소프트웨어공학과

고급프로그래밍 1
기말고사 범위 정리

OpenCV 4 로 배우는 컴퓨터 비전과 머신러닝 - 5~12 장

이시헌

2023-6-12

내용

- 5 장: 영상의 밝기와 명암비 조절 (193p) 2
  - A. 그레이스케일 영상으로 변환 2
  - B. 영상의 밝기 조절 2
  - C. 영상의 명암비 조절 3
  - D. 히스토그램 분석 3
- 6 장: 영상의 산술 및 논리 연산 (229p) 4
- 7 장: 필터링 (243p) 4
  - A. 필터링 연산 4
  - B. 블러링 5
  - C. 샤프닝 5
  - D. 잡음 제거 필터링 6
- 8 장: 영상의 기하학적 변환 (275p) 7
  - A. 어파인 변환 7
  - B. 이동 변환 7
  - C. 전단 변환 8
  - D. 크기 변환. 8
  - E. 회전 변환. 8
  - F. 대칭 변환. 8
  - G. 투시 변환 9
- 9 장: 에지 검출과 응용 (303p) 10
  - B. 마스크 기반 에지 검출 10
  - C. 케니 에지 검출기. 11
  - D. 허프 변환 직선 검출 11
  - E. 허프 변환 원 검출 11
- 10 장: 컬러 영상 처리 11
- 11 장: 이진화와 모폴로지 (357p) 12
  - A. 이진화 12
  - B. 적응형 이진화 12
  - C. 모폴로지 연산 12
  - D. 이진 영상의 열기와 닫기 13
- 12 장: 레이블링과 외곽선 검출 (379p) 13
  - A. 레이블링 13
  - B. 외곽선 검출 14
  - C. 외곽선 처리 함수 14

기말고사 범위 내용 정리. 주의! 모든 예외처리 코드는 생략하겠음. 교재를 보고 예외처리를 하기 바람.

- 5장: 영상의 밝기와 명암비 조절 (193p)

  A. 그레이스케일 영상으로 변환

  i. 읽어올 때 변환: Mat img1 = imread("lenna.bmp", IMREAD_GRAYSCALE);

  ii. 읽어온 후 변환:

```
Mat img3 = imread("lenna.bmp", IMREAD_COLOR);
Mat img;
cvtColor(img3, img4, COLOR_BGR2GRAY);
```

  B. 영상의 밝기 조절

  i. 조절 수식: dst(x, y)=src(x, y)+n //dst: 출력 영상. src: 입력 영상. 모든 픽셀에 n 값을 더함.

  ii. OpenCV에서 연산자 재정의를 할 때 포화연산이 고려됨. (0미만과 255초과의 연산결과를 0과 255로 만들어 출력함.) 즉, 아래처럼 산술연산 하듯이 더하고 빼도 정상적인 영상이 출력됨.

  iii. 연산 예시: Mat dst = src + 100; //Mat src = imread("lenna.bmp", IMREAD_GRAYSCALE);

  iv. 다른 방법으로, 모든 픽셀에 대하여 수동으로 밝기를 지정할 수 있음.

```
Mat dst(src.rows, src.cols, src.type());    //입력 영상과 동일한 크기와 타입으로 미리 설정해야 함.
for(int j = 0; j < src.rows; j++){
    for(int i = 0; i < src.cols; i++){
        Int v = src.at<uchar>(j, i) + 100; //이 경우 포화연산이 고려되지 않아서 다음 행같은 처리 필요.
        dst.at<uchar>(j, i) = v > 255 ? : v < 0 ? 0 : v;
        //src 영상의 모든 픽셀에 접근하여 100 값을 더하여 dst 영상에 저장.
        //변수 v에 100값이 더해진 픽셀값을 담아두고, 3항연산자를 통하여 포화연산을 처리한다.
    }
}
```

  v. 다만 위의 모든 방법보다는 saturate_cast( )라는 캐스팅 함수를 사용하는게 좋다. 이 함수는 포화연산을 고려하여 밝기 조정 연산을 하는 함수이다. (탬플릿 함수이므로 자료형을 명시해야 한다.)

```
for(int j = 0; j <src.rows; j++){
    for(int i = 0; i < src.cols; i++){
        dst.at<uchar>(j, i) = saturate_cast<uchar>(src.at<uchar>(j, i) + 100);
    }
}
```

  vi. 트랙바의 이벤트 핸들러에서 적당한 방법을 택하여 밝기 조절을 구현할 수 있다 (206p 참고)

C. 영상의 명암비 조절

i. 조절 수식: dst(x, y) = saturate(s*src(x, y)); //영상에 s를 곱연산 함. 밝은 픽셀이 더 크게 밝아진다.

