증가하게 된다. 본 논문에서는 4개의 피라미드 n과 4개의 옥타브 m을 가지고 실험하였다.

2. 병렬 특징점 추출

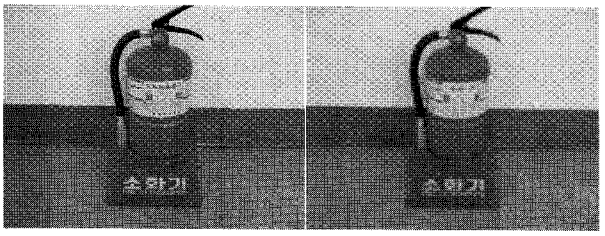
스케일 공간 영상에서 Hessian 행렬은 가우시안 블러된 영상을 2차 미분하여 구하여 진다.

$$H(x,y,\sigma) = \begin{bmatrix} L_{xx}(x,y,\sigma) & L_{xy}(x,y,\sigma) \\ L_{xy}(x,y,\sigma) & L_{yy}(x,y,\sigma) \end{bmatrix} \tag{2}$$

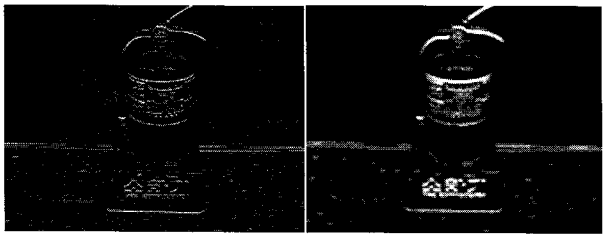
여기서 L_{xy}는 가우시안 영상을 수평(x-방향) 과 수직(y-방향)에 대하여 2차 미분한 값이다. 일반적인 2차 미분 영상은 그림 4(c)와 같이 잡음이나 불필요한 점들이 존재하여 특징점 추출 시 많은 점들을 내포하게 된다. 따라서 SURF[4]에서 제안된 박스 필터를 적용하여 (d)와 같이 잡음 제거와 함께 특징 부분에 가중치를 주어 안정된 점들을 추출할 수 있다.

그림 5(a)의 9×9 박스 필터는 σ=1.2의 LoG (Laplacian of Gaussian) 예측 분포 이며, 스케일 공간에서 가장 높은 스케일 영상에 적용 될 필터이다. 박스 필터의 사이즈는 입력영상의 옥타브 레벨에 따라 증가하게 된다. 가우시안의 분산 값과 적용될 박스 필터의 크기는 고정되어 있기 때문에 D_{xx}, D_{yy}와 D_{xy}로 정의 된 필터의 반응 값을 사전에 계산 할 수 있다.

그림 5의 입력 영상을 위한 옥타브 숫자는 그림 3에서처럼 하나의 피라미드 내에서 순서를 나타내고, 피라미드 숫자는



(a) Original image (b) Blurred image ($\sigma = 1.2$)



(c) Second derived image (d) Box filtered image

그림 4. 각 단계별 결과 영상.

Fig. 4. Resulting images for each step.

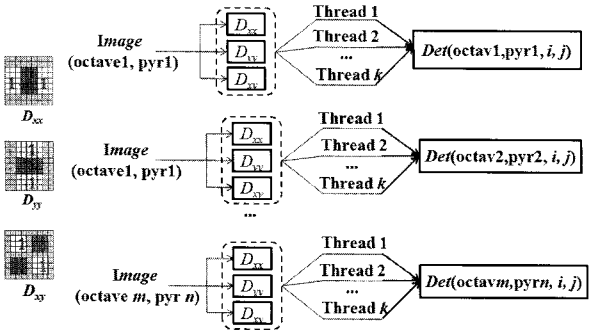


그림 5. 병렬 특징점 추출 (a) LoG의 x- (D_{xx}), y- (D_{yy})와 xy- (D_{xy}) 방향에 대한 예측 필터 (b) 예측 필터의 병렬 적용.

Fig. 5. Parallel configuration of key point detection (a) approximation of LoG in x- (D_{xx}), y- (D_{yy}) and xy- direction (D_{xy}) (b) parallel application of approximations.

공간의 순서를 나타낸다. 미리 예측된 박스 필터는 k개의 스레드에 의해 나누어져 병렬로 처리되고, 행렬식은 모든 스레드의 결과가 완료되면 계산되기 때문에 k가 증가할수록 오버헤드는 증가하게 된다. 본 논문에서는 4개의 스레드를 갖고 실험하였다.

