한국컴퓨터정보학회 하계학술대회 논문집 제21권 제2호 (2013. 7)

모바일 기반 OpenCV 라이브러리를 이용한 마커리스 객체 인식 성능 향상

정현섭[O], 윤희원[*], 김신덕[+]
[O*+]연세대학교 컴퓨터과학과
e-mail:jung2413@naver.com[O], yunhiwen@hotmail.com[*], sdkim@yonsei.ac.kr[+]

Performance improvement for marker-less object recognition through OpenCV mobile library

Hyeon-Sub Jung[O], Yin Xiyuan[*], Shin-Dug Kim[+]
[O*+]Dept. of Computer Science, Yonsei University

● 요 약 ●

본 논문에서는 모바일 기반 OpenCV 라이브러리를 이용한 마커리스 객체 인석 성능 향상을 위한 소프트웨어적인 관점의 방법을 제안한다. 기존의 마커리스 기반 알고리즘을 이용하여 테스트를 수행한 후 성능에 저하를 발생시키는 요인들을 분석하고 그에 따른 상황별 적절한 해결책을 제시한다. 이에 따라 크게 프로그램 코드 개선, 마커리스 기반 알고리즘 코드 개선, 센서를 활용한 성능 향상을 도모한다. 프로그램 코드 개선은 테스트 결과를 분석 한 후 수행시간이 가장 많이 소요되는 함수를 최적화하고 또한 최적의 특징점의 수를 제한한다. 마커리스 기반 알고리즘 코드 개선은 병렬 처리가 제공되는 모바일에 한하여 병렬처리 기법으로 코드를 수정한다. 마지막 센서를 활용한 성능향상은 실시간 작업 처리 단위를 묶음으로 처리하였을 때 발생하는 품질의 저하를 보정하는 역할을 수행한다. 본 논문에서는 이러한 마커리스 객체 인식 성능 향상 방법을 소프트웨어적인 관점에서 제안하고 이에 대한 결과 모바일 기반 실시간 증강현실 서비스를 위한 성능 향상 면에서 효과적이다.

키워드: 객체인식(Object recognition), 스마트폰(Smart phone), OpenCV(Open Computer Vision), 마커리스(Marker-less), 패턴인식(Pattern recognition)

I. 서 론

고속 데이터 처리 능력을 제공하는 중앙처리장치와 카메라 그리고 3D 그래픽 처리가 가능한 칩셋의 도입 등 최근 스마트 폰의 발전은 과거와 비교가 되지 않을 정도로 진행되고 있다. 이와 같은 스마트 폰 하드웨어의 눈부신 발달은 휴대 단말을 기반으로 제공되는 서비스의 품질 향상과 응용의 다양성을 증대 시키는 원동력으로 작용하고 있다. 이는 최근에 각광을 받고 있는 증강현실 서비스의 품질 향상과 응용의 다양성을 증대 시키는 원동력으로 작용하고 있다. 특히 증강현실 서비스를 위해서는 서비스 하고자 하는 객체를 인식하는 것이 가장 큰 화제이다. 객체를 인식하는 방법으로 과거 마커(Marker) 기반의 객체 인식에서 현재는 마커리스(Marker-less) 기반의 객체 인식으로 바뀌어 가고 있는 추세이다. 마커리스 기반의 객체 인식은 컴퓨터 비전에서 가장 도전적인 과제 중에 하나이다. 이미 많은 알고리즘들이 객체를 인식하기 위해 연구가 되어 왔다. 이러한 많은 알고리즘들이 빠르며, 외부 요인에도 강인하나 여전히 알고리즘들은 복잡하며 대부분의 알고리즘들이 실시간에 적합하지 않다. 이러한 문제를 해결하기 위해 새로운 알고리즘을 개발하거나, 기존의 알고리즘을 개선하거나, 혹은 하드웨어의 아키텍처 디자인을 바꾸는 등 이와 같은 연구가 많이 진행되어 왔다.

본 논문에서는 소프트웨어 적인 관점에서 모바일 자원의 활용을 통한 기존의 마커리스 기반의 객체 인식 알고리즘을 활용하여 증강현실 서비스의 성능 향상을 위한 방법을 연구한다.