ii. 조절 예시: 곱셈 연산자도 포화연산을 고려하여 재정의 되어 있다.

```
Mat src = imread("lenna.bmp", IMREAD_GRAYSCALE);
float s = 2.f;
Mat dst = s * src;      //dst에 s가 곱해진 영상이 대입됨. 255를 초과하는 모든 결과는 255로 처리됨.
```

iii. 하지만 위와 같이 일괄적으로 명암을 조절하면 좋지 않다. 다음과 같이 특정한 기준을 중심으로 명암을 양방향으로 조절하는게 좋다.

```
float alpha = 1.f;
mat dst = src + (src – 128) * alpha;      //영상 픽셀의 128 밝기를 기준으로 alpha값을 곱한다.
//128 이하의 밝기는 어두워지고, 128 이상의 밝기는 더 밝아진다. (직선의방정식의 기울기 변경)
```

D. 히스토그램 분석

i. 히스토그램은, 밝기별로 픽셀이 얼마나 있는지를 그래프로 표현한 것.(픽셀 값 분포도)

ii. calcHist( ) 함수를 사용하여 히스토그램을 구할 수 있다. (215p 참고)

```
CV_Assert(img.type() == CV_8UC1); //영상이 그레이스케일인지 검사함. 조건이 거짓이면 경고 출력함.

Mat hist;                                  //출력 히스토그램을 저장할 영상 선언.
int channels[ ] = { 0 };                   //히스토그램을 구할 채널을 지정하는 배열 선언.
int dims = 1;                              //출력 히스토그램의 차원 수.
const int histSize[ ] = { 256 };           //각 차원의 히스토그램 배열 크기를 나타내는 배열.
float graylevel[ ] = { 0, 256 };           //각 차원의 히스토그램 범위를 지정하기 위한 배열.
const float* ranges[ ] = { graylevel };    //위에서 정의한 배열로 함수에 넘겨줄 const 배열 정의.

calcHist(&img, 1, channels, noArray(), hist, dims, histSize, ranges); //더 자세한건 필요시 215p로 갈 것.
```

iii. 히스토그램을 직접 구하는 방법은 생략하겠다. (219p 참고)

iv. 히스토그램을 화면에 그리는 방법! 위에서 구한 hist 객체의 값을 반복문으로 읽어서 line함수에 넘긴다.

```
Double histMax;
minMaxLoc(hist, 0, &histMax);                    //hist 행렬 원소의 최댓값을 histMax에 저장함.

Mat imgHist(100, 256, CV_8UC1, Scalar(255));     //100x256 크기, 1채널, 흰색 영상 생성.
for(int i=0; I < 256; i++){
    Line(imgHist, Point(I, 100), Point(I, 100 – cvRound(hist.at<float>(i, 0)*100/histMax)), Scalar(0));
}  //반복문으로 hist 행렬을 읽어서, 백분율로 변환하여 세로 100px 공간에 선분을 끝까지 그린다.
```

v. 히스토그램 스트레칭: 히스토그램을 늘려서 가능한 밝기 범위에 골고루 분포하도록 만드는 것. 스트레칭 후에는 영상의 명암비가 높아진다. (선명해진다)

```
double gmin, gmax;
minMaxLoc(src, &gmin, &gmax);

Mat dst = (src – gmin) * 255 / (gmin – gmax);
// 영상에 대한 산술연산자 재정의가 잘 되어 있어서 이런 방법으로 히스토그램을 늘릴 수 있다.
```

vi. 히스토그램 평활화: 영상의 픽셀 값 분포가 그레이스케일 전체 영역에서 골고루 나타나도록 변경하는 알고리즘의 하나.

vii. 알고리즘 소개는 넘어가겠다. equalizeHist( ) 함수를 사용하면 평활화를 할 수 있다.

```
Mat dst;
equalizeHist(src, dst); // dst 객체에 평활화가 수행된 영상이 저장된다.
```

E. 5장이 끝났다.

- 6장: 영상의 산술 및 논리 연산 (229p)

A. 6장은 읽어보라고만 하셨으니, 개념만 요약하여 적어두겠다.

B. 영상은 2차원 행렬이므로, 산술연산이 가능하다. 두 영상을 더하면, 각 행렬 원소의 값이 더해진다. 영상에는 상수를 곱할 수도 있고, 영상끼리 뺄셈도 가능하며, 별게 다 된다.

C. 영상은 2차원 행렬이므로... 논리 연산도 당연히 된다. 영상의 각 픽셀에 대하여 논리 연산을 수행할 수 있다.

- 7장: 필터링 (243p)

