고급소프트웨어실습
중간 리포트
(CSE 4152)

Due: 2020년, 10/19일, 23:59분

학번 :20181593
반 번호 :2반
이름 :계인혜

1. 이미지 검색 프로그램의 구현에 필요한 주요 함수들의 관계도 그래프를 자유로운 형식으로 그리고, 각 함수들의 역할에 대하여 설명하시오. (이미지는 모두 로컬 하드 드라이브에 저장되어 있는 것으로 가정. 함수의 개수는 주요 기능 위주로 5개 이상 10개 이하로 선정할 것)

이미지 검색은 주어진 검색어 또는 이미지로부터 연관된 이미지를 찾는 것으로 기존에는 수동적으로 이미지에 할당하거나 웹페이지에서 얻어진 keywords를 통한 검색이 일반적으로 사용되었다. 그러나 점차 이미지의 내용에 기반한 검색(Content Based Image Retrival)으로 발전중이다. 해당 문제에서는 CBIR을 이용한 SIFT(Scale Invariant Feature Transformation)에 대하여 살펴보도록 하자.

SIFT 알고리즘의 도식적인 표현은 다음과 같다.

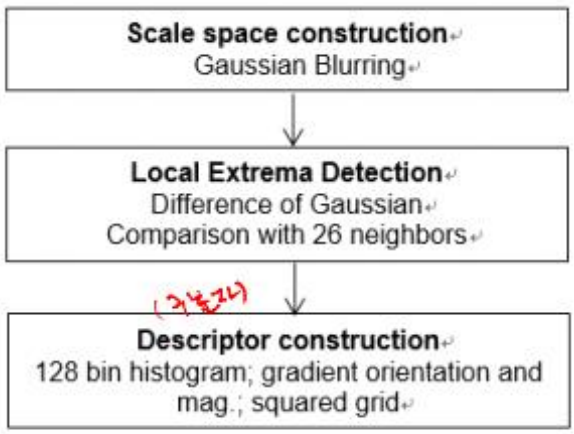


그림 1. SIFT 알고리즘의 도식적 표현

이중 첫 번째 단계인 Gaussain Blurring과 마지막 단계인 Descriptor construction에 대해 자세히 알아보자. 이미지 검색 프로그램 구현에 필요한 주요 함수의 역할은 다음과 같다.

1) **matchTemplate(InputArray image, InputArray templ, OutputArray result, int method)** :

참조 영상 image에서 templ을 method의 방법에 따라 템플릿 매칭을 계산하여 result에 반환한다. 템플릿 매칭은 회전 및 스케일링된 물체에 대해서는 적용이 어려우며 영상의 밝기 등에 덜 민감하도록 정규화 과정이 필요하다.

[templ : image에서 찾고자 하는 작은 영역의 템플릿(자료형은 image와 같다.)]
[result : 결과를 저장할 32비트 실수 행렬]
[method : 비교하는 방법을 저장한다.(CV_TM_SQDIFF, CV_TM_SQDIFF_NORMED)]

2) **minMaxLoc(InputArray src, double* minVal, double* maxVal = 0, Point* minLoc = 0, Point* maxLoc = 0, InputArray mask = noArray())** :

행렬 또는 영상에서 최솟값, 최댓값 그리고 이에 해당하는 위치를 찾을 때 사용한다. 이 함수는 마스크 연산을 지원하므로 행렬 일부 영역에서의 최솟값, 최댓값 또는 해당 위치를 구할 수 있다.

[src : 입력 영상, 단일 채널]
[minVal : 최솟값을 받을 double형 변수의 주소, 필요 없으면 0이 됨]
[maxVal : 최댓값을 받을 double형 변수의 주소, 필요 없으면 0이 됨]
[minLoc : 최솟값 위치 좌표를 받을 Point형 변수의 주소. 필요 없으면 0이 됨]
[maxLoc : 최댓값 위치 좌표를 받을 Point형 변수의 주소. 필요 없으면 0이 됨]
[mask : 마스크 영상. 마스크 영상의 픽셀 값이 0이 아닐때만 연산을 수행함]

3) **normalize(src, dst, alpha, beta, type_flag)** :

인자로 입력된 배열을 정규화하는 함수로 최솟값을 alpha에 맞추고 최댓값을 beta에 맞춘다.