필터는 모든 스케일 공간 영상에 적용되고, 루프 안에 지시어 삽입을 통해 각각의 스레드에서 동시에 처리하게 된다. 또한 필터의 적용은 적분 영상, $I_s(x,y) = \sum_{i=0}^{i<x}\sum_{j=0}^{i<y} I(i,j)$, 을 이용하여 추가적인 네 번의 연산으로 이루어져 수행 속도를 감소 시키고, 필터 크기에 독립적인 성능을 갖는다. 적분 영상은 다음 절에 설명 될 wavelet 반응을 구할 시에도 사용하게 된다. 박스 필터의 가중치는 계산의 효율을 갖기 위해 일정하게 적용하지만, Hessian 행렬식은 각 점에서 다음 식, $\frac{|L_{xy}(1.2)|_F |D_{xx}(9)|_F}{|L_{xx}(1.2)|_F |D_{xy}(9)|_F} = \theta$ 에 의해 조절 된다. 여기서 $|L|_F$ 는 *Frobenius* 크기를 나타내며 Hessian 행렬식은 식 (3)에 의해 계산 된다

$$Det(\mathbf{H}) = D_{xx}D_{yy} - (\theta \times D_{xy})^2 \tag{3}$$

행렬식 값이 일정한 임계 값을 넘게 되면, 대응 되는 점은 후보 점이라 판단하고 저장하게 된다.

앞 단계에서 추출된 후보점들은 여전히 불안정한 점들을 포함하고 있어서 안정성 여부에 대하여 구별해야 한다. 만약 한 점의 행렬식 값이 그 점을 중심으로 이웃한 영상의 3×3 영역의 모든 행렬식 값보다 극 값을 갖는 다면, 그 점은 안정적이라 판단하여 저장하게 된다. 저장된 점들은 3차원 fitting 모델에 적합한 Taylor 시리즈를 적용하여 보간을 하게 된다[8].

3. SSE명령어를 이용한 서술자 구성

서술자는 기본적으로 고차원의 벡터 성분을 갖고 있기 때문에 생성이나 정합 과정에서 많은 시간을 소요하게 된다. 따라서 본 논문은 SIFT의 속도를 개선 시킨 SURF를 재구성하여 고속화된 서술자 알고리즘을 구현하였다. 먼저, 각 특징점에서 지배적인 방향 성분을 OpenMP를 이용하여 병렬적으로 추출한다. 그리고 특징점을 중심으로 방향에 따라 ROI를 회전시켜 SSE 명령어를 이용하여 서술자를 생성하게 된다.

3.1 병렬 방향 성분 추출

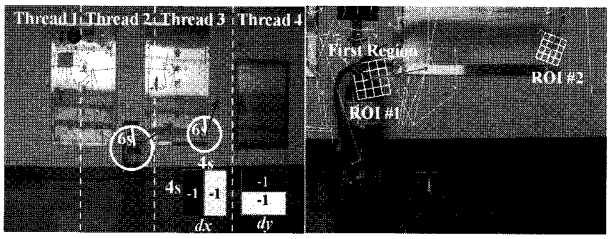
입력 영상에서 물체의 회전에 강인하기 위해서는 각 특징점에 대한 방향 정보가 필요하다. 본 논문에서는 그림 6(a)처럼 Haar wavelet의 x와 y 방향에 대한 반응을 계산하여 방향을 구하게 된다. Wavelet의 크기는 $4s$이고, 필터는 특징점을 중심으로 반지름이 $6s$의 원 영역 내에서 적용 된다. 여기서 s는 각 점의 스케일을 나타낸다. 이 과정은 박스 필터에 사용된 방법처럼 적분 영상을 사용하고 영상을 스레드에 의해 분할하여 처리하게 된다. 즉, wavelet반응은 추가적인 여섯 번의 연산으로 구해지고, OpenMP를 이용하여 각 영역에서 동시에 수행된다.

원 영역 내에서 wavelet 반응을 더하여 히스토그램을 생성하게 되고, 가장 큰 벡터를 그 영역을 나타내는 방향 성분으로 저장한다. 그림 6의 화살표는 저장된 특징점들의 방향 성분을 가리킨다.

3.2 SSE 명령어를 이용한 병렬 서술자 구성

서술자를 생성하기 위하여 먼저 그림 6(b)처럼 각 특징점의 방향에 따라 사각형 ROI를 회전 시킨다. ROI의 크기는 $16s \times 16s$이고, 그 영역은 다시 $4s \times 4s$ 윈도우 크기의 부분 영역 나누어진다. ROI 크기는 실험적으로 결정할 수 있지만, $16s$가 16의 배수로 데이터를 정렬 할 수 있기 때문에 SSE 명령어 구조에 적합하여 본 논문에 사용하였다. 그 다음 방향 정보를 추출하는 과정과 마찬가지로 수직과 수평 방향(dx와 dy)으로 wavelet 반응을 구하게 된다.