A. 필터링 연산: 마스크(mask) 행렬(커널이라고도 함)을 영상의 모든 픽셀에 합성곱 하는 연산이다. 마스크 행렬의 형상에 따라서, 다른 패턴을 영상에서 검출해낼 수 있다.

i. 필터링 연산을 하면, 영상의 외곽의 일부가 소실된다. 이때 출력 영상의 크기가 작아지는 것을 막기 위하여 원본 영상의 가장자리를 확장하여 연산을 수행하는데, 이 확장 방법은 열거형 상수로 정의되어 있다. (247p)

ii. 일반적으로 filter2D( )함수를 사용하여 필터링 연산을 수행한다. 엠보싱 필터링을 예시로 들겠다. (250p)

```
Mat src = imread("rose.bmp", IMREAD_GRAYSCALE);

float data[] = { -1, -1, 0, -1, 0, 1, 0, 1, 1 };  //엠보싱 필터 마스크로 data 배열을 초기화 했다.
Mat emboss(3, 3, CV_32FC1, data);  //data 배열의 값으로 3X3의 엠보싱 필터 마스크 행렬을 만들었다.

Mat dst;
filter2D(src, dst, -1, emboss, Point(-1, -1), 128);  //src에 emboss로 한 필터링 결과를 dst에 저장.
//Point(-1, -1)은 커널 중심을 연산의 고정점으로 지정하도록 함. 128은 연산 결과에 128을 더한다는 것.
```

B. 블러링: 영상을 뭉개서 부드럽게 만드는 연산.

i. 평균값 필터: 마스크의 크기 내부의 모든 픽셀의 평균을 산출하도록 하는 필터이다. 말 그대로 평균값을 출력하여 영상을 뭉갠다. blur( ) 함수를 사용하여 평균값 필터링을 수행한다. (253p)

```
Mat dst;
blur(src, dst, Size(5, 5)); //마스크 크기만 지정하면 된다. 이 코드는 5X5 크기를 지정한 것.
//마스크 크기가 커질수록 더 넓은 범위를 평균내기에, 영상이 더 흐려진다.
```

ii. 가우시안 필터: 가우시안 분포에 근사하여 생성한 필터 마스크. 정규분포의 표준편차를 달리 하여 블러링 정도를 정할 수 있다. (256p 참고)

```
Mat dst;
int sigma = 1;   //표준편차를 저장한다. 표준편차가 커질수록 영상이 더 흐려진다.
GaussianBlur(src, dst, Size( ), (double)sigma);
//지정한 표준편차에 따라서 가우시안 필터를 생성하고, 블러링 연산을 한 결과를 dst에 저장한다.
//Size( )는 위와 같이 비워 둘 경우, 표준편차에 따라서 자동으로 커널(마스크) 크기를 지정해 준다.
```

C. 샤프닝: 영상을 날카롭게 만드는 연산.

i. 언샤프 마스크 필터: 영상의 에지 근방에서 픽셀 값의 명암비가 커지도록 하는 필터. 이 필터 적용 이전에 영상을 블러링 시켜야 하기 때문에, '언샤프' 마스크 필터라 부른다. (블러링 후 샤프닝을 수행해야 함)

```
Mat blurred;                                  //블러처리 된 영상을 저장할 객체 선언.
GaussianBlur(src, blurred, Size( ), sigma);   //가우시안 블러링 수행하여 blurred 객체에 저장.

float alpha = 1.f;
Mat dst = (1+alpha) * src – alpha * blurred; // 수식에 따라서 언샤프 필터링 수행. 수식은 263p 참고.
```

D. 잡음 제거 필터링

i. 잡음 모델: 잡음이 생성되는 방식. 가우시안 잡음 모델이 가장 일반적이다.

ii. 잡음은 randn( )함수와, add( )함수로 더할 수 있다.

```
Mat noise(src.size( ), CV_32SC1);      //난수를 저장할 영상 객체 선언. 크기와 타입을 지정해둠.
randn(noise, 0, stddev);               //가우시안 분포를 따르는 난수를 noise에 채움.
Mat dst;
add(src, noise, dst, Mat( ), CV_8U);   //src에 noise행렬을 더한 영상을 dst에 저장. Mat( )은 230p 참고.
```

iii. 양방향 필터: 에지 정보를 유지하면서 가우시안 잡음을 제거하기 위한 필터. bilateralFilter( )함수로 양방향 필터 연산을 수행할 수 있다.

```
위에서 노이즈를 더한 dst 영상에서 노이즈를 제거해 보겠다.
Mat dst2;
bilateralFilter(dst, dst2, -1, 10, 5);  //노이즈가 있는 dst에 양방향 필터링을 수행하여 dst2에 저장함.
//-1은 270p 참고, 10은 색 공간의 표준편차, 5는 좌표 공간의 표준편차.
```

iv. 미디언 필터: 특정 지점의 주변 픽셀 값 중에서 중간값을 찾아 영상의 픽셀 값으로 하는 필터링 기법. 잡음 픽셀 값이 주변 픽셀과 큰 차이가 있을 때 효과적으로 동작. 소금&후추 잡음(0 또는 255의 잡음)을 잘 제거함. medianBlur( )함수를 사용하여 미디언 필터링 수행 가능.

```
medianBlur(src, dst, 3);  //마지막 인수인 3은 마스크 행렬의 크기다. 지금은 3X3.
```

