SURF 특징 검출기와 기술자를 이용한 파노라마 이미지 처리에 관한 연구

김남우* · 허창우*

*목원대학교

Study on the panorama image processing using the SURF feature detector and technicians.

Nam-woo Kim* · Chang-Wu Hur**

*MOKWON University

E-mail : gotree94@gmail.com

요 약

다중의 영상을 이용하여 하나의 파노라마 영상을 제작하는 기법은 컴퓨터 비전, 컴퓨터 그래픽스 등과 같은 여러 분야에서 널리 연구되고 있다. 파노라마 영상은 하나의 카메라에서 얻을 수 있는 영상의 한계, 즉 예를 들어 화각, 화질, 정보량 등의 한계를 극복할 수 있는 좋은 방법으로서 가상현실, 로봇비전 등과 같이 광각의 영상이 요구되는 다양한 분야에서 응용될 수 있다. 파노라마 영상은 단일 영상과 비교하여 보다 큰 몰입감을 제공한다는 점에서 큰 의미를 갖는다. 현재 다양한 파노라마 영상 제작 기법들이 존재하지만, 대부분의 기법들이 공통적으로 파노라마 영상을 구성할 때 각 영상에 존재하는 특징점 및 대응점을 검출하는 방식을 사용하고 있다. 본 논문에서 사용한 SURF(Speeded Up Robust Features) 알고리즘은 영상의 특징점을 검출할 때 영상의 흑백정보와 지역 공간 정보를 활용하는데, 영상의 크기 변화와 시점 검출에 강하며 SIFT(Scale Invariant Features Transform) 알고리즘에 비해 속도가 빠르다는 장점이 있어서 널리 사용되고 있다. 본 논문에서는 두 영상 사이 또는 하나의 영상과 여러 영상 사이에 대응되는 매칭을 계산하여 파노라마영상을 생성하는 처리 방법을 구현하고 기술하였다.

키워드

Panorama, SURF, descriptor, feature detector

I. 서론

최근 컴퓨터 성능의 향상과 카메라 가격의 하락으로 인해 영상처리와 관련된 다양한 프로그램들이 개발되고 있는 추세이다[1].

영상에서 임의의 점에 대한 고유한 특징을 계산하는 알고리즘은 파노라마 영상의 제작, 스테레오 영상에서의 3차원 정보 획득, 물체 인식, 이미지 분석 등에 다양하게 사용되는 핵심 기술이다.

특징 검출(feature detection)은 영상에서 관심있는 화소, 영역, 물체 등을 찾는 과정이고, 특징 기술(feature description)은 검출된 특징점 주위의 밝기, 색상, 그래디언트 방향 등의 정보를 계산하는 과정으로 매칭 또는 인식을 위해 사용된다.

II. 본론

특징검출에 있어서 가장 초기의 방법은 에지를 이용하는 방법이었다. 이 방법은 강도와 방향의 정보를 이용하여 특징점을 찾는 방법으로 다른 곳과 두드러지게 달라 풍부한 정보 추출 가능한 곳, 즉 에지 토막에서 곡률이 큰 지점을 코너로 검출하는 방법을 사용하였다. 코너 검출, dominant point 검출 등의 주제로 80년대 왕성한

연구가 이루어 졌으며, 90년대 소강 국면, 2000년대에는 지역 특징이라는 새로운 물줄기 출현으로 사라지게 되었다.

지역 특징이라는 방법은 명암 영상에서 직접 검출 하는것인데, 지역 특징은 위치, 스케일, 방향, 특징 벡터 ((y,x),s,Θ,x)로 표현한다.

이런 지역 특징을 위치와 스케일을 알아내는 검출 단계를 거쳐 방향과 특징 벡터 알아내는 기술 단계를 통해 최종 매칭까지 진행 할 수 있다.

지역 특징이 만족해야 할 특성으로는 반복성, 분별력, 지역성, 정확성, 적당한 양, 계산 효율 등이 있다.