부분 영역에서 방향에 따른 반응 dx와 dy는 각각 더하여 Σdx와 Σdy를 생성하고, 반응에 대한 절대값도 더하여 부분 영역에 대한 서술자를 하나의 벡터 형태, $[\sum dx; \sum dy; \sum |dx|; \sum |dy|]$ 로 저장하게 된다. 한 점에 대한 서술자는 하나의 ROI 내의 16개 부분 영역으로부터 16개의 벡터를 갖게 된다. 이러한 과정들은 그림 7에서처럼 128 비트 레지스터를 바탕



(a) (b)

그림 6. 방향성분과 ROI를 구하는 과정 (a) 각 특징점에서 Haar Wavelet을 이용한 병렬 방향 추출 (b) 방향 성분을 중심으로 회전된 ROI.

Fig. 6. Procedure to obtain the orientation and ROI (a) Orientation of each key point obtained in parallel by the Haar-wavelet and (b) Rotation of the ROI with respect to the dominant orientation.

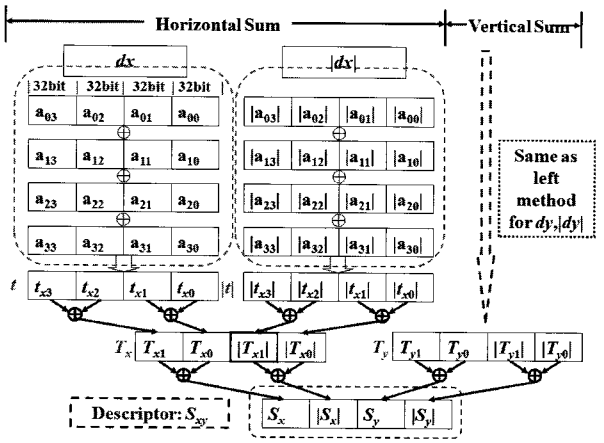


그림 7. 부분 영역에서 병렬 서술자 구성.

Fig. 7. Parallel process for constructing a descriptor for a sub-region.

으로 SSE 명령어를 이용하여 병렬처리 되었다.

먼저, 부분 영역의 각 반응에 대한 합을 구하기 위하여, 하나의 ROI 내의 첫 번째 부분 영역의 *dx*를 *_mm_load_si128*를 이용하여 로드(load) 한다. 그 다음 절대값 |*dx*|는 *_mm_abs_epi32*를 이용하여 계산된다. 기본적으로 wavelet 반응은 실수(16 bits)의 데이터 타입이기 때문에 128 bit 레지스터를 이용하여 4개의 배정도(double precision) 실수(32 bits) 로드 하였다. 그림 7에서 4개의 레지스터에 모든 *dx*값이 로드 되고 나면 각 행의 성분들은 *_mm_add_epi32* 함수를 이용하여 더해져 *t*에 저장된다. *t*의 일시적인 결과들은 수평 방향으로 더해져 T_x로 저장되고 T_x의 결과들은 다시 더해져 S_x로 저장된다. 같은 방법으로 |*dx*|, *dy*와 |*dy*|의 합들이 병렬로 계산되고, 결과들은 레지스터 S_{xy} 에 차례대로 저장된다. 결국 S_{xy}는 하나의 부분 영역에 대한 서술자를 나타낸다.

하나의 ROI의 16개 부분 영역에서 64차원의 서술자가 구해지고 나면, 조명의 변화에 강인 하기 위하여 정규화 과정이 필요하게 된다. 그림 8과 9는 한 특징점에 대한 서술자의 l^2-norm 을 병렬로 처리하는 과정과 그 것을 이용하여 정규화 시키는 과정을 보여주고 있다. l^2-norm은 $\mathbf{x}=[x_1,x_2,...,x_n]$ 일 때, $|\mathbf{x}|=\sqrt{\sum_{k=1}^{n}x_k^{\,2}}$ 와 같이 정의 되며 벡터의 크기를 구하기 위해 주로 사용된다. 본 논문에서 *n*은 64가 된다. 벡터의 크기를 구하기 위해 각 성분의 제곱 합을 계산해야 하고, 각 성분에 대한 제곱은 *_mm_mul_ps*를 이용하여 계산 된다. 이러한 과정은 그림 8과 같이 M_1, M_2, ... M_n 까지 반복되고, 각 레지스터의 성분들을 더하고 나면 ROI 내의 모든 성분의 합이 구해진다.

