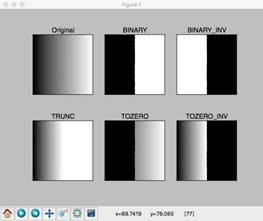
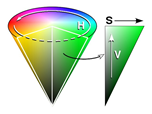
1. OpenCV#1(Binarization,Morphology,Arithmetic )

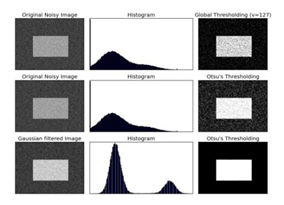
**1. 이치화(Binarization)**  
OpenCV에서 아래와 같이 네가지 API로 이치화를 할수 있습니다.  
이 외에도 직접 알고리즘을 작성하여 사용하는 경우도 많이 있습니다.  
   
**1)     기본임계처리1 : cvThreshold()**

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221341480823&categoryNo=10&parentCategoryNo=0&viewDate=&currentPage=1&postListTopCurrentPage=&from=postList&userTopListOpen=true&userTopListCount=5&userTopListManageOpen=false&userTopListCurrentPage=1)

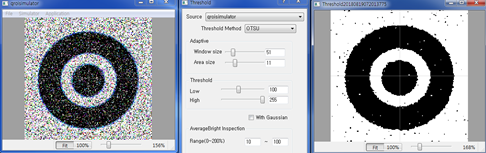
<https://opencv-python.readthedocs.io/en/latest/doc/09.imageThresholding/imageThresholding.html>  
위 사이트에서 가져온 이미지입니다. 임계값보다 크면 백, 작으면 흑이 됩니다.  
   
**2)     기본임계처리2 : cvInRangeS()**  
CvScalar lower 와 upper 사이의 값을 백으로, 나머지는 흑이 됩니다.  
cvInRangeS(grayImg, cv::Scalar(low), cv::Scalar(high), outImg) 형태로 사용하면 gray 이미지를 이치화 할수 있습니다.  
Qroisimulator의 DialogThreshold::**ThresholdRange**()함수에 사용 예가 있습니다.  
   
cv::inRange(imgHSV, cv::Scalar(30, 50, 50), cv::Scalar(90, 255, 255), outImg);  
위의 함수는 HSV Color space 이미지를 입력으로 Hue Value가 30~90(Green Color영역)  
Saturation 50~255, Value 50~255의 이미지 영역은 백, 나머지는 흑이됩니다.

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221341480823&categoryNo=10&parentCategoryNo=0&viewDate=&currentPage=1&postListTopCurrentPage=&from=postList&userTopListOpen=true&userTopListCount=5&userTopListManageOpen=false&userTopListCurrentPage=1)

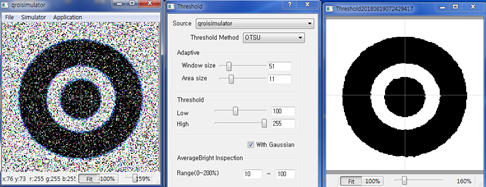
즉, 순수한 Green Color 영역만 확인할 목적이라고 보면 되겠습니다.  
   
**3)     OTSU 이진화**

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221341480823&categoryNo=10&parentCategoryNo=0&viewDate=&currentPage=1&postListTopCurrentPage=&from=postList&userTopListOpen=true&userTopListCount=5&userTopListManageOpen=false&userTopListCurrentPage=1)

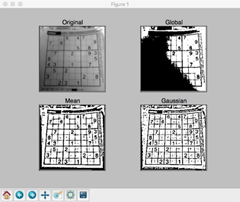
<https://opencv-python.readthedocs.io/en/latest/doc/09.imageThresholding/imageThresholding.html>  
위 사이트에서 가져온 이미지입니다. 임계값보다 크면 백, 작으면 흑이 됩니다.  
bimodal image (히스토그램으로 분석하면 2개의 peak가 있는 이미지)의 경우는 히스토그램에서 임계값을 어느정도 정확히 계산 할 수 있습니다. OTSU의 이진화(Otsu’s Binarization)란 bimodal image에서 임계값을 자동으로 계산해주는 것을 말합니다.  
Qroisimulator의 DialogThreshold::**ThresholdOTSU** ()함수에 사용 예가 있습니다.  
   
원본이미지를 Gaussian filtering를 거친후 OTSU 이진화를 실시하면 좋은 이진화 결과를 얻을 수 있습니다.

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221341480823&categoryNo=10&parentCategoryNo=0&viewDate=&currentPage=1&postListTopCurrentPage=&from=postList&userTopListOpen=true&userTopListCount=5&userTopListManageOpen=false&userTopListCurrentPage=1)

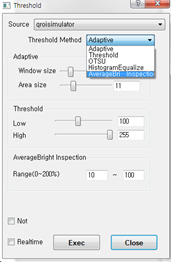
cvSmooth(iplImg, iplImg, CV\_GAUSSIAN,7,7);  
위와같이 가우시안 7x7 필터를 이용한후 CV\_THRESH\_OTSU를 이용하면 아래와 같은 결과를 얻을수 있습니다.