E. 7장 끝!

- 8장: 영상의 기하학적 변환 (275p)

A. 어파인 변환: 영상을 구성하는 픽셀의 배치 구조를 변경하여 영상의 모양을 바꾸는 변환. 가역적인 반응이다. 즉, 어파인 함수에 의하여 변환 결과를 산출하기에 가역적이다. (어파인 함수와 행렬의 모습은 교재 276~278p 참고)

i. 어파인 변환은 영상의 픽셀을 좌표로 취급하여, x좌표에 대한 변환 함수와, y좌표에 대한 변환 함수로 변환된 영상의 픽셀 위치를 구한다. 이 두 함수를 행렬로 만들면, 여섯 개의 파라미터를 가진 2x3크기의 어파인 변환 행렬이 만들어지고, 이 변환 행렬을 통해서 어파인 변환을 수행한다.

ii. 어파인 변환행렬을 구하는 함수는 getAffineTransform( ) 이며, 어파인 변환 행렬로 어파인 변환을 하는 함수는 warpAffine( )이다. (280p 예제 참고)

```
Mat src = imread("tekapo.bmp");

Point2f srcPts[3], dstPts[3];  //입력 영상과 출력 영상의 세 점 좌표를 저장할 두 Point클래스 배열 선언.
srcPts[0] = Point2f(0, 0);                          //입력 영상의 좌측 상단.
srcPts[1] = Point2f(src.cols - 1, 0);               //입력 영상의 우측 상단.
srcPts[2] = Point2f(src.cols - 1, src.rows - 1);    //입력 영상의 우측 하단.
dstPts[0] = Point2f(50, 50);                        //출력 - 좌측 상단.
dstPts[1] = Point2f(src.cols - 100, 100);           //출력 – 우측 상단.
dstPts[2] = Point2f(src.cols - 50, src.rows - 50);  //출력 – 우측 하단.

Mat M = getAffineTransform(srcPts, dstPts);
//위에서 입력한 어파인 변환을 수행 가능한 어파인 변환 행렬을 M에 저장.

Mat dst;
warpAffine(src, dst, M, Size( ));  //어파인 변환 행렬을 이용하여 src를 어파인 변환하여 dst에 저장.
```

iii. 어파인 변환 행렬을 갖고 있는 상태에서, 변환 결과 영상의 각 점이 어디로 이동하는지 알고싶다면, openCV에서 지원하는 transform( ) 함수를 사용하면 된다. (282p 참고)

B. 이동 변환: 영상을 가로 또는 세로 방향으로 일정 크기만큼 이동시키는 연산. 시프트 연산이라고도 한다. 이때 이동변환을 하도록 하는 어파인 변환 행렬은 단순한 평행이동이므로, 행렬의 원소는 다음과 같이 쓸 수 있다. (283p 참고)

i. 이동변환의 어파인 행렬: {1, 0, a, 0, 1, b} //영상을 x방향으로 a만큼, y방향으로 b만큼 이동시키는 행렬임.

```
Mat M = Mat_<double>({2, 3}, {1, 0, 150, 0, 1, 100});  //x방향으로 150px, y방향으로 100px.

Mat dst;
warpAffine(src, dst, M, Size( ));  //어파인 행렬 M에 따라서 어파인 변환 수행하여 dst에 저장.
```

C. 전단 변환: 직사각형 형태의 영상을 한쪽 방향으로 밀어서 평행사변형 모양으로 변형시키는 변환.

i. 전단 변환을 수행하는 어파인 변환 행렬은 두가지이다. (285p 참고)

ii. 가로방향 전단변환 어파인 행렬: {1, mx, 0, 0, 1, 0} //mx는 가로방향 밀림 정도. (영상 상단 모서리 유지)

iii. 세로방향 전단변환 어파인 행렬: {1, 0, 0, my, 1, 0} //my는 세로방향 밀림 정도. (영상 좌측 모서리 유지)

iv. 위의 행렬을 warpAffine( )에 넣으면 전단변환 된 영상을 얻을 수 있다.