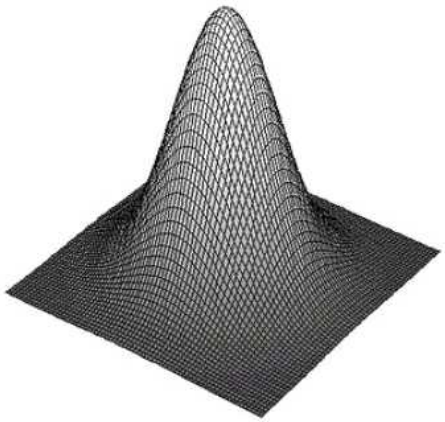
[src : 정규화 이전의 데이터]
[dst : 정규화 이후의 데이터]
[alpha : 정규화 구간1]
[beta : 정규화 구간2]
[type_flag : 정규화 알고리즘 선택 플래그 상수]

4) **_sift_features(IplImage* img, struct feature** feat, int intvls, double sigma, double contr_thr, int curv_thr, int img_dbl, int descr_width, int descr_hist_bins)** :

사용자가 지정한 매개 변수 값을 사용하여 이미지에서 SIFT 특징점을 찾는다. 검출된 모든 특징점은 feature 배열 안에 저장된다.

[img : 특징을 발견할 이미지]
[feat : 탐지된 특징을 저장할 배열의 포인터]
[intvls : 옥타브당 샘플링된 간격의 수]
[sigma : 옥타브의 scale space 표현 전 각 이미지에 gaussian smoothing이 적용된 양]
[contr_thr : scale space function |d(x)| 값에 대한 임계값]
[curv_thr : 과도하게 edge-like한 특징을 버리는데 사용되는 곡선 비율에 대한 임계값]
[img_dbl : scale space를 두 배로 늘리려면 1, 아니라면 0]
[descr_width : 히스토그램의 너비]
[descr_hist_bins : 각 히스토그램에서 특징점 기술자를 계산하는데 사용된 orientation의 수]

5) **GaussianBlur(InputArray src, OutputArray dst, Size ksize, double sigmaX, double sigmaY=0, int borderType=BORDER_DEFAULT)** :



다음 그림과 같이 현재 픽셀에서 가까울수록 더 큰 가중치를 갖고 멀수록 더 작은 가중치를 갖는 커널을 만든다.

[src : 소스 이미지]
[dst : 목표 이미지]
[ksize : 가우시안 커널 사이즈로써 홀수여야 함]
[sigmaX, sigmaY : 가우시안 커널의 표준편자]

6) **filter2D(InputArray src, OutputArray dst, int ddepth, InputArray kernel, Point anchor=Point(-1,-1), double delta=0, int borderType=BORDER_DEFAULT) :**

입력 영상과 커널을 convolution 시킨다.

[src : 입력 이미지]
[dst : 입력 이미지와 크기, 채널 수가 같은 결과 이미지]
[ddepth : 결과 이미지의 깊이]
[kernel : convolution할 커널로 부동 소수점 단일 채널 행렬]
[anchor : 커널의 중심점]
[delta : 필터 적용 후에 픽셀에 더해질 값으로 옵션임]
[borderType : 이미지의 테두리 부분을 처리할 메소드]

7) **feat_detection_data(struct detection_data*) :**

feature's detection data를 리턴한다.

[detection_data : 발견과 관련된 feature 데이터를 담고 있는 구조체]

8) **drawKeyPoints(InputArray image, vector<KeyPoint> &keypoints, InputOutputArray outImage, Scalar &color=Scalar::all(-1), int flags=DrawMatchesFlags::DEFAULT) :**

keypoints를 그려준다.

[imge : 입력 이미지]
[keypoints : 입력 이미지로부터 찾은 keyPoints]

[outImage : 출력 이미지. 출력 이미지의 내용은 플래그의 내용에 따라 달라짐]
[color : keyPoints의 색상]
[flags : features를 그리기 위한 설정값]

9) **cvSmooth(const cvArr* src, cvArr* dst, int smoothtype, int param1, int param2, double param3) :**

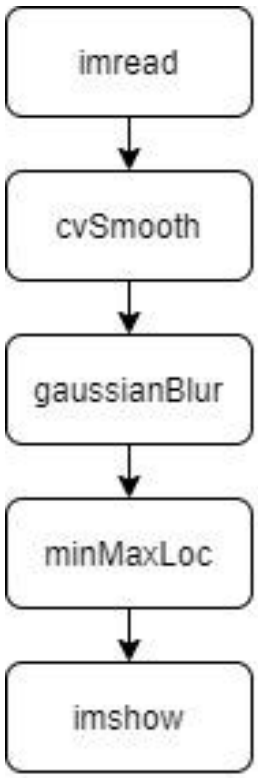
입력 이미지를 주어진 타입에 맞게 blurring해서 dst 이미지에 저장한다.