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221341480823&categoryNo=10&parentCategoryNo=0&viewDate=&currentPage=1&postListTopCurrentPage=&from=postList&userTopListOpen=true&userTopListCount=5&userTopListManageOpen=false&userTopListCurrentPage=1)

**4)     적응적 임계처리 : cv::adaptiveThreshold()**

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221341480823&categoryNo=10&parentCategoryNo=0&viewDate=&currentPage=1&postListTopCurrentPage=&from=postList&userTopListOpen=true&userTopListCount=5&userTopListManageOpen=false&userTopListCurrentPage=1)

<https://opencv-python.readthedocs.io/en/latest/doc/09.imageThresholding/imageThresholding.html>  
위 사이트에서 가져온 이미지입니다. 임계값보다 크면 백, 작으면 흑이 됩니다.  
설정영역의 평균값을 계산해서 이치화를 실행하므로 불규칙한 조명 아래에서 capture된 이미지들에대한 이치화에 적합합니다.  
ADAPTIVE\_THRESH\_MEAN\_C : 영역의 평균값으로 결정  
ADAPTIVE\_THRESH\_GAUSSIAN\_C : 영역내의 Gaussian가중치를 이용한 평균값으로 결정  
Qroisimulator의 DialogThreshold::**AdaptiveThreshold** ()함수에 사용 예가 있습니다.  
   
<https://github.com/JonglockYoon/QT-qroilib/blob/master/qroisimulator/app/dialogthreshold.cpp>

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221341480823&categoryNo=10&parentCategoryNo=0&viewDate=&currentPage=1&postListTopCurrentPage=&from=postList&userTopListOpen=true&userTopListCount=5&userTopListManageOpen=false&userTopListCurrentPage=1)

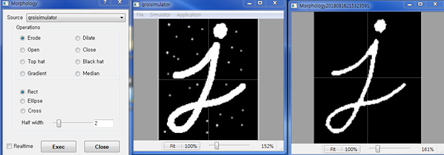
qroilibsimulator Threshold 기능을 이용해서 시험을 할수 있습니다.  
 

##### 2.    모폴로지(Morphology)

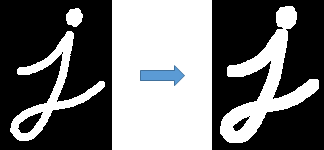
**1)  Erosion(침식연산)**

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221341480823&categoryNo=10&parentCategoryNo=0&viewDate=&currentPage=1&postListTopCurrentPage=&from=postList&userTopListOpen=true&userTopListCount=5&userTopListManageOpen=false&userTopListCurrentPage=1)

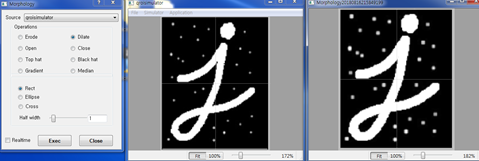
<https://docs.opencv.org/3.4/d9/d61/tutorial_py_morphological_ops.html>  
위 사이트에서 가져온 이미지입니다. 백색 잡음은 제거 되지만 원본이미지의 백색영역이 축소된 현상을 볼수 있습니다.  
   
다음 그림은 qroisimulator의 Eroide 기능을 이용해본것입니다.

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221341480823&categoryNo=10&parentCategoryNo=0&viewDate=&currentPage=1&postListTopCurrentPage=&from=postList&userTopListOpen=true&userTopListCount=5&userTopListManageOpen=false&userTopListCurrentPage=1)

**2)  Dilate(팽창연산)**

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221341480823&categoryNo=10&parentCategoryNo=0&viewDate=&currentPage=1&postListTopCurrentPage=&from=postList&userTopListOpen=true&userTopListCount=5&userTopListManageOpen=false&userTopListCurrentPage=1)

<https://docs.opencv.org/3.4/d9/d61/tutorial_py_morphological_ops.html>  
위 사이트에서 가져온 이미지입니다. 흑색 잡음은 제거 되지만 원본이미지의 백색영역이 확장된 현상을 볼수 있습니다.