이런 특징검출 방법 중에서 진보된 알고리즘 중의 하나인 SURF는 Herbert Bay의 논문 "SURF:Speeded Up Robust Features"에서 제시한 방법이다. SURF는 속도를 높이기 위해 적분 영상을 사용하여 헤시안을 계산하고, 기술자 계산에서도 적분 영상을 사용한다. SURF는 특허권이 설정된 알고리즘으로, 상업적으로 이용할 때는 이용이 제한될 수 있다. [3]

2.1 SURF 특징검출

SURF는 SIFT에 영향을 받아 유사한 단계로 특징을 검출 및 기술자를 계산하지만, 각 단계에서 사용하는 기법은 다르다. 관심 있는 측정점을 검출하기 위하여, 스케일 σ에서의 헤시안(Hessian) 행렬, $H(x,y,\sigma)$를 사용한다. [2]

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

여기서,

$$H_{xx}(x,y,\sigma)=G_{xx}(x,y,\sigma)*I(x,y)$$
$$=\frac{\partial^2}{\partial x^2}G(x,y,\sigma)*I(x,y)$$

$$H_{yy}(x,y,\sigma)=G_{yy}(x,y,\sigma)*I(x,y)$$
$$=\frac{\partial^2}{\partial y^2}G(x,y,\sigma)*I(x,y)$$

$$H_{xy}(x,y,\sigma)=G_{xy}(x,y,\sigma)*I(x,y)$$
$$=\frac{\partial^2}{\partial x\,\partial y}G(x,y,\sigma)*I(x,y)$$

$G_{xx}(x,y,\sigma)$, $G_{yy}(x,y,\sigma)$, $G_{xy}(x,y,\sigma)$은 가우시안 함수 $G(x,y,\sigma)$의 2차 미분으로, [그림1]과 같은 가우시안 박스 필터로 근사되어, 적분 영상(integral image)을 사용하여 빠르게 계산한다. 근사화된 가우시안 박스 필터로 계산한 H_{xx}, H_{yy}, H_{xy}를 D_{xx}, D_{yy}, D_{xy}라 할 때, 가중치 0.9를 사용하여 근사된 헤시안의 행렬식은 다음과 같이 계상하면, 마스크 크기에 관해 정규화되고 필터 크기에 무관하게 Frobenius 놈이 상수가 된다.

$$\det(H(x,y,\sigma))=$$
$$H_{zz}(x,y,\sigma)H_{yy}(x,y,\sigma)-H_{xy}(x,y,\sigma)^2$$

$$\det(H_{\approx})=D_{xx}D_{yy}-(0.9D_{xy})^2$$

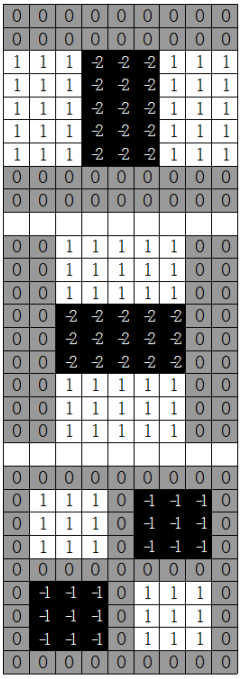


그림 1. 가우시안 함수의 2차 미분을 근사화한 9x9박스 필터

2.2 특징점의 방향 계산

특징점이 검출된 스케일 s를 이용하여 반지름 6s의 원현 이웃에서 x축, y축의 Haar웨이블릿을 적분 영상을 사용하여 웨이블릿의 길이를 4s로 하여 계산한다. 웨이블릿 반응 값이 계산되면, 특징점을 중심으로 한 가우시안 함수로 가능 필터링하고, 특징점 주위의 영역에서의 웨이블릿 반응값을 수평, 수축 방향으로 $\pi/3$를 커버하는 영약까지를 포함하는 윈도우 영역 내의 반응값의 합계를 계산하여, 특징점의 방향으로 계산한다.