D. 크기 변환: 영상의 전체적인 크기를 확대 또는 축소하는 변환.

i. 크기변환을 수행하는 어파인 변환 행렬: {sx, 0, 0, 0, sy, 0} //sx는 가로방향 변환비율, sy는 세로방향.

ii. 크기변환 수행 함수: resize( ) //크기변환 수행이 빈번하므로, 자동으로 변환하는 함수를 제공한다.

```
Mat dst, dst1;
resize(src, dst, Size( ), 4, 4, INTER_NEAREST);  //4와 4는 각각 가로 세로를 4배씩 확대한다.
//마지막 인자는 보간법 지정 상수이다. 보간법 지정용 열겨형은 289p 참고.
resize(src, dst1, Size(1920, 1080));              //이 경우 영상을 1920x1080으로 확대한다.
```

E. 회전 변환: 특정 좌표를 중심으로 영상을 원하는 각도만큼 회전시키는 변환.

i. 회전변환도 빈번하게 사용하므로, openCV는 영상의 회전을 위한 어파인 변환 행렬을 생성하는 함수 getRotationMatrix2D( )를 제공한다. (293p 예제 참고)

```
Point2f cp(src.cols / 2.f, src.rows / 2.f);          //영상의 중심 좌표를 cp객체에 저장함.
Mat M = getRotationMatrix2D(cp, 20, 1);   //cp좌표를 중심으로 20도를 반시계방향으로 회전시키도록
하는 어파인 변환 행렬을 리턴하여 M에 저장함. (마지막 인자가 음수면, 시계방향 회전함)

Mat dst;
warpAffine(src, dst, M, Size( ));                    //어파인 변환 수행.
```

F. 대칭 변환: 영상의 상하 혹은 좌우를 바꾸는 변환.

i. 영상의 대칭 변환 수행 함수: flip(src, dst, flipCode) //flipCode가 양수: 좌우대칭, 음수: 상하좌우, 0: 상하.

G. 투시 변환: 직사각형 형태의 영상을 임의의 볼록 사각형 형태로 변환.

i. 어파인 변환은 3개의 점을 기준으로 2X3의 어파인 변환 행렬을 만들어 수행한다. 이와 달리 투시 변환은 4개의 점을 기준으로 8개의 방정식을 얻어서 3X3의 어파인 변환 행렬을 만들어 변환을 수행한다. (298p 참고)

ii. 3X3 투시 변환 행렬 계산 함수: getPerspectiveTransform( )

iii. 3X3 어파인 행렬로 영상을 변환하는 함수: warpPerspective( )

```
Point2f srcQuad[4], dstQuad[4];    //srcQuad에는 src 영상에서 찍은 4개의 점 좌표가 저장되어 있다.
int 2 = 200, h = 300;
dstQuad[0] = Point2f(0, 0);
dstQuad[0] = Point2f(w -1, 0);
dstQuad[0] = Point2f(w -1, h -1);
dstQuad[0] = Point2f(0, h -1);          //dst 영상에 표현될 변환된 영상의 네 꼭지점 저장.

Mat pers = getPerspectiveTransform(srcQurd, dstQurd);  //어파인 변환 행렬 반환. (영상에서 만든 사면체를 dstQuad의 모양으로 변환하는 어파인 변환 행렬 반환)

Mat dst;
warpPerspective(src, dst, pers, size(w, h));                    //어파인 변환 수행.
```

H. 8장 끝!

- 9장: 에지 검출과 응용 (303p)

A. 에지 검출: 영상에서 한쪽 방향으로 픽셀 값이 급격하게 바뀌는 부분을 검출하는 것. 픽셀 값의 변화율을 측정하는 것으로 검출해낸다. 이를 위해서는 영상의 x좌표들과 y좌표들에 대하여 각각 픽셀 값을 편미분 해야 한다. (미분된 함수는 변화율을 나타내고, 영상은 2차원이므로 각각의 축에 대하여 편미분이 요구됨.)

i. 에지는, 미분 근사 마스크를 통하여 필터링 연산을 수행해서 얻는다. (근사 마스크는 307p 참고)

ii. 그래디언트: 2차원 공간에서 정의된 2변수 함수의 x축 방향 미분과 y축 방향 미분을 한꺼번에 벡터로 표현한 것. (벡터이므로 크기와 방향이 있다)

iii. 임계값: 에지 여부를 판단하기 위하여 기준으로 삼는 값. 임계값 이하의 그래디언트 크기는 무시됨.

iv. 그래디언트와 임계값에 대한 자세한 내용은 308~309p 참고.