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221341480823&categoryNo=10&parentCategoryNo=0&viewDate=&currentPage=1&postListTopCurrentPage=&from=postList&userTopListOpen=true&userTopListCount=5&userTopListManageOpen=false&userTopListCurrentPage=1)

다음 사이트에 OpenCV 모폴로지 기능을 이용한 Simulation Program이 있습니다.  
<https://github.com/JonglockYoon/QT-qroilib/blob/master/qroisimulator/app/dialogmorphology.cpp>  
   
**3)  Open**

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221341480823&categoryNo=10&parentCategoryNo=0&viewDate=&currentPage=1&postListTopCurrentPage=&from=postList&userTopListOpen=true&userTopListCount=5&userTopListManageOpen=false&userTopListCurrentPage=1)

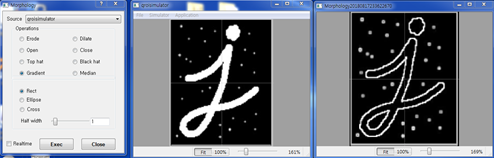
<https://docs.opencv.org/3.4/d9/d61/tutorial_py_morphological_ops.html>  
위 사이트에서 가져온 이미지입니다. 백색잡음을 제거할 목적으로 많이 이용됩니다  
원본이미지의 원하는 백색영역은 그대로 유지되고 잡음만 제거 되었습니다.  
  
**4)  Close**

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221341480823&categoryNo=10&parentCategoryNo=0&viewDate=&currentPage=1&postListTopCurrentPage=&from=postList&userTopListOpen=true&userTopListCount=5&userTopListManageOpen=false&userTopListCurrentPage=1)

<https://docs.opencv.org/3.4/d9/d61/tutorial_py_morphological_ops.html>  
위 사이트에서 가져온 이미지입니다. 흑색잡음을 제거할 목적으로 많이 이용됩니다  
원본이미지의 원하는 백색영역은 그대로 유지되고 잡음만 제거 되었습니다.  
   
**5)  Gradient**

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221341480823&categoryNo=10&parentCategoryNo=0&viewDate=&currentPage=1&postListTopCurrentPage=&from=postList&userTopListOpen=true&userTopListCount=5&userTopListManageOpen=false&userTopListCurrentPage=1)

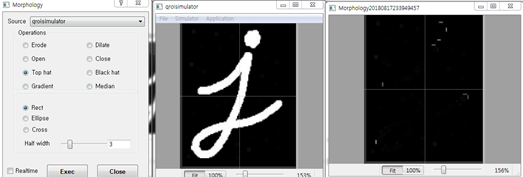
<https://docs.opencv.org/3.4/d9/d61/tutorial_py_morphological_ops.html>  
위 사이트에서 가져온 이미지입니다. 백색 영역의 경계선을 추출할 때 용이하게 이용됩니다.

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221341480823&categoryNo=10&parentCategoryNo=0&viewDate=&currentPage=1&postListTopCurrentPage=&from=postList&userTopListOpen=true&userTopListCount=5&userTopListManageOpen=false&userTopListCurrentPage=1)

**6)  Top Hat**

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221341480823&categoryNo=10&parentCategoryNo=0&viewDate=&currentPage=1&postListTopCurrentPage=&from=postList&userTopListOpen=true&userTopListCount=5&userTopListManageOpen=false&userTopListCurrentPage=1)

<https://docs.opencv.org/3.4/d9/d61/tutorial_py_morphological_ops.html>  
위 사이트에서 가져온 이미지입니다. 백색 이미지영역에서 급격히 변화가 생기는 부분을 추출할 때 유용합니다.

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221341480823&categoryNo=10&parentCategoryNo=0&viewDate=&currentPage=1&postListTopCurrentPage=&from=postList&userTopListOpen=true&userTopListCount=5&userTopListManageOpen=false&userTopListCurrentPage=1)

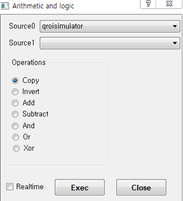
**7)  Black Hat**

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221341480823&categoryNo=10&parentCategoryNo=0&viewDate=&currentPage=1&postListTopCurrentPage=&from=postList&userTopListOpen=true&userTopListCount=5&userTopListManageOpen=false&userTopListCurrentPage=1)

<https://docs.opencv.org/3.4/d9/d61/tutorial_py_morphological_ops.html>  
위 사이트에서 가져온 이미지입니다. 흑색 이미지영역에서 급격히 변화가 생기는 부분을 추출할 때 유용합니다.

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221341480823&categoryNo=10&parentCategoryNo=0&viewDate=&currentPage=1&postListTopCurrentPage=&from=postList&userTopListOpen=true&userTopListCount=5&userTopListManageOpen=false&userTopListCurrentPage=1)

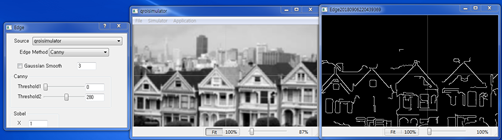
**3.    연산(Arithmetic 과 Logic)**OpenCV에서 이미지들의 선술 연산 및 논리연산을 지원합니다  
   
<https://github.com/JonglockYoon/QT-qroilib/blob/master/qroisimulator/app/dialogarithmeticlogic.cpp>