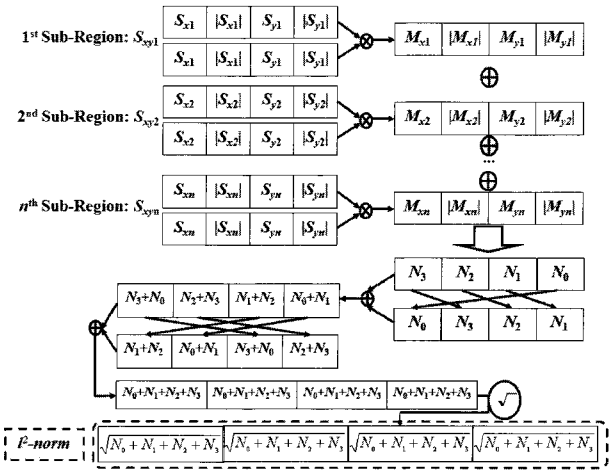


그림 8. ROI 에서 l^2-norm의 병렬 처리.

Fig. 8. Parallel process for computing the l^2-norm at each ROI area.

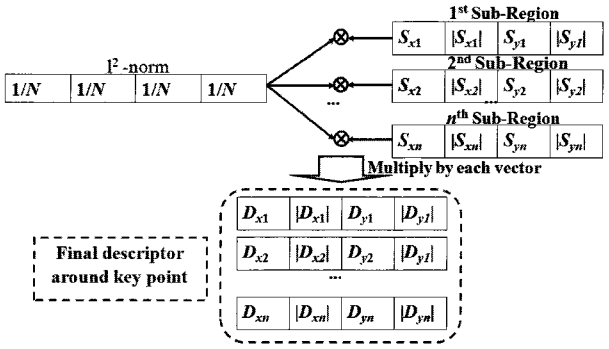


그림 9. 특징점에서 정규화 된 서술자의 병렬 처리.

Fig. 9. Parallel process for normalizing the descriptor vector for each key point.

정규화 과정을 동시에 병렬로 처리하기 위해 하나의 레지스터에 4개의 같은 l^2-norm을 저장해야 한다. 이를 위해 하나의 *_mm_shuffle_ps* 함수를 이용하여 데이터를 재배열 시킨 다음, *_mm_add_epi32*를 이용하여 더하고 한번 더 반복한다. 마지막으로 *_mm_sqrt_ps* 함수를 이용하여 근호를 계산하면 $\left[\sum dx;\sum dy;\sum|dx|;\sum|dy|\right]$ 의 l^2-norm이 구해진다.

그림 9는 4개의 l^2-norm을 이용하여 *n*개의 부분 영역의 4차원 서술자를 병렬로 정규화 하는 과정을 보여주고 있다. 기본적인 연산은 *_mm_div_ps*와 *_mm_mul_ps*를 이용하여 이루어 진다. 최종적인 결과는 하나의 특징점에서 64 차원의 정규화된 서술자를 나타낸다. 이러한 과정은 각각의 그림 6(b)에서 보여주는 것처럼 특징점을 중심으로 하는 모든 ROI 영역에서 수행되고, 연산이 완료되면 서술자는 *_mm_store_ps*를 이용하여 레지스터 타입으로 저장하게 된다.

IV. GPU 상의 병렬 특징 추출 알고리즘 구현

CUDA는 GPU의 메모리를 그림 10에서와 같이 register, shared memory의 온칩(on-chip)과 global/device, texture memory의 오프칩(off-chip)으로 나누어 관리한다. 메모리의 전송 지연을 줄이기 위해 가급적 온칩 메모리만을 사용 하도록 구성하는 것이 보다 많은 속도 증가를 가져오며, 적은 양의 온칩 메모리를 효율적으로 관리하는 것이 중요하다. 하나의 블록은 하나의 멀티 프로세서에 할당되어 공유 메모리와 레지스터를 공동으로 활용하게 되며, 블록 내의 스레드는 배열과 일대일로 매핑하여 병렬처리 성능을 극대화 시킬 수 있다[17].

SIFT를 이용한 서술자 생성은 그림 11에서처럼 먼저 입력 영상을 GPU상에 업로드(upload)하여 영상의 크기를 축소 시킴과 동시에 constant memory에 할당된 가우시안 커널을 이용하여 필터를 적용하게 된다. 각 단계별 블록과 스레드에 개수는 필터와 입력 영상의 크기에 따라 달라지게 된다. 첫번째 단계에서는 800×640 영상에 대해 5×40의 블록 안에 32개의 스레드를 사용하였다. 각 스레드에 의해 블러된 영상은 마지막 단계인 서술자 생성에서 빠른 접근을 위하여 texture 메모리에 할당하게 된다.