B. 마스크 기반 에지 검출

i. 앞서 소개한 미분 근사 마스크(1x3, 3x1)는 잡음에 효과적으로 대응할 수 없음... 그래서 더 큰 크기의 적당한 미분 근사 마스크로, 소벨 필터(Sobel filter) 마스크가 잘 쓰이고 있다. 또한 소벨 마스크로 영상을 미분하는 Sobel( ) 함수가 제공된다. (310p 참고)

ii. 다른 잘 쓰이는 필터 마스크로 샤르 필터(Scharr filter) 마스크도 있다. 또한 샤르 필터 마스크로 미분을 수행하는 scharr( ) 함수도 제공된다. (311p 참고)

```
Mat src = imread("lenna.bmp", IMREAD_GRAYSCALE);

Mat dx, dy;
Sobel(src, dx, CV_32FC1, 1, 0);       //x축 방향으로 편미분 수행하여 dx에 저장.
Sobel(src, dy, CV_32FC1, 0, 1);       //y축 방향으로 편미분 수행하여 dy에 저장.

Mat fmag, mag;
magnitude(dx, dy, fmag);              //dx, dy로부터 그래디언트의 크기를 구하여 fmag행렬에 저장.
fmag.convertTo(mag, CV_8UC1);    //실수형 행렬 fmag를 그레이스케일 형식으로 변환하여 mag에 저장.

Mat edge = mag > 150;                 //mag: 미분된 결과물. edge: 150을 임계값으로 이진화된 결과물.
//연산자 재정의로 150을 임계값으로 하여 mag에 이진화를 수행하여 edge에 저장.
```

C. 케니 에지 검출기: 아주 잘 쓰이는 에지 검출방법. 소벨이나 샤르 마스크보다 더 정확하다.

i. 케니 에지 검출기는 4단계의 연산 과정을 거친다.

가우시안 필터링: 잡음 제거 **그래디언트 계산**: 소벨 마스크를 통해 가로방향과 세로방향 필터링 수행 후 그래디언트 계산 **비최대 억제**: 그래디언트가 국지적 최대인 픽셀만을 에지 픽셀로 설정하여, 두께를 한 픽셀로 억제. **이중 임계값 적용**: 두 임계값으로 강한 에지와 약한 에지 구간 형성. 강한 에지와 연결되어 있는 에지만을 최종적으로 에지로 판단)

ii. 위의 모든 과정을 자동으로 처리해주는 Canny( ) 함수를 사용하면 케니 에지 검출을 할 수 있다.

```
Mat dst1, dst2;
Canny(src, dst1, 50, 100);    //마지막의 두 인자는 각각이 낮은 임계값 / 높은 임계값이다.
Canny(src, dst2, 50, 150);    //높은 임계값을 더 높이면, 검출되는 에지의 양이 감소한다.
```

D. 허프 변환 직선 검출: 허프 변환이란 기법을 통해서, 영상에서 직선을 검출하는 것.

i. 허프 변환의 원리: 직선의 방정식에 존재하는 두 파라미터(계수)를 미지수로, 본래의 미지수를 파라미터로 전환하여, xy공간에서 에지로 검출된 여러 점 각각으로부터 다수의 직선을 얻어내어 ab파라미터 공간에 그릴 수 있다. (y=ax+b -> b=-xa+y 로 전환)

ii. 그리고 그 직선들이 가장 많이 교차하는 지점의 (a, b)값을 갖는 직선의 방정식이, 영상 속의 에지로부터 검출된 직선의 방정식 인 것이다. 자세한 그림과 설명은 321p에서 보기를 바란다...

iii. 허프 변환 직선 검출은 HoughLines( )함수로 할 수 있다. 이 함수는 직선을 점과 각도로 vector객체에 저장한다. (영상의 최외곽까지 이어지는 직선을 저장한다)

```
Mat edge;
Canny(src, edge, 50, 150);                    //에지 검출을 수행하여 edge객체에 저장.

vector<Vec2f> lines;                          //검출한 직선을 저장할 vector객체 lines 선언.
HoughLines(edge, lines, 1, CV_PI / 180, 250); //직선 검출. 각 인자에 대한 설명은 323p 함수 원형 참고.
```

iv. 다른 방법으로, 확률적 허프 변환 방법은 HoughLinesP( )함수로 적용할 수 있다. 이 함수는 HoughLines( )와 달리, 직선의 양 끝점을 vector객체에 저장한다. 즉, 영상 끝까지 이어지는 직선이 아니라, 선분이 검출된다. 자세한 사용 예시는 327p의 예제를 참고할 것.

E. 허프 변환 원 검출: 거의 사용하지 않으므로, 넘어가겠다.

F. 9장 끝!

- 10장: 컬러 영상 처리 (333p) – 안배우고 넘어갔다. 12주차 강의자료에는 있지만... 넘어갔다. 이유는 불명.