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221341480823&categoryNo=10&parentCategoryNo=0&viewDate=&currentPage=1&postListTopCurrentPage=&from=postList&userTopListOpen=true&userTopListCount=5&userTopListManageOpen=false&userTopListCurrentPage=1)

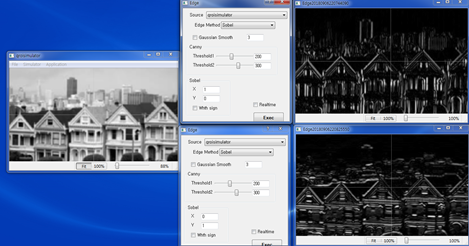
qroilib의 simulator 기능을 이용해서 두개 영상을 입력으로 시험을 할수 있습니다.  
위에 cvMul(), cvDiv()없지만 여러분들이 직접 기능을 추가해서 확인할수 있습니다.  
cvAdd() : Add  
cvSub() : Subtract  
cvMul() :   
cvDiv() :   
cvAnd() : And  
cvOr() : Or  
cvXor() : Xor  
cvNot() : Invert

### 2. OpenCV#2(Canny,Sobel,Laplacian)

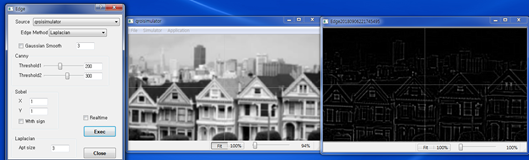
앞에서 이치화(Binarization), 모폴로지(Morphology),연산(Arithmetic 과 Logic) 에 대해서 알아 보았습니다. 여기서는 Edge에 대하여 알아보도록 하겠습니다.  
Edge란 영상의 밝기가 급격하게 변하는 부분을 뜻하며, 실물에서도 변화가 있는 부분이 될 것입니다. Edge를 추출하면 면적 또는 거리등을 측정할수 있을것입니다.  
이치화를 통한 Blob기능으로도 면적,거리등을 측정할수 있지면 Edge기능을 이용하면 좀더 정밀한 결과를 얻을수 있습니다.  
<http://blog.daum.net/trts1004/12109067> 사이트에 Edge에대한 설명이 잘 되어 있는 것 같습니다.  
OpenCV에서 edge기능은 canny, sobel, laplacian이 대표적으로 이야기 됩니다  
   
**캐니(canny)에지**  
   OpenCV Edge검출에서 가장 많이 이용됩니다.  
   Canny Edge함수 호출전 가우시안 필터(Gaussian Filter)를 사용하여 영상에서 노이즈(Noise)를 제거하면 좋은 결과를 얻을수 있습니다 .

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221354695256&parentCategoryNo=&categoryNo=10&viewDate=&isShowPopularPosts=false&from=postList)

<http://carstart.tistory.com/188> 사이트를 참조하면 도움이 될수 있을것으로 보여집니다.  
 cvCanny(in, out, threshold1, threshold2, aperture\_size(3));  
threshold2(강한에지) 이상은 에지로 분류되며,  
threshold1(약한에지) 이하는 에지에서 제외됩니다.  
Threshold1 과 threshold2 사이의 에지들은 연관성을 판별하여 에지를 필터링 하는 과정을 거치게됩니다.   
   
**소벨(sobel)에지**

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221354695256&parentCategoryNo=&categoryNo=10&viewDate=&isShowPopularPosts=false&from=postList)

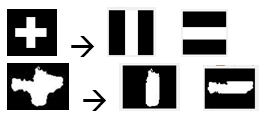
**수평 또는 수직 에지를 추출하는데 유용하다고합니다.  
cvSobel(in, out, xorder, yorder, aperture\_size(3));  
  
   
라플라시안 (Laplacian)에지**

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221354695256&parentCategoryNo=&categoryNo=10&viewDate=&isShowPopularPosts=false&from=postList)

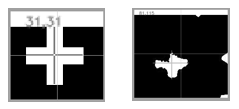
**2차미분을 이용하며 밝기가 점진적으로 변화되는 영역은 반응하지 않고 급격하게 변하는 Edge추출에 적합합니다.  
cvLaplace(in, out, aperture\_size(3));  
   
   
전체 Source Code는 다음에서 볼수 있습니다.  
[https://github.com/JonglockYoon/QT-qroilib/tree/master/qroisimulator](https://github.com/JonglockYoon/QT-qroilib/tree/master/qroisimulator" \t "_blank)**

### 3. OpenCV#3(LineFit ,EdgeCorner,FindShape,CenterOfBlob)

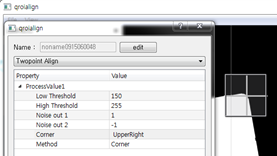
앞에서 에지(Edge)에 대해 살펴 보았으니 여기서는 OpenCV Contours의 몇가지 응용 내용을 소개하고자합니다.  
   