블러된 영상은 50×40 그리드와 16×16 스레드 블록의 입력으로 넘겨져 가우시안 차분 영상을 얻게 된다. 그 다음 얻어진 가우시안 차분 영상에서 이웃한 영상 영역의 점들과 비교하여 국부적인 극대, 극소점의 위치를 찾아 후보로 선택하게 된다. 국부 영역 내에 비교를 위한 일시적인 값들은 shared 메모리에 할당하여 공유된 값들을 비교하여 중복 연산을 피함으로써 수행 속도를 증가시켰다. 앞선 단계에서 선택된 후보점들은 여전히 불안정한 점들을 포함하고 있기 때문에 그림 11의 네 번째 단계를 통해 제거하게 된다. 여기서 블록의 수는 스레드 간의 균형을 위하여 선택된 후보점의 개수에 따라 달라지게 된다. 즉, 후보점 개수×32로 블록을 나누고 각 블록 안에 32개의 스레드를 할당하여 특징점을 추출하게 된다.

추출된 특징 점들은 그래디언트 변화를 이용하여 각 점들의 방위 히스토그램을 구하고, 히스토그램 내에서 최대 각을 그 점의 방향 성분으로 저장하게 된다. 히스토 그램을 생성하여 최대 값을 찾는 과정은 앞선 단계와 유사하게 shared 메모리를 이용하여 영역내의 값을 공유하여 비교 연산이 이루어 진다. 최종적으로 구해진 방향 성분을 기준으로 texture 메모리에 할당된 블러 영상에서 회전된 영역을 설정하고, 8 방위에 대한 누적 합을 통해 서술자를 생성하게 된다. 여기서 블록의 수는 각 특징점에서 소요되는 연산 강도가 크기 때문에 특징점의 개수가 되고 스레드는 16개를 생성하여 사용하였다.

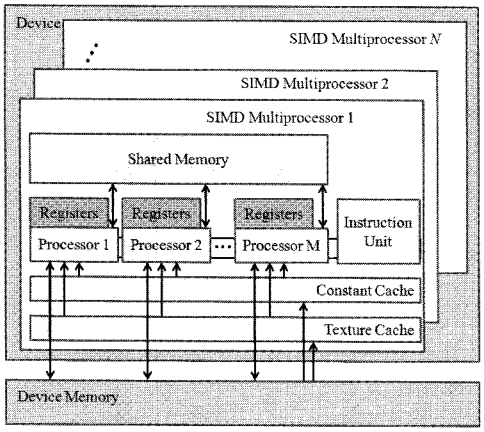


그림 10. 멀티 프로세서 구조(reprinted[17]).
Fig. 10. A set of SIMT multiprocessors.

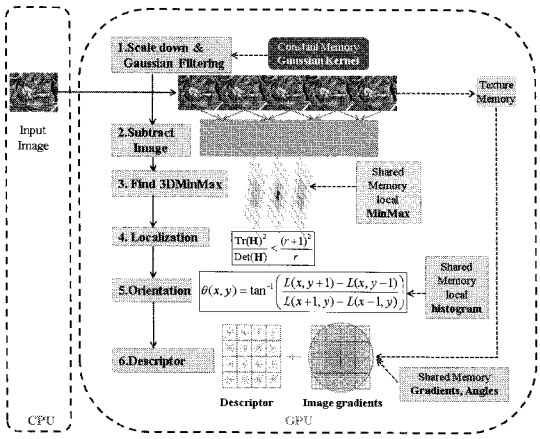


그림 11. CUDA를 이용한 SIFT 서술자 생성 과정.
Fig. 11. Parallel process for constructing a descriptor using CUDA.

V. 실험 결과

본 논문은 CPU와 GPU상의 병렬처리를 이용한 고속화 방법과 기존 연구들의 성능과 수행 속도를 비교하였다. 실험은 800×640 graffiti [8] 영상을 Windows XP 기반에 Intel Core2 Duo 2.66GHz CPU와 Nvidia Geforce 8800 GT 그래픽 카드를 사용하였다. 그림 12는 CPU와 GPU의 병렬처리 특징 추출 결과 영상을 나타내며, 물체 부분에 특징점이 많이 분포함을 확인할 수 있다. 그림 13은 Harris-Laplace, Hessian-Laplace, Harris-Affine, Hessian-Affine과 CPU레벨의 고속화 방법의 카메라 각도에 따른 특징점 추출 시간을 보여주고 있다. 각각의 방법은 10번 반복하여 평균 시간을 측정하였으며, 특징점의 수는 임계값을 조절하여 대략 1600개를 유지하도록 하였다. 제안된 Parallel-Hessian 방법은 기존의 방법 중 가장 빠른 Hessian-Laplace 방법에 비해 2.5배 빠른 결과를 가져왔다.