2.3 기술자 계산

특징점의 방향에 따라, 특징점을 중심으로, 스케일 s에 의존하는 20s 크기의 사격형역을 4x4 영역으로 분할하고, 수평방향 Haar 웨이블릿 값

d_x, 수직 haar 웨이블릿 값 d_y를 이용하여, 각 부분 영역에서 extended=false 이면 4개 v[0], v[1], v[2], v[3]의 특징을 생성하고, extended=true이면 8개의 특징, v[0]에서 v[7]을 생각한다. 각 영역에서 특징은 다음과 같이 부분영역 내에서의 합계로 계산한다. 기술자를 계상하기 위한 합계 역시 적분 영상을 사용한다.

extended=false 인 경우

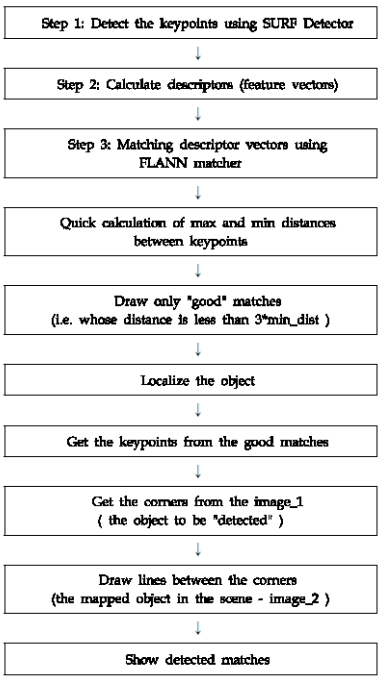
$$v[0] = \sum d_x, v[1] = \sum d_y$$
$$v[2] = \sum |d_x|, v[3] = \sum |d_y|$$

extended=true 인 경우

$$v[0] = \sum_{d_y \geq 0} d_x, v[1] = \sum_{d_y \geq 0} |d_x|$$
$$v[2] = \sum_{d_y < 0} d_x, v[3] = \sum_{d_y < 0} |d_x|$$
$$v[4] = \sum_{d_x \geq 0} d_y, v[5] = \sum_{d_x \geq 0} |d_y|$$
$$v[6] = \sum_{d_x < 0} d_y, v[7] = \sum_{d_x < 0} |d_y|$$