**1)   LineFit : DialogApplication::ExecRansacLinefit()**  
Qroisimulator의 DialogApplication::centerOfPlusmaek() 에서 사용된 ExecRansacLinefit() 함수를 살펴 보겠습니다.  
아래 두개 이미지를 “+” Mark를 찾기 위해 세로선 및 가로선 으로 분리합니다.

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221359367760&parentCategoryNo=&categoryNo=10&viewDate=&isShowPopularPosts=false&from=postList)

위 분리된 세로선 과 가로선 이미지를 ExecRansacLinefit() 함수를 사용해서 선을 구합니다.  
ExecRansacLinefit(croppedImage, 0); // 세로선  
ExecRansacLinefit(croppedImage, 2); // 가로선  
  
cvFindContours(iplImg, storage, &contours, sizeof(CvContour), CV\_RETR\_TREE, CV\_CHAIN\_APPROX\_SIMPLE);  
cvFitLine(contours, CV\_DIST\_L2, 0, 0.01, 0.01, line);  
CvRect boundbox = cvBoundingRect(contours);  
   
double d = sqrt(line[0] \* line[0] + line[1] \* line[1]);  
line[0] /= d;  
line[1] /= d;  
double t = boundbox.width + boundbox.height;  
   
// 올바른 선을 계산하는지 확인하기 위해 영상에 예상 선을 그림  
int x0= line[2]; // 선에 놓은 한 점  
int y0= line[3];  
int x1= x0 - t \* floor(line[0]+0.5); // 기울기에 길이를 갖는 벡터 추가  
int y1= y0 - t \* floor(line[1]+0.5);  
int x2= x0 + t \* floor(line[0]+0.5);  
int y2= y0 + t \* floor(line[1]+0.5);  
cvLine( img2, CvPoint(x1,y1), CvPoint(x2,y2), CV\_RGB(128,128,128), 1, 8 );  
  
위 함수에서는 영상에 선을 그리기위해 정수로 위치를 변환 했지만 정확한 위치를 위해서는 실수로 사용하는 것이 좋습니다.  
   
구해진 세로선 및 가로선의 만나는 지점을 구합니다  
CImgProcBase img;  
img.getIntersectionPoint(plusmarkpt[0],plusmarkpt[1],plusmarkpt[2],plusmarkpt[3], intPnt);  
   
만나는 지점을 그린 결과를 아래와 같이 그려줍니다.

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221359367760&parentCategoryNo=&categoryNo=10&viewDate=&isShowPopularPosts=false&from=postList)

**2)   코너찾기 : CImgAlign::EdgeCorner()**  
Qroialign의 CImgAlign::EdgeCorner() 에서 사용된 contour 사용예를 살펴 보겠습니다.  
다음 이미지의 오른쪽 상단 코너를 찾기위해 rectangle ROI를 생성하고   
  
\_Inspect\_Roi\_Align\_TowPoint, CParam(\_ProcessValue1, ("Low Threshold"), \_IntValue, ("150")),  
\_Inspect\_Roi\_Align\_TowPoint, CParam(\_ProcessValue1, ("High Threshold"), \_IntValue, ("255")),  
\_Inspect\_Roi\_Align\_TowPoint, CParam(\_ProcessValue1, ("Noise out 1"), \_IntValue, ("1")),             
\_Inspect\_Roi\_Align\_TowPoint, CParam(\_ProcessValue1, ("Noise out 2"), \_IntValue, ("-1")),  
\_Inspect\_Roi\_Align\_TowPoint, CParam(\_ProcessValue1, ("Corner"), \_ComboValue, ("0"), ("UpperLeft, UpperRight, BottomLeft, BottomRight")), \_Inspect\_Roi\_Align\_TowPoint, CParam(\_ProcessValue1, ("Method"), \_ComboValue, ("0"), ("Corner, CornerByLine")),  
  
Threshold를 위한 150~255값 입력, Noiseout을 위한 1, -1 값 입력, Corner type선택, 코너찾기 방법 선택을 해서 “Inspection”을 클릭하면

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221359367760&parentCategoryNo=&categoryNo=10&viewDate=&isShowPopularPosts=false&from=postList)

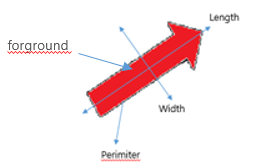
이와 같은 결과를 얻을수 있습니다.

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221359367760&parentCategoryNo=&categoryNo=10&viewDate=&isShowPopularPosts=false&from=postList)

코너찾기 방법의 CornetByLine은 가로세로 수직 경계선을 찾기 때문에 이와 같은 많이 기울어진 이미지에는 적합하지 않습니다.  
각도가 있는 라인을 구해서 만나는 지점을 찾는것도 구현해봄직합니다.  
   