· 본 연구는 정부(교육기술과학부)의 재원으로 이공분야기초연구사업의 일환으로 수행함. [과제번호 2012R1A1A2043400]
+ 교신 저자

II. 관련 연구

기존의 마커리스 기반의 객체 인식에 대한 연구는 새로운 알고리즘의 개발, 기존 알고리즘의 성능 향상 및 하드웨어의 아키텍처 디자인의 변경을 통한 방법으로 연구가 진행이 되어 왔다. 마커리스 기반의 객체 인식의 대표적인 알고리즘으로 SIFT(Scale Invariant Features Transform)[1]가 있다. SIFT 알고리즘은 명

암, 회전, 스케일에도 강건한 불변의 특징점을 찾아낸다. 하지만 이러한 장점이 속도적인 측면에서 상당한 느린 단점으로 적용되어 실시간에 적합하지 않다. 이후 SIFT의 장점을 살리며 단점을 보완해 나온 알고리즘이 SURF(Speed Up Robust Features)[2]이다. SURF 알고리즘은 Integral Image를 이용하여 4개의 point만 알면 그 영역의 면적을 간단하게 바로 계산이 가능하며 간편화한 Detector와 Descriptor를 활용하여 결과적으로 차원수를 줄여서 성능 향상을 도모하였다. 그러나 SIFT와 SURF는 부동 소수점 자료형 연산을 사용하기 때문에 연산량이 많아 여전히 모바일 환경의 실시간 서비스에 적합하지 않다. 부동 소수점이 아닌 부호 없는 정수형을 사용하는 FAST 알고리즘[3]은 Corner 정보를 이용하여 객체를 인식하는 방법으로 객체의 특징점 주변의 Corner 정보를 추출해 낸다. 하지만 FAST 알고리즘은 인식률 측면에서 약한 단점을 지니고 있다. 이후 나온 알고리즘으로 BRISK(Binary Robust Invariant Scalable Keypoints)[4], FREAK가 각각 있으며 이들은 기존의 알고리즘들을 개선하여 나온 알고리즘들이다.

III. 본 론

1. 마커리스 기반 객체 인식 방법

1.1 시스템 구성

본 논문에서는 모바일 환경에서의 마커리스 객체 인식을 위하여 기존의 알고리즘 중 FAST 알고리즘을 이용하며 객체를 인식한다. FAST 알고리즘을 선택한 경우는 인식률은 낮지만 성능이 가장 우수한 알고리즘을 기준으로 선정하였기 때문이다. 객체를 인식하는 단계는 이미지로부터 특징점을 추출해 낸 후 Descriptor를 이용해 정보를 가공한 후 미리 훈련된 객체 정보와의 특징점을 통해 매칭 작업을 수행한다.아래 그림 1은 증강현실 응용 프로그램 시스템을 도식화 한 것이다.

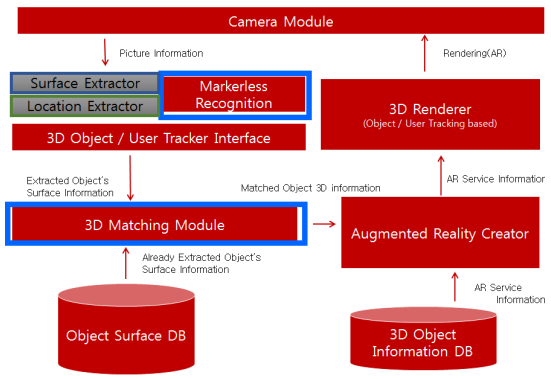


그림 1. 증강현실 응용 프로그램 시스템
Fig. 1. Augmented reality application system

1.2 특징점을 이용한 객체 인식

본 논문에서 객체 인식 과정은 마커리스 기반이다. 객체의 순수한 정보를 특징점이라고 하며 이 특징점을 이용해 인식을 한다. 서비스 하고자 하는 객체를 사전에 먼저 훈련시킨 후 그 훈련된 데이터를 DB화를 하여 선처리를 한다. 선처리 작업이 끝이 나면 모바일로부터 영상을 획득한 후 이미지에서 객체의 특징점을 추출해 낸다. 다음은 특징점 정보를 활용해 디스크립터 정보를 만든 후 선처리 된 데이터와 비교하여 매칭 작업을 수행한다. 다음 그림 2는 객체 인식 흐름도이다.