[src : 원본 이미지]
[dst : 결과 이미지]
[smoothtype : smoothing 타입]
[param : parameter는 smoothing 타입에 의해 아래 표와 같이 결정됨]

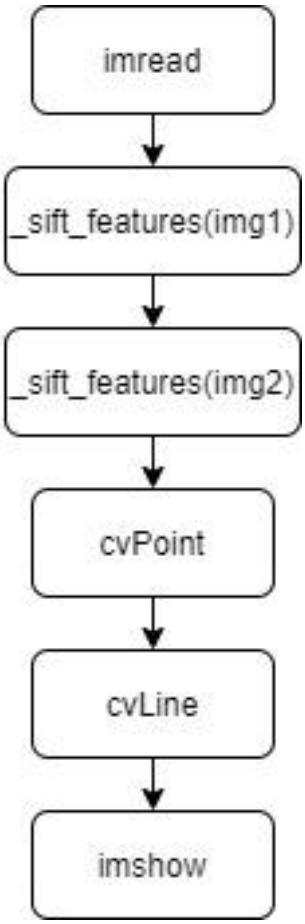
smoothtype 값	이름	바꿔치기	채널 개수	입력 영상 타임	출력 영상 타입	간략한 설명
CV_BLUR	단순 블러	O	1, 3	8u, 32f	8u, 32f	param1 × param2 크기의 이웃 픽셀값의 평균
CV_BLUR_NO_SCANL	스케일링 없는 단순 블러	X	1	8u	8u 입력이면 16s, 32f 입력이면 32f	param1 × param2 크기의 이웃 픽셀값의 합
CV_MEDIAN	중간값 블러	X	1, 3	8u	8u	param1 × param2 크기의 이웃 픽셀값에서 중간값
CV_GAUSSIAN	가우시안 블러	O	1, 3	8u, 32f	8u 입력이면 16s, 32f 입력이면 32f	param1 × param2 크기의 이웃 픽셀값의 가중치 합
CV_BILATERAL	양방향 필터	X	1, 3	8u	8u	3 × 3 크기의 양방향 필터 적용 색상시그마 = param1 공간시그마 = param2

다음은 실습 시간에 진행했던 코드를 기반으로 위에서 작성한 기본 라이브러리 함수를 이용한 주요 함수들의 관계도 플로우 차트이다. 로컬 하드 드라이브에 저장된 이미지를 탐색하면서 일정 수준 이상의 유사도가 나오면 그 이미지에 대한 검색이 성공했다고 할 수 있다.