서술자 생성에 대한 평가는 SIFT [2], PCA-SIFT [4] 그리고 SURF [5]를 CPU와 GPU의 병렬 서술자를 영상의 크기에 따라 수행 속도를 비교하여 이루어졌다. 각 서술자에 대한 특징점 추출은 Hessian을 바탕으로 이루어졌고, 특징점 추출에 대한 시간을 포함하여 그림 14에서 보여주고 있다. OpenMP와 SSE를 이용한 서술자의 생성 방법은 일반적인 CPU 상의 SIFT 알고리즘에 비해 4.5배 빠르고, 800×640 크기의 동영상

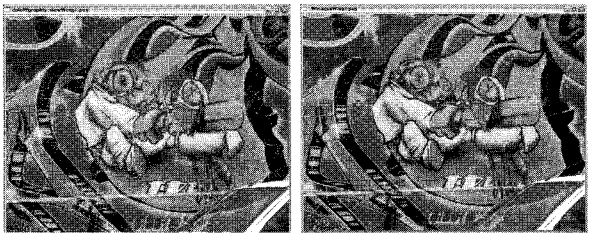


그림 12. GPU와 CPU의 병렬 특징 추출 결과 영상.
Fig. 12. Result of parallel feature extraction on GPU and CPU.

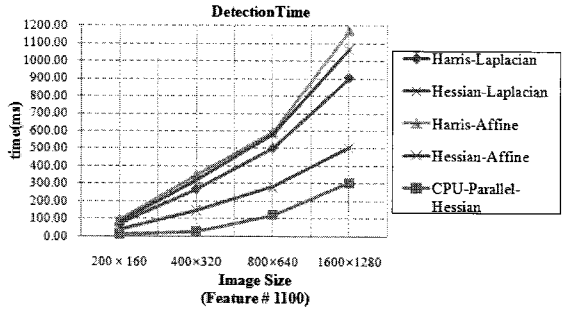


그림 13. 영상의 크기에 특징점 추출 시간 비교.

Fig. 13. Comparison of the detection time with respect to variation of image-size.

에 1,100개 정도의 특징점이 추출 됐을 때, 평균 8Hz 이상의 성능을 보였다.

그림 14와 표 1을 통해 알 수 있듯이 CUDA를 이용한 서술자 생성 방법이 일반적인 방법뿐 아니라 CPU상의 병렬처리 방법보다 평균 5배 이상으로 가장 빠르고, 동영상 적용 시 15Hz 이상의 성능을 보였다. 하지만, GPU 상에서 추출된 서술자를 실제 물체 인식 시스템에 적용할 경우, 데이터 베이스의 검색 및 정합을 위하여 CPU상으로 다운로드 과정이 필요하기 때문에 데이터 베이스 증가에 따라 속도가 저하 될 수 있다.

본 논문에서 제안된 방법의 성능은 Mikolajczyk[10]에서 제공되는 방법을 이용하여 그림 15와 같은 *recall* vs. 1-*precision* 그래프를 통해 평가 하였다. 이론적인 *recall* vs. 1-*precision* 그래프는 *precision* 값이 증가 하더라도 일정한 *recall* 값을 갖지

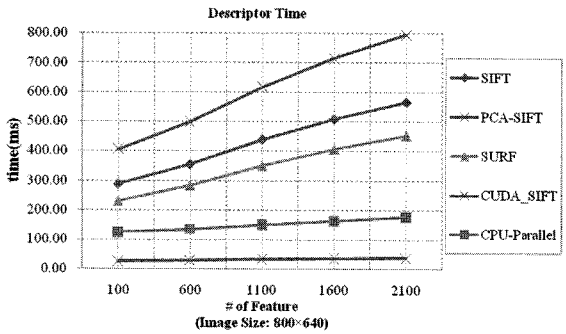


그림 14. 영상의 크기에 따른 서술자 생성 시간 비교.

Fig. 14. Comparison of the processing time in constructing descriptors with respect to variation of feature-count.

표 1. UDA를 이용한 서술자 생성의 단계별 수행 속도 및 병렬 CPU 버전과 Speed-up 비교.

Table 1. Descriptor construction time for each step and Speed-up compared to Parallel- CPU version.