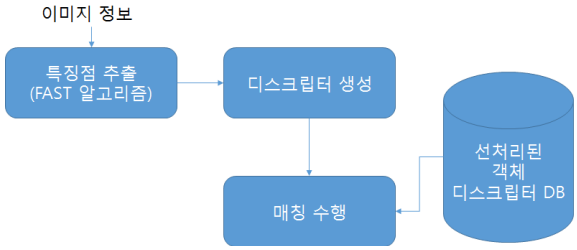


그림 2. 객체 인식 흐름도
Fig. 2. Object recognition flowchart

특징점 추출은 관련된 연구에서 소개한 FAST 알고리즘을 이용한다. FAST 알고리즘은 인식률은 다소 낮지만 가장 빠른 속도를 자랑한다. 또한 Threshold 값에 따라서 객체의 특징점을 추출해내는 개수 또한 틀려지기 때문에 Threshold 값을 조정하여 최소의 특징점 정보로 매칭 수행이 가능하다.

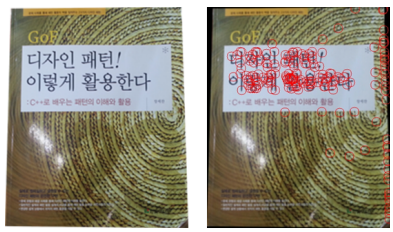
2. 마커리스 기반 객체 인식 성능 향상 방법

마커리스 기반 객체 인식 성능 향상을 위하여 기존의 FAST 알고리즘을 그대로 이용하여 테스트를 수행하였다. 테스트 시스템 환경은 아래 표1과 같다.

표 1. 테스트 시스템 환경
Table 1. Test System Environment

항목	값
CPU	Quad Core (1.4GHz X 4)
Android OS	4.0(Ice Cream Sandwich)
Camera	8MP AF w/Flash, BSI
Memory	1GB RAM

아래 그림 3은 테스트 객체와 특징점이 추출된 후 테스트 객체이다.



(a) (b)

그림 3. (a) 원본 테스트 객체. (b) 특징점이 추출된 후의 테스트 객체
Fig. 3. (a) Original test object (b) Test object after extracting features.

위의 테스트 환경과 테스트 객체를 수행해본 결과 아래 그림 4와 같은 로그 결과를 얻었다.

```
FindFeatures-match elapsed time : 0.0008 seconds
FindFeatures-findHomography elapsed time : 0.3773 seconds
FindFeatures-perspectiveTransform elapsed time : 0.0001 seconds
FindFeatures-line elapsed time : 0.0003 seconds
```

그림 4. 테스트 결과
Fig. 4. Test result

위의 로그 결과를 분석하면 findHomography함수에서 약 0.4초의 시간비용이 발생한 것을 알 수 있다. 이 말은 즉 모바일 환경에서 객체 인식 프로그램의 속도가 약 2.5fps 밖에 나오질 못한다는 결과이다.

2.1 프로그램 코드 개선

findHomography 함수 내부를 분석 해 본 결과 선처리된 DB에 저장된 객체 정보와 실시간으로 들어온 객체 정보와 매칭을 했을 때 매칭율이 낮으면 함수의 수행시간이 오래 걸린다는 것을 알 수 있었다. 따라서 기존 테스트 코드에 일정 매칭율 이상일 경우에만 저 함수를 수행하도록 변경해보았다. 아래 그림 5는 코드의 일부분이다.

```
const static float LIMIT_MATCH_PERCENTAGE = 20.0f;
const float matchPercentage = CalculateMatchPercentage(matchedInformation);
if(matchPercentage > LIMIT_MATCH_PERCENTAGE)
{
    // do matching process
```

그림 5. findHomography 수행성능 개선을 위한 코드
Fig. 5. Code for improvement on findHomography function time cost

다음은 특징점 정보의 개수를 확인해보았다. 현재 테스트 이미지의 규격은 가로 200 × 세로 240이다. FAST 알고리즘의 Threshold 값을 50으로 했을 때 특징점 개수가 185개가 나왔다. FAST 알고리즘의 특징 중 하나는 특징점의 개수가 너무 많이 나온다는 것이다. 그래서 반복적인 실험을 통해 최적의 특징점의 개수를 50으로 한정하였다.