1) Gaussian Filtering

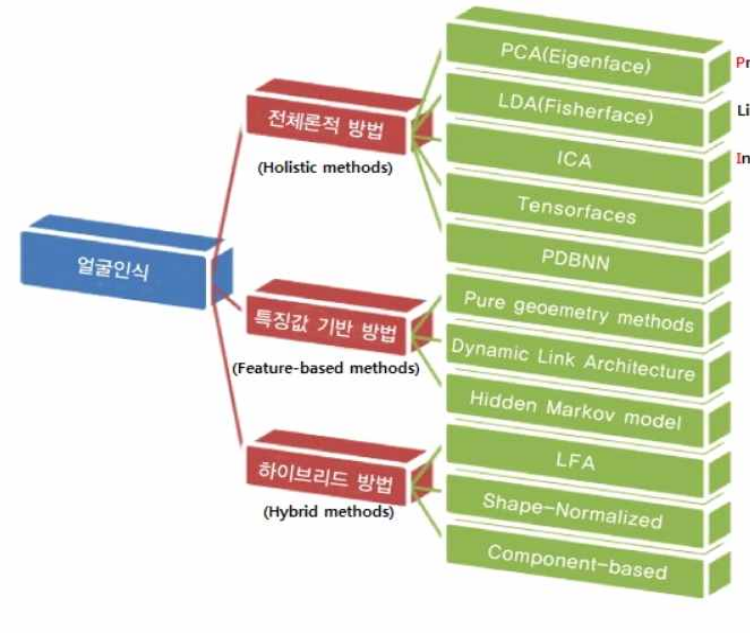


2) SIFT Descriptor Matching



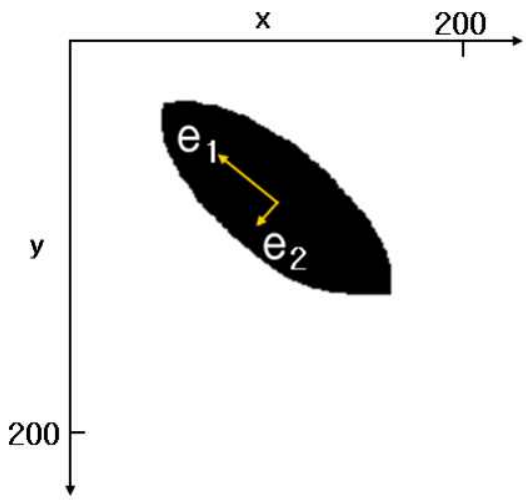
2. 얼굴 인식 프로그램의 구현에 필요한 주요 함수들의 관계도 그래프를 자유로운 형식으로 그리고, 각 함수들의 역할에 대하여 설명하시오. (이미지는 모두 로컬 하드 드라이브에 저장되어 있는 것으로 가정. 함수의 개수는 주요 기능 위주로 5개 이상 10개 이하로 선정할 것)

얼굴인식 방법론은 아래 그림과 같이 크게 3가지로 분류된다. 이중 전체론적 방법의 PCA 방법에 대해 살펴보도록 하자.



PCA(Principal Component Analysis, 주성분 분석) 방법은 분포된 데이터들의 주성분을 찾는 것으로 여러 데이터들이 모여 하나의 분포를 이룰 때 이 분포의 주성분을 분석한다.

이때 주성분이란 그 방향으로 데이터들의 분산이 가장 큰 방향벡터를 의미한다. 오른쪽 그림에서 주성분 벡터는 e1,e2가 될 것이다. 이때 주성분 벡터끼리는 서로 수직이다.



<그림 1> 2D에서의 PCA 예

1) **CVAPI(void) cvCalcEigenObjects(int nObjects, void* input, void* output, int ioFlags, int ioBufSize, void* userData, CvTermCriteria* calcLimit, IplImage* avg, float* eigVals) :**

PCA에서 필요로 하는 평균 영상, EigenVector, EigenValue를 계산한다.

2) **CVAPI(double) cvCalcDecompCoeff(IplImage* obj, IplImage* eigObj, IplImage* avg)** :

계산된 Eigen space에 이미지를 사영한다.

3) **cascadeClassifier(생성자):**
여러 개의 검출기를 사용하여 미리 학습되어 있는 XML 포맷으로 저장된 분류기를 로드한다. 파일 이름을 가지고 해당 분류기 파일을 읽는다.

4) **equalizeHist** : [0,255]사이의 값으로 히스토그램을 균등하게 분포시킨다.

5) **detectMultiScale** :
검출기의 특성에 따라 입력 이미지에서 특정 값을 검출한다. 검출된 객체는 cv::Rect 형 목록으로 반환된다.

7) **subspaceReconstruct** :
새로운 이미지와 기존에 수집된 이미지를 Reconstruct한다. 이렇게 Reconstruct된 이미지와 새로 들어오는 이미지를 비교하고 유사도를 기반으로 예측도를 측정한다.

8) **cvtColor** :
흑백으로 이미지를 변환한다.

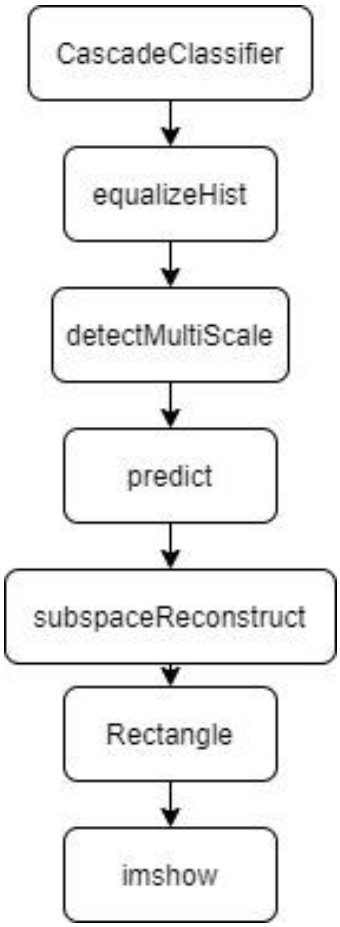
9) **rectangle** :
원본 이미지에 얼굴의 위치를 사각형의 형태로 표시한다.