View	Parallel-CPU	Key-point Detection	Descriptor	CUDA	
				Total	Speed-up
0	163	22	13	33	4.9
20	161	22	36	36	4.4
30	162	22	33	33	5.3
40	161	21	32	32	5.2
50	162	23	34	34	4.7
60	161	23	31	31	5.1

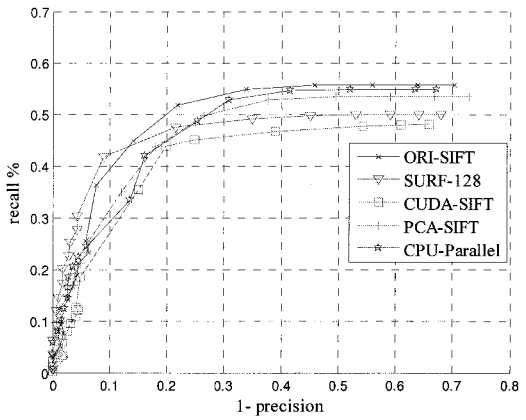


그림 15. 서술자의 recall vs. 1-precision 그래프.

Fig. 15. Performance of descriptor respect to recall versus 1-precision curve.

만, 실제 시스템에서는 다양한 요소에 의해 단조 증가하는 특성을 보인다. 즉, 그림처럼 1-*precision*이 증가함에 따라 *recall*이 증가 하는 것을 알 수 있고, CPU상의 병렬처리 방법 또한 기존의 방법과 유사한 패턴을 보임으로써 성능을 유지하고 있음을 보여주고 있다. 하지만, CUDA를 이용한 방법의 경우 기존의 방법에 다소 낮은 정확도를 보이고 있어, 속도 개선에 따른 trade-off를 가져왔음을 알 수 있다.

VI. 결론

본 논문에서는 OpenMP와 SSE 명령어 및 CUDA를 이용한 특징점 추출과 서술자 생성을 비교 분석하였다. CUDA를 이용한 방법은 가장 빠른 수행 속도를 보이지만 기존의 방법에 비해 다소 낮은 성능을 보이고 있고, OpenMP와 SSE를 이용한 방법은 CUDA를 이용한 방법에 비해 느리지만, 비교적 안정된 성능을 가져왔다. 이러한 고속 특징 추출 방법은 복잡하고 동적인 환경에서 실시간으로 물체 인식을 가능하게 하고, 다양한 영상처리 분야에서 응용이 가능하다. 하지만, GPU를 이용한 특징 추출의 방법은 검색 및 정합의 과정에서 오버헤드가 발생하게 될 수 있다. 따라서 추후 GPU 상에서 데이터 베이스의 검색 및 정합이 이루어진다면 좀더 개선된 성능을 가져올 것으로 기대한다.

참고문헌

[1] S. Ullman, "High-level Vision - Object Recognition and Visual Cognition." MIT Press, 2000.

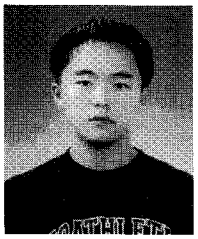
[2] D. G Lowe, Distinctive image features from scale invariant keypoints, *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91-110, 2004.

[3] Y. Ke and R. Sukthankar, "PCA-SIFT: A more distinctive representation for local image descriptors," *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 511-517, 2004.

[4] H. Bay, Y. Tuytelaars, and G L. Van, "SURF: Speeded up robust features," *Computer Vision and Image Understanding*, vol. 110, pp. 346-359, 2008.

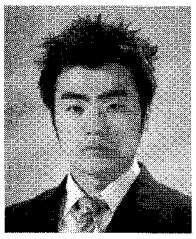
[5] S. Cagnoni, F. Bergenti, M. Mordonini, and G Adorni, "Evolving binary classifiers through Parallel computation of multiple fitness cases," *IEEE Trans. on Systems, Man, and*