III. 결론

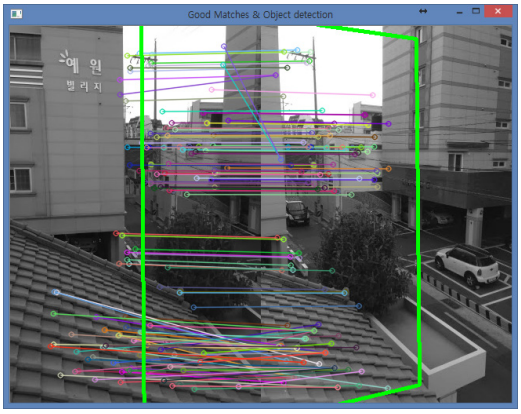
본 논문에서 구현한 영상연결의 방법의 흐름을 아래와 같다. [4]




```
Mat img_object = imread("c:\\S1.jpg", CV_LOAD_IMAGE_GRAYSCALE);
Mat img_scene = imread("c:\\S2.jpg", CV_LOAD_IMAGE_GRAYSCALE);


if (!img_object.data || !img_scene.data)
{
std::cout << " --(!) Error reading images " << std::endl; return -1;
}

//-- Step 1: Detect the keypoints using SURF Detector
int minHessian = 400;

SurfFeatureDetector detector(minHessian);

std::vector<KeyPoint> keypoints_object, keypoints_scene;

detector.detect(img_object, keypoints_object);
detector.detect(img_scene, keypoints_scene);

//-- Step 2: Calculate descriptors (feature vectors)
SurfDescriptorExtractor extractor;

Mat descriptors_object, descriptors_scene;

extractor.compute(img_object, keypoints_object, descriptors_object);
extractor.compute(img_scene, keypoints_scene, descriptors_scene);

//-- Step 3: Matching descriptor vectors using FLANN matcher
FlannBasedMatcher matcher;
std::vector< DMatch > matches;
matcher.match(descriptors_object, descriptors_scene, matches);

double max_dist = 0; double min_dist = 100;

//-- Quick calculation of max and min distances between keypoints
for (int i = 0; i < descriptors_object.rows; i++)
{
double dist = matches[i].distance;
if (dist < min_dist) min_dist = dist;
if (dist > max_dist) max_dist = dist;
}

printf("-- Max dist : %f \n", max_dist);
printf("-- Min dist : %f \n", min_dist);

//-- Draw only "good" matches (i.e. whose distance is less than 3*min_dist )
std::vector< DMatch > good_matches;

for (int i = 0; i < descriptors_object.rows; i++)
{
if (matches[i].distance < 3 * min_dist)
{
good_matches.push_back(matches[i]);
}
}

Mat img_matches;
drawMatches(img_object, keypoints_object, img_scene, keypoints_scene,
good_matches, img_matches, Scalar::all(-1), Scalar::all(-1),
vector<char>(), DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS);

//-- Localize the object
std::vector<Point2f> obj;
```

```
std::vector<Point2f> scene;

for (int i = 0; i < good_matches.size(); i++)
{
//-- Get the keypoints from the good matches
obj.push_back(keypoints_object[good_matches[i].queryIdx].pt);
scene.push_back(keypoints_scene[good_matches[i].trainIdx].pt);
}

Mat H = findHomography(obj, scene, CV_RANSAC);

//-- Get the corners from the image_1 ( the object to be "detected" )
std::vector<Point2f> obj_corners(4);
obj_corners[0] = cvPoint(0, 0); obj_corners[1] = cvPoint(img_object.cols, 0);
obj_corners[2] = cvPoint(img_object.cols, img_object.rows); obj_corners[3] = cvPoint(0,
img_object.rows);
std::vector<Point2f> scene_corners(4);

perspectiveTransform(obj_corners, scene_corners, H);

//-- Draw lines between the corners (the mapped object in the scene -
image_2 )
line(img_matches, scene_corners[0] + Point2f(img_object.cols, 0), scene_corners[1] +
Point2f(img_object.cols, 0), Scalar(0, 255, 0), 4);
line(img_matches, scene_corners[1] + Point2f(img_object.cols, 0), scene_corners[2] +
Point2f(img_object.cols, 0), Scalar(0, 255, 0), 4);
line(img_matches, scene_corners[2] + Point2f(img_object.cols, 0), scene_corners[3] +
Point2f(img_object.cols, 0), Scalar(0, 255, 0), 4);
line(img_matches, scene_corners[3] + Point2f(img_object.cols, 0), scene_corners[0] +
Point2f(img_object.cols, 0), Scalar(0, 255, 0), 4);

//-- Show detected matches
imshow("Good Matches & Object detection", img_matches);
```

본 논문에서는 두 영상 사이 또는 하나의 영상과 여러 영상 사이에 대응되는 매칭을 계산하여 파노라마영상을 생성하는 처리 방법을 구현하고 기술하였다.



참고문헌

[1] 라연, 신성식, 박현주, 권오봉, “SURF와 멀티밴드 블랜딩에 기반한 파노라마 스티칭”, 멀티미디어학회논문지, 제 14권, 제 2호, 201-209, 2011

[2] H. Bay, T. Tuytelaars, and L. Van Gool. “Surf: Speeded up Robust Features.” Computer Vision – ECCV, 404- 417. 2006

[3] Opencv org, OpenCV API Refernce, http://docs.opencv.org/doc/tutorials/features2d/feature_homography/feature_homography.html

[4] 김동근, C++ API OpenCV프로그래밍“, 가메출판사