2.2 FAST 알고리즘 코드 개선

안드로이드 OpenCV에서는 ARM_NEON 아키텍처를 적용하여 병렬처리를 가능하도록 한다. OpenCV 라이브러리 내부를 분석 해 본 결과 이미 ARM_NEON 아키텍처가 적용되는 스마트폰에는 자동으로 병렬처리가 되도록 구현이 되어있다. 그러나 몇몇 부분에 적용되어 있지 않다는 것을 발견해냈다. 이는 입력 값에 따라 병렬처리가 틀려지기 때문에 기존 코드를 유지했지만 본 연구에서는 Threshold 값을 고정화 시켜 최적화를 수행하였다. 다음 그림 6은 ARM NEON코드를 적용한 일부분이다.

```
for( i = -255; i <= 255; i++ )
    threshold_tab[i+255] = (uchar)(i < -threshold ? 1 : i > threshold ? 2 : 0);
```

[Original code]

```
for( i = -255; i < -threshold; i+=16 )
{
    temp8x16_t = vdupq_n_u8((uint8_t)1);
    vst1q_u8(&threshold_tab[i+255], temp8x16_t);
}

for( ; i < -threshold; i++)
    threshold_tab[i+255] = (uchar)1;

for(; i <= threshold; i += 16)
{
    temp8x16_t = vdupq_n_u8((uint8_t)0);
    vst1q_u8(&threshold_tab[i+255], temp8x16_t);
}

for( ; i <= threshold; i++)
    threshold_tab[i+255] = (uchar)0;

for( ; i <= 255; i += 16)
{
    temp8x16_t = vdupq_n_u8((uint8_t)2);
    vst1q_u8(&threshold_tab[i+255], temp8x16_t);
}

for( ; i <= 255; i++)
    threshold_tab[i+255] = (uchar)2;
```

[New code for ARM_NEON]

그림 6. ARM NEON 코드를 적용한 예
Fig. 6. Code sample using ARM NEON

2.3 센서를 활용한 성능 향상

연구를 진행하면서 위의 2가지 방법을 적용한 후에 수행 성능을 측정해 본 결과 평균 6.2fps의 성능이 나왔다. 이는 기존의 2.5fps에 비해 약 2.4배 정도 향상을 가져왔음에도 불구하고, 여전히 모바일 환경에서 실시간 증강현실 응용프로그램을 수행하면 사람이 인식할 수 있을 정도로 느리다는 것을 알 수 있다. 최소 10프레임 이상의 성능향상을 도모하기 위해 현재 처리되는 단위를 줄여야 한다. 즉 초당 6.2fps이 나온다면 객체 인식을 하는 단위를 매번 해야 할 것을 두 번에 한 번씩 묶음으로 진행을 해야 한다는 것이다. 하지만 이렇게 되면 실시간 서비스의 품질이 떨어질 수밖에 없다. 그렇다면 서비스의 품질을 떨어트리지 않으면서 성능 향상을 도모하려면 사용자가 서비스하고 있는 객체를 계속 보고 있는지 아니면 다른 곳으로 이동 중인지 혹은 다른 객체를 원해서 카메라의 시야를 변경하고 있는지를 캐치하면 품질이 떨어지는 구간을 보정할 수 있다.

이를 위해 센서를 활용해서 현재의 상황을 알 수 있다. 안드로이드에는 여러 가지의 센서가 존재한다. 본 연구에서 사용한 센서는 가속계이다. 가속계는 스마트 폰이 막 움직이기 시작할 때 속도가 얼마나 급격하게 늘어나는지, 이동을 중지할 때 얼마나 빨리 멈추는지를 감지해내는 센서이다. 즉 사용자가 새로운 서비스를 원해서 이동중이라면 이전 속도에 비해 일정 수준 높아진다는 것을 알 수 있다. 다음 그림 7은 실험을 통해 얻어낸 가속계 데이터를 그래프로 나타낸 것이다.