얼굴 인식 프로그램의 구현에 필요한 주요 함수들의 관계도 그래프를 살펴보기 전에 이에 대해 간단히 설명해보도록 하자. 먼저 얼굴 검출의 단계에서 openCV에서 제공하는 Harr 분류기 XML 파일을 CascadeClassifier 함수를 통해 불러온다.

이후 얼굴 전처리 단계로 넘어가는데, 얼굴 인식은 조명, 방향, 표정 등의 변화에 매우 취약하기 때문에 이러한 차이를 줄여주어야 한다. 이 과정중 하나로 equalizeHist 함수를 사용한다. 얼굴 인식 단계에서는 얼굴을 검출하기 위해 detectMultiScale 함수를 사용하고, 기존의 이미지들과 새로운 이미지를 Reconstruct하기 위해 subspaceReconstruct 함수를 사용한다.

마지막으로 이렇게 검출된 얼굴 주변에 사각형을 그려(Rectangle) 원본 이미지에 저장하고, 이를 보여주는 것(imshow)으로 얼굴 인식 프로그램이 종료된다.

다음은 얼굴 인식 프로그램의 구현에 필요한 주요 함수들의 관계도 플로우차트이다.



3. 주어진 확률 밀도 함수 p(x)를 통계적으로 따르는 stochastic process, X0, X1, X2, ... 를 생성해주는 소프트웨어를 설계한 후, 구현에 필요한 주요 함수들의 관계도 그래프를 자유로운 양식으로 그리고, 각 함수들의 역할에 대하여 설명하시오.

확률 밀도 함수 p(x)를 통계적으로 따르는 stochastic process를 발생시키는 방법은 Inversion이다. 다음은 Inversion 방법을 사용하여 확률 과정을 구하는 것이다.

> $U_0, U_1, U_2, U_3, \cdots$를 $[0,1]$ 구간의 값을 가지는 균등 확률 변수에 대하여 생성한 난수 수열이라고 하자. $F_X(x)$를 주어진 확률 밀도 함수 $p_X(x)$에 대한 누적 분포 함수라 하면, 이 확률 분포를 따르는 난수 수열 값 X_i $(i = 0,1,2,3,\cdots)$는 $X_i = F_X^{-1}(U_i)$와 같이 구할 수 있다.

그러나 복잡한 형태의 누적 분포 함수의 경우 역함수를 찾는 것이 어렵거나 불가능하다. 이를 해결하기 위한 방법은 다음과 같은 비선형 방정식의 근을 찾는 것이다.

$$X_i = F_X^{-1}(U_i) \rightarrow F_X(x) = U_i \rightarrow f(x) \equiv F_X(x) - U_i = 0, \quad x \in [x_0, x_n] \rightarrow X_i$$

비선형 방정식을 풀기 위한 방법에는 여러 가지가 있는데, 이중 실습에서 사용했던 bisection method를 이용하도록 하자. 주요 함수들의 역할은 다음과 같다.

1) **trapezoidal(double* ax, double* ay, int n, double h, double start, double end)** :

x0~x까지 적분한다고 하면 일반적으로 적분의 윗구간은 샘플링 되어 있지 않다. 따라서 x 직전의 샘플링 지점(xm)를 찾아 합성 사다리꼴 공식을 이용해 넓이를 구한다.

[ax, ay : pdf 함수의 x,y 값]
[n : 샘플링 개수]
[h : 구간 간격]
[start, end : 각각 x0와 x를 의미]

2) **bisectionMethod(double *fx, double *fy, int n, double h, double u)** :

초기값 a0=0, b0=1을 설정하여 x1 = (a0+b0)/2의 중간값을 설정한다. 아래에서 설명하는 linearEquation을 이용하여 구간을 줄여나가며 다음 식의 근을 찾는다.

$$X_i = F_X^{-1}(U_i) \rightarrow F_X(x) = U_i \rightarrow f(x) \equiv F_X(x) - U_i = 0, \;\; x \in [x_0, x_n] \rightarrow X_i$$

3) **linearEquation(double* fx, double *fy, int idx, double x1)** :

(xm, f(xm)), (xm+1, f(xm+1))을 지나는 직선의 방정식을 구한다. (선형보간) 이를 이용하여 xm~x까지의 넓이를 구한다.

[fx, fy : cdf 함수의 x,y 값]
[idx : 위의 설명에서 m값을 의미]
[x1 : 구간의 중점의 x 좌표]

다음은 **구현에 필요한 주요 함수들의 관계도 플로우 차트이다.**