두번째 그림의 이치화 및 가장 적합한 Blob을 찾은 이후 cvFindContours()를 이용해서 세번째 결과를 얻을수 있습니다.  
세번째 결과를 가지고 네개의 꼭지점을 같는 도형을 찾기위해 다음 함수를 이용합니다.  
cvApproxPoly(contours, sizeof(CvContour), storage, CV\_POLY\_APPROX\_DP, d, 1);  
네개 꼭지점을 찾을때까지 d값을 변화 시키면서 루프를 돌립니다.  
네번째 이미지가 네개의 꼭지점을 찾은 결과입니다.  
   
CImgProcBase::**getCorner**() 함수를 이용해서 원하는 코너타입의 위치 Point를 구합니다.  
pt = getCorner(corners, center, CornerType);  
위에서 다섯번째 그림입니다.  
  
  
**3)   도형찾기(qroieditor): MainWindow::SingleROIFindShape()**  
SingleROIFindShape()함수도 이치화와 NoiseOut 과정을 거져서   
cvFindContours()험수로 외곽선을 구합니다.  
구한 외곽선들을 cvApproxPoly함수(윤곽선을 근사화하는 함수)를 이용해서 도형을 추출합니다.  
eps(epsilon,근사화정밀도)는 ROI Parameter로 입력합니다.  
   
**아래 내용은**cvApproxPoly()결과를 이용해서 도형을 판단하는 내용입니다.  
Stack OverFlow에서 가져온것으로 기억됩니다.  
  
vector<cv::Point> approx;  
int size = result->total;  
CvSeq\* c = result;  
//for (CvSeq\* c = result; c != nullptr; c = c->h\_next)  // 엣지의 링크드리스트를 순회하면서 각 엣지들에대해서 출력한다.  
for (int i = 0; i < size; i++)  
{  
// CvSeq로부터좌표를얻어낸다.  
CvPoint\* p = CV\_GET\_SEQ\_ELEM(CvPoint, c, i);  
approx.push\_back(\*p);  
}  
//모든 코너의 각도를 구한다.  
vector<int> angle;  
for (int k = 0; k < size; k++) {  
int ang = base.GetAngleABC(approx[k], approx[(k + 1) % size], approx[(k + 2)%size]);  
angle.push\_back(ang);  
}  
   
std::sort(angle.begin(), angle.end());  
int minAngle = angle.front();  
int maxAngle = angle.back();  
int threshold = 18;  
int type = -1;  
//도형을 판정한다.  
if ( size == 3 ) // triangle  
type = 0;  
else if ( size == 4 && maxAngle < 120 ) // && minAngle >= 90- threshold && maxAngle <= 90+ threshold) // rectangle  
{  
IplImage\* img = cvCreateImage(cvSize(20, 20), IPL\_DEPTH\_8U, 1);  
cvZero(img);  
cvRectangle(img, cvPoint(5, 5), cvPoint(15,15), CV\_RGB(255, 225, 255), CV\_FILLED); // filled rectangle.  
   
CvMemStorage \*s2 = cvCreateMemStorage(0); //storage area for all contours 모든 형상들을 위한 저장공간.  
CvSeq\* c2 = 0;         // 경계 계수를 저장할 변수.  
   
cvFindContours(img, s2, &c2, sizeof(CvContour), CV\_RETR\_TREE, CV\_CHAIN\_APPROX\_SIMPLE);  
double matching = cvMatchShapes(c2, c, CV\_CONTOURS\_MATCH\_I2); // 작은 값 일수록 두 이미지의 형상이 가까운 (0은 같은 모양)라는 것이된다.  
if (matching> 0.5) {  
    type = 1;  
   str = "Parallelogram";  
}  
else if (0.3 < matching && matching < 0.5) {  
    type = 2;  
   str = "rectangle";  
}  
else if (0.02 < matching && matching < 0.3) {  
    type = 3;  
   str = "Rhombus";  
}  
else {  
    type = 4;  
   str = "Square";  
}  
qDebug() << str;  
   
if (img) cvReleaseImage(&img);  
if (c2) cvClearSeq(c2);  
if (s2) cvReleaseMemStorage(&s2);  
   
} else if (size == 5 && minAngle >= 108- threshold && maxAngle <= 108+ threshold) // pentagon  
{  
str = "pentagon";  
qDebug() << str;  
type = 5;  
} else if (size == 6 && minAngle >= 120 - threshold && maxAngle <= 140 + threshold) // hexagon  
{  
str = "hexagon";  
qDebug() << str;  
type = 6;  
} else {  
// if found convex area is ~ PI\*r² then it is probably a circle  
float radius;  
CvPoint2D32f center;  
cvMinEnclosingCircle(c, &center, &radius);  
double tmp = 1.0 - (area / (PI \* pow(radius, 2)));  
if (tmp < 0.3) {  
   str = "circle";  
   qDebug() << str;  
   type = 7;  
}  
}  
  