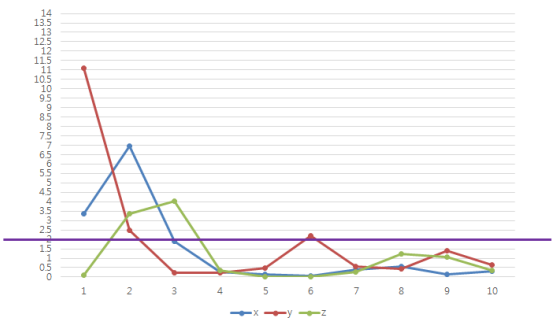
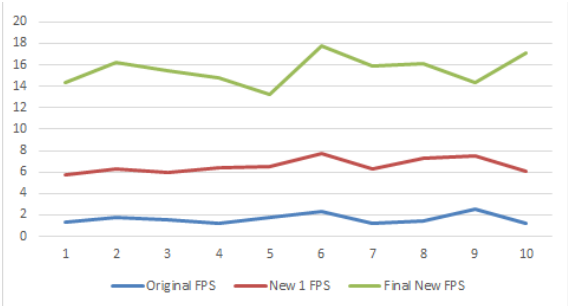


그림 7. x,y,z 가속계 데이터 값
Fig. 7. x,y,z accelerator data values

위와 같은 반복적인 실험을 통해 x,y,z 축의 가속계 값이 한 개라도 2이상의 값이 나온다면 사용자가 현재 서비스 받고 있는 객체에서 더 이상 관심이 없다는 것으로 알 수 있다.

3. 실험결과

기존의 FAST 알고리즘의 성능과 성능 향상을 위해 제안하는 방법1(2.1, 2.2), 모든 방법을 활용했을 때 나타나는 결과 데이터는 아래 그림 8과 같다.



실험 결과 평균 약 15.5fps의 결과가 나왔으며 이는 기존의 성능에 비해 약 6배 정도 향상이 가져왔다는 것을 알 수 있다. 하지만 5회에서 6회 사이에 급격한 프레임 차이를 알 수 있는데 이는 안드로이드 센서 정보가 민감하게 반응하기 때문에 나타나는 현상이다. 제안하는 객체 인식 방법을 적용하여 실제 증강현실 3D 응용 프로그램을 테스트 해 본 결과 약 10.3fps이라는 결과가 나왔다. 이 결과에는 객체 인식, 3D 모델 생성, 3D 모델 렌더링까지 모두 포함한 결과이다.

IV. 결 론

본 연구는 모바일 환경에서 마커리스 기반 객체 인식의 성능 향상을 목표로 두고 연구하였다. 기존의 많은 연구들과는 다르게 소프트웨어적인 관점에서 어떻게 성능을 향상시킬 것인가에 초점을 두었다. 먼저 기존의 방법대로 프로그램을 수행하고 로그 정보를 활용하여 성능에 영향을 미치는 부분을 최적화 혹은 개선하였다.

또한 스마트 폰 센서를 활용하여 실시간 처리단위를 변경함으로 인해 발생하는 품질저하를 보정하였다. 향후 과제로는 서비스하고자 하는 객체를 분류하여 해당하는 분류에 최적화된 객체 인식 알고리즘의 선택과 성능향상에 대하여 연구하고자 한다.

참고문헌

[1] David G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," International Journal of Computer Vision, Vo 60. no 2, pp. 91-110, November 2004.

[2] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool, "SURF: Speeded Up Robust Features," Computer Vision – ECCV 2006, Vol. 3951, pp. 404-417, 2006.

[3] Edward Rosten, and Tom Drummond, "Machine Learning for High-Speed Corner Detection," Computer Vision – ECCV 2006, Vol. 3951, pp. 430-443, 2006.

[4] Stefan Leutenegger, Margarita Chli and Roland Siegwart, "BRISK: Binary Robust invariant scalable keypoints," Computer Vision (ICCV), 2011 IEEE International Conference on, pp. 2548-2555, November 2011.

[5] A.Alahi, R. Ortiz, and P. Vandergheynst, "FREAK: Fast Retina Keypoint," Computer Vision (ICCV), Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on , pp. 510-517, June 2012.

[6] OpenCV, http://www.opencv.org