Cybernetics, part B, vol. 35, no. 3, 2005.
[6] E. N. Mortensen, D. Hongli, and L. Shapiro, "A SIFT descriptor with global context," *Proc. of the Conference on Computer Vision and Pattern Recognition*, pp. 184-190, 2005.
[7] J. Shotton, A. Blake, and R. Cipolla, "Multiscale categorical object recognition using contour fragments," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 7, pp. 1270-1281, 2008.
[8] M. Brown and D. G. Lowe, "Invariant features from interest point groups," *Proc. of the British Machine Vision Conference*, pp. 656-665, 2002.
[9] K. Mikolajczyk, et al., "A comparison of affine region detectors," *International Journal of Computer Vision*, vol. 65, no. 1, pp. 43-72, 2005.
[10] K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, pp. 1615-1630, 2005.
[11] OpenMP Architecture Review Board, OpenMP Application Program Interface, ver. 2.5, 2005, http://www.openmp.org.
[12] R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, and R. Menon, Parallel Programming in OpenMP, Morgan Kaufmann, 2005.
[13] C. Nicolescu and P. Jonker, "A data and task parallel image processing environment," Parallel Computing, vol. 28, pp. 945-965, 2005.
[14] Intel®64 and IA-32 Architectures Software Developer's Manual, Intel Corporation, vol 2A,B, Instruction Set Reference., 2007, http://www.intel.com
[15] Intel®64 and IA-32 Architectures Optimization Reference manual, Intel Corporation, 2007, http://www.intel.com.
[16] S. Asadollah, B. Juurlink, and S. Vassiliadis, "Performance comparison of SIMD implementations of the discrete wavelet transform," *Proc. of the 16th IEEE International Conference on Application-Specific Systems, Architecture Processors*, pp. 393-398, 2005.
[17] NVIDIA CUDA, Programming Guide, v2.1, 2008, http://www.nvidia.com/object/cuda_home.html
[18] L. Yuancheng and R. Duraiswami, "Canny edge detection on NVIDIA CUDA," *Computer Vision and Pattern Recognition Workshops*, 2008.
[19] J. D. Owens, D. Luebke, N. govindaraju, M. Harris, J. Kruuger, A. E. Lefohn, and T. J. Purcell, " A survey of general-purpose computaion on graphics hardware," *In Eurographics 2005*, State of the Art Reports, pp. 21-51, 2005.
[20] V. Garcia, E. Debreuve, and M. Barlaud, "Fast k nearest neighbor search using GPU," *Computer Vision and Pattern Recognition Workshops*, 2008.
[21] N. S. Sudipta, J.-M. Frahm, M. Pollefeys, and Y. Genc, "Feature tracking and matching in video using programmable graphics hardware," *Machine Vision and Applications*, 2007.



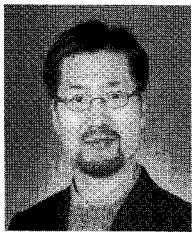
김 준 철

2008년 인하대학교 정보통신 공학부(공학사). 2008년~현재 인하대학교 정보공학과 석사 과정 재학중. 관심분야는 로봇비젼, 물체인식, 병렬 영상처리, 물체추적.



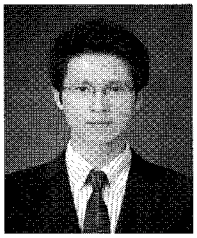
박 은 수

2007년 인하대학교 정보통신 공학부(공학사). 2007년~현재 인하대학교 정보공학과 석사 과정 재학중. 관심분야는 병렬 처리, AOI, 로봇 비젼.



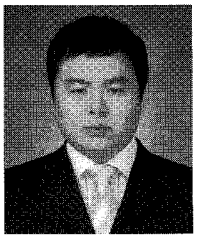
김 학 일

1983년 서울대학교 제어계측공학과(공학사). 1985년 Purdue Univ. 전기/컴퓨터 공학과(공학석사). 1990년 Purdue Univ. 전기/컴퓨터 공학과(공학박사). 1990년~2001년 인하대학교 자동화 공학과 조교수. 2001년~현재 인하대학교 정보통신공학과 교수. 2002년~현재 한국정보보호학회 생체인증연구회 위원장. 2003년~현재 ISO/IEC JTC1/SC37 WG5 Rapporteur. 2005년~현재 ITU-T SG17 Q.8 (Telebiometircs) Rapporteur. 관심분야는 패턴인식, 컴퓨터비전, 바이오 인식, 자율주행로봇, 센서 및 계측 공학.



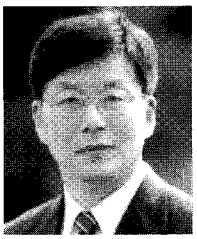
정 용 한

2008년 인하대학교 정보통신 공학부(공학사). 2008년~현재 인하대학교 정보공학과 석사 과정 재학중. 관심분야는 네비게이션, 스테레오 비젼, 위치인식.



최 학 남

2007년 상명대학교 컴퓨터과학과(공학석사). 2007년~현재 인하대학교 정보공학과 박사 과정 재학중. 관심분야는 로봇비젼, 지능제어, 위치인식, 병렬처리.



허 욱 렬

1974년 서울대학교 전기공학과(공학사). 1978년 서울대학교 전기공학과(공학석사). 1982년 서울대학교 전기공학과(공학박사). 1980년~현재 인하대학교 전기공학과 교수. 2004년~2006년 한국 과학재단 전기정보 전문위원. 1997년~현재 대한전기학회 부회장. 2001년~현재 제어로봇시스템학회 부회장. 관심분야는 지능 제어 시스템, 인공지능, 모션제어, 자율주행로봇.