  
**4)   Blob 중심 찾기 : CImgProcBase::FindCenterOfBlob()**  
cvFindContours() 와 cvApproxPoly()를 이용해서 근사치 도현을 구하고 cvMoments() 와 cvGetSpatialMoment() 를 이용해서  
도형의 중심을 구합니다.  
Point2f point;  
CvSeq\* m\_seq = 0, \*ptseq = 0;  
   
CvMemStorage\* storage = cvCreateMemStorage(0);  
cvFindContours(pImage, storage, &m\_seq, sizeof(CvContour), CV\_RETR\_LIST, CV\_CHAIN\_APPROX\_SIMPLE);  
if (m\_seq != 0)  
{  
CvSeq\* m\_approxDP\_seq = cvApproxPoly(m\_seq, sizeof(CvContour), storage, CV\_POLY\_APPROX\_DP, 0, 1);  
if (m\_approxDP\_seq == NULL)  
{  
    if (storage) cvReleaseMemStorage(&storage);  
    return Point2f();  
}  
   
for (CvSeq\* c = m\_approxDP\_seq; c != NULL; c = c->h\_next)   // 엣지의 링크드리스트를 순회하면서 각 엣지들에대해서 출력한다.  
{  
    ptseq = cvCreateSeq(CV\_SEQ\_KIND\_GENERIC | CV\_32SC2, sizeof(CvContour), sizeof(CvPoint), storage);  
    if (ptseq == NULL) continue;  
    for (int i = 0; i < c->total; ++i)  
    {  
        CvPoint\* p = CV\_GET\_SEQ\_ELEM(CvPoint, c, i);    // CvSeq로부터좌표를얻어낸다.  
        CvPoint temp;  
        temp.x = p->x;  
        temp.y = p->y;  
        cvSeqPush(ptseq, &temp);  
    }  
    point = CenterOfMoment(ptseq);  
    cvClearSeq(ptseq);  
}  
if (m\_approxDP\_seq) cvClearSeq(m\_approxDP\_seq);  
}  
   
if (m\_seq) cvClearSeq(m\_seq);  
cvReleaseMemStorage(&storage);  
return point;  
  
   
**5)   함수 Parameter**  
**cvFindContours()함수의 mode**  
       CV\_RETR\_EXTERNAL 가장 외각선만 찾는다  
       CV\_RETR\_LIST 모든 윤곽선을 검색한다. 계층관계를 설정하지 않는다.  
       CV\_RETR\_CCOMP 2레벨 계층구조로 모든 윤곽선을 가져온다.   
       CV\_RETER\_TREE모든 윤곽선을 계층적 트리 형태로 찾는다.  
   
**cvFindContours()함수의 method**  
       CV\_CHAIN\_APPROX\_NONE 모든윤곽선 point를 찾는다.  
       CV\_CHAIN\_APPROX\_SIMPLE 라인을 그릴수 있는 끝점들만 남겨놓음                    
       CV\_CHAIN\_APPROX\_TC89\_L1   
       CV\_CHAIN\_APPROX\_TC89-KCOS  
      참고 사이트 : <http://lueseypid.tistory.com/tag/cvDrawContours>  
   
   
**void drawContours()함수의 maxLevel**  
     그려진 윤곽의 최대 레벨.   
     값이 0이면 지정된 윤곽선 만 그려집니다.   
     1이면이 함수는 윤곽선과 모든 중첩 된 윤곽선을 그립니다.   
     2이면이 함수는 윤곽선, 모든 중첩 된 윤곽선, 모든 중첩 된 중첩 윤곽선 등을 그립니다.   
     이 매개 변수는 사용 가능한 계층 구조가있을 때만 고려됩니다.  
     참고 사이트 : <https://blog.naver.com/sea5727/220947909525>  
   
전체 Source Code는 다음에서 볼수 있습니다.  
<https://github.com/JonglockYoon/QT-qroilib/tree/master>

### 4. OpenCV#4(blob feature)

Blob의 특성에 대해 알아 보고자 하고자 합니다.  
이치화는 많이 언급되는 용어인데 Blob이란 용어는 그다지 자주 보이지 않는것 같습니다.  
하지만 머신비젼에서 Blob은 아주 많이 이용되고 있습니다.  
에지는 Blob보다 시간이 많이 소요되는 단점이 있어 Blob처리가 힘은 부분은 에지처리를 하는것이 바람직합니다.  
  
일반적인 영상처리순서는 다음과 같습니다.  
이미지획득 -> 이치화 -> 잡음제거 -> blob화 -> 기준에 만족하지 않는 blob제거 -> 결과 도출

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221365405520&parentCategoryNo=&categoryNo=10&viewDate=&isShowPopularPosts=false&from=postList)

Blob을 형성하고 있는 부분을 Forground라고 하며 1의 값을 가지고 이외의 배경부분을 Background라고 0의 값을 가집니다.  
Application들은 특정 기준을 만족하는 feature를 가지는 blob들을 추출할려고합니다.  
  
blob feature는 area, perimeter, convex perimeter,moment, mean, stddev, breath, length, bounding box, ellipse box 등의 값을 가지며 비젼Library에 따라 더 많은 정보를 제공합니다.  
여기에서는 github의 OpenCVBlobsLib/opencvblobslib 기준으로 설명을 합니다.  
Opencvblobslib를 qroilib에서 그대로 사용하기 때문입니다.  
  