- 11장: 이진화와 모폴로지 (357p)

A. 이진화: 영상이 이진 값을 갖도록 만드는 것. 비전 분야에서는 임계값 이하는 0으로, 임계값 이상을 255로 설정한다. (검정, 흰색) 이진화는 threshold( )함수로 수행할 수 있다.

```
Mat src = imread("camera.bmp", IMREAD_GRAYSCALE);

Mat dst;
threshold(src, dst, 128, 255, THRESH_BINARY);  //임계값 128, 최댓값 255로 이진화 수행. 여기서의 최댓값은 마지막 인자의 종류에 따라서 기능이 달라질 수 있다.
//마지막 인자는 임계값 종류를 결정하는 열거형 상수이다. 361p를 참고하여 선택할 것.
```

i. 이때 이진화는 영상 전체에 적용된 것으로, "전역 이진화"라고 부른다.

B. 적응형 이진화: 전역 이진화와 달리, 영상의 각 픽셀마다 다른 임계값을 적용하여 이진화를 하는 것. 적응형 이진화는 픽셀의 인근 픽셀들이 갖는 값의 평균값을 임계값으로 하여 수행된다. 이때 평균은 산술평균도, 가중평균도 적용 가능하다. 그리하여 이런 적응형 이진화는, adaptiveThreshold( )함수로 수행할 수 있다.

```
Mat dst;
Int bsize = 11;  //블록은 11X11크기이다. (목표 픽셀을 중심으로 11X11)
adaptiveThreshold(src, dst, 255, ADAPTIVE_THRESH_GAUSSIAN_C, THRESH_BINARY, bsize, 5);
//255 : 최댓값
//ADAPTIVE_~ : 블록 평균 계산 방법 지정(산술/가중). 현재는 가우시안 가중평균. (이것이 임계값)
//THRESH_BINARY : 이진화 임계값 종류 결정. (361p 참고)
//bsize : 평균을 구할 블록의 크기 지정. 크기가 클수록 더 많은 픽셀로부터 평균을 구함.
//5 : 블록 평균에서 마지막 인수를 뺀 값을 임계값으로 한다. 지금은 5를 뺀다.
```

C. 모폴로지 연산: 객체의 형태 및 구조에 대하여 분석하고 처리하는 연산. (모폴로지: 형태 또는 모양에 관한 학문)

i. 침식 연산: 객체의 외곽을 골고루 깍아 내는 연산. 이는 erode( )함수로 수행한다. (372p 참고)

ii. 팽창 연산: 객체의 외곽을 골고루 확장시키는 연산. 이는 dilate( )함수로 수행한다. (372p 참고)

iii. 위의 두 연산은 마스크 연산이다. 이때 적당한 구조 요소를 정의하여 마스크로 사용해야 하는데, 그 구조 요소 행렬을 만드는 함수가 제공된다. getStructuringElement( )이다. 371p의 함수 원형을 참고하길.

```
Mat bin;   //이진화된 영상을 잠시 담아둘 객체 선언.
threshold(src, bin, 0, 255, THRESH_BINARY | THRESH_OTSU);  //이진화 수행.

Mat dst1, dst2;     //각 연산의 결과물을 담을 객체 선언.
erode(bin, dst1, Mat( ));    //침식 연산 수행.
dilate(bin, dst2, Mat( ));    //팽창 연산 수행.
```

D. 이진 영상의 열기와 닫기

i. 열기 연산: 침식연산 수행 -> 팽창연산 수행 //작은 객체를 지운 후, 크기 복원.

ii. 닫기 연산: 팽창연산 수행 -> 침식연산 수행 //객체 내부의 작은 구멍을 메운 후, 크기 복원.

iii. 모폴로지 열기와 닫기 연산은 morphologyEx( )함수로 수행.

```
Mat bin;
Threshold(src, bin, 0, 255, THRESH_BINARY | THRESH_OTSU);  //이진화 하여 bin에 저장.

Mat dst1, dst2;
morphologyEx(bin, dst1, MORPH_OPEN, Mat( ));  //열기연산.
morphologyEx(bin, dst2, MORPH_CLOSE, Mat( ));  //닫기연산.
```

E. 11장 끝!

- 12장: 레이블링과 외곽선 검출 (379p)

A. 레이블링: 영상 내의 객체에 번호를 붙이는(객체를 구분하고 분석하는) 작업. 객체 인식을 위한 전처리 과정이다. 연결된 픽셀들에 대하여 동일한 번호를 부여하여 레이블 맵에 저장한다.

i. 레이블링은 connectedComponents( )함수로 수행 (382p 참고)

```
Uchar data[ ] = {
    0, 0, 1, 1, 0, 0, 0, 0,
    1, 1, 1, 1, 0, 0, 1, 0,
    1, 1, 1, 1, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 1, 1, 0,
    0, 0, 0, 1, 1, 1, 1, 0,
    0, 0, 0, 1, 0, 0, 1, 0,
    0, 0, 1, 1, 1, 1, 1, 0,
    0, 0, 0, 0, 0, 0, 0, 0,
};  //0과 1로 이진화된 원본 영상.
Mat src = Mat(8, 8, CV_8UC1, data) * 255;
//임시 객체(8X8크기, 그레이스케일, data행렬을 데이터로 함)에 255를 곱연산 한 영상을 src에 저장한다.

Mat labels;
int cnt = connectedComponents(src, labels);  //labels객체에 레이블링이 된 행렬 저장.
```

ii. 위의 원본 영상은 왼쪽 상단부터 각각의 객체의 값이 1, 2, 3의 번호로 교체되어 labels에 저장된다. (상상이 잘 안된다면, 383p 예제 실행결과 참고)

iii. connectedComponentsWithStats( )함수를 사용하여, 레이블 맵과 각 객체 영역의 통계 정보를 한번에 얻을 수 있다. (일일히 for문으로 픽셀에 접근하여 통계 정보를 구하기 힘들기에... openCV에서 함수 제공)

iv. 자세한 사항은 384p의 함수 원형과, 386p의 예제 코드 12-2를 참고할 것.

B. 외곽선 검출: 객체 영역 픽셀 중, 배경 영역과 인접한 픽셀을 검출하는 것. 일반적으로 이진화 된 영상에서 외곽선을 검출한다. 필요하다면 모폴로지 이후의 영상에서 외곽선을 검출해야 할 수 있다. (원본 영상 – 그레이스케일 변환 – 이진화 – 모폴로지 – 레이블링 의 순서대로, 현재까지 배운 내용이 이어지는 것이다. 객체를 검출하고 객체를 구분하는 것.)

i. 객체의 외곽선 정보는 " vector<vector<Point>> contours; " 이와 같은 형식의 변수에 저장한다. 이는 각각의 객체가 갖는 외곽선 픽셀 좌표들을 담은 vector들을 vector에 저장한 것이다. 2차원 배열과 유사한 느낌.

ii. 외곽선 정보가 vector에 저장되므로, 일반 배열처럼 인덱스를 붙여 특정 객체의 외곽선 픽셀 정보에 접근할 수 있다. (contours[2] : 3번째 객체의 외곽선을 구성하는 픽셀 데이터들이 들어있는 vector)

iii. 영상 내부 객체들의 외곽선 검출 함수: findContours( ) //389p 함수 원형 참고.

iv. 위의 함수로 검출한 외곽선을 영상에 그리는 함수: drawContours( ) //392p 원형 참고.

```
Mat src = imread("contours.bmp", IMREAD_GRAYSCALE);

vector<vector<Point>> contours;    //외곽선 정보 저장할 vector객체 선언.
findContours(src, contours, RETR_LIST, CHAIN_APPROX_NONE);  //외곽선 검출 후 저장.
//RETR_LIST : 객체 바깥과 안쪽 외곽선 모두 검색 & 계층 구조 없음.
//CHAIN_APPROX_NONE : 모든 외곽선 점들의 좌표 저장.
//다른 열거형 상수들은 390p 참고.

Mat dst;
cvtColor(src, dst, COLOR_GRAY2BGR);  //트루컬러 영상으로 변환하여 dst에 저장.

for(int i=0; i<contours.size( ); i++){  //모든 외곽선에 순차적으로 접근.
    Scalar c(rand( ) &255, rand( ) &255, rand( ) &255);  //1회 반복될 때 마다 무작위 색상 지정.
    drawContours(dst, contours, i, c, 2);  //외곽선 그리는 함수.
    //dst영상에 contours객체에서, i번째 외곽선을, c의 색상으로, 2의 두께로, 외곽선을 그린다!
}
```

v. 외곽선에 계층구조를 적용한다면, findContours( )의 열거형 상수를 바꾸고, 계층구조를 저장할 변수를 만들어줘야 한다. (395p 예제를 참고하고, 필요하다면 390p의 열거형 상수를 직접 볼 것)

vi. 계층구조 저장 변수 선언 형식 : vector<Vec4i> hierarchy;

vii. 계층구조도 drawContours( )에 인자로 넘길 수 있다.

C. 외곽선 처리 함수: 외곽선을 검출한 이후, 외곽선 좌표 정보를 이용하는 다양한 함수들

i. 다양한 함수가 있다. 함수 원형과 예제만 보면 바로 사용 가능할, 이해가 필요치 않은 함수들이므로 자세한 설명은 생략하겠다.

ii. 중요한 것은, 대부분의 함수가 " vector<Point> "(입력 점들의 집합)의 형태로 외곽선 정보를 받는다는 것이다.

iii. 실제 사용 예는 399p의 에제코드 12-5에서 보라.

D. 12장 끝!