**Breath, Length, Area, Perimeter**  
   
1개 픽셀은 면적이 1이며 둘레가 4입니다. 그럼 4개 픽셀은 면적이 4이면서 둘레가 8이 되겠지요  
위 이미지는 둘레 : 339, 면적 : 3078, 폭 : 20.6(Breath), 길이(Length) : 148.8 입니다.  
다음 코드로 결과를 구할수 있습니다.  
qDebug() << i << "(width,length):" << width << length; //아래 공식 참고.  
qDebug() << "    (area):" << blob->Area();  
qDebug() << "    (perimeter):" << blob->Perimeter();  
  
OpenCVBlobsLib Git에서 폭과 길이를 구하는 공식은 아래와 같습니다.  
 double widthC,lengthC;  
 double tmp;   
 tmp = blob->Perimeter()\*blob->Perimeter() - 16\*blob->Area();  
 if( tmp > 0.0 )  
     widthC = (double) (blob->Perimeter()+sqrt(tmp))/4;  
 else  
     widthC = (double) (blob->Perimeter())/4;  
  if(widthC<=0.0)  
     length = width = 0;  
 else {  
     lengthC=(double) blob->Area()/widthC;  
     length=MAX( lengthC , widthC );  
     width=MIN( lengthC , widthC );  
 }  
   
 Blob의 외곽선을 구해서 **ellipse box**를 구할수 있습니다.

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221365405520&parentCategoryNo=&categoryNo=10&viewDate=&isShowPopularPosts=false&from=postList)

blob ellipse

CvBox2D box2d = blob->GetEllipse(); // GetEllipse()함수는 원본 Library가 원하는 결과가 도출되지 않아 Opencv API로 대체하였습니다.  
DrawRotatedRect(tmp,box2d,CVX\_WHITE,1,8,0);  
\* 수정사항 확인은 <https://github.com/JonglockYoon/QT-qroilib/blob/master/qroilib/engine/blob.cpp> 를 참조하세요.  
  
**GetBoundingBox**() 와 **GetConvexHull**()의 결과를 보여줍니다.

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221365405520&parentCategoryNo=&categoryNo=10&viewDate=&isShowPopularPosts=false&from=postList)

convexhull

CvRect rect = blob->GetBoundingBox();  
t\_PointList convexseq = blob->GetConvexHull();  
cvDrawContours(tmp, convexseq, CVX\_WHITE, CVX\_WHITE, 1, 1, 8);  
cvDrawRect(tmp, CvPoint(rect.x, rect.y),CvPoint(rect.x+rect.width, rect.y+rect.height),  
CvScalar(255,255,255), 1, 8);

**표준 편차**(standard deviation)는 산포도를 나타내는 수치로, 표준편차가 작을수록 Pixel평균값에서 Pixel값들의 거리가 가깝다. 즉 Blob내부의 이미지 변화량을 수치화 시킬수 있고, 기준선을 정해서 판단할수 있습니다.  
qDebug() << "    (stddev):" << blob->StdDev(iplImg);

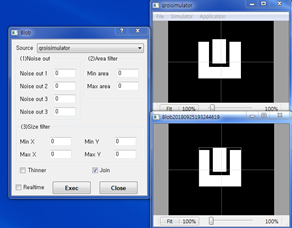
[https://postfiles.pstatic.net/MjAxODA5MjVfMTg1/MDAxNTM3ODcyNDE3NDU4.Cwa6lZDLM372fGleNJJluS7vnrVcAgEWfXEWLyLzyYAg.08Cw1xtnQqh1jtLAriBiDD-c1LpAsxmjrQtEq7HyUaIg.PNG.jerry1455/image.png?type=w773](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221365405520&parentCategoryNo=&categoryNo=10&viewDate=&isShowPopularPosts=false&from=postList)

StdDev : 22.42

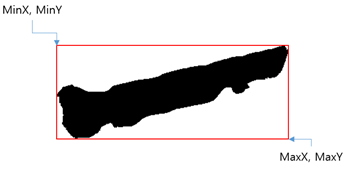
[https://postfiles.pstatic.net/MjAxODA5MjVfMzcg/MDAxNTM3ODcyNDIzMjg3.62_m0gHUZzGeyqJYeWGYbszmyGLOOJlhZtDNZCDSfQog.hHctueD2p7FLxNuX6-rhpz1k_ukePk9MyF-mSL8Xa9og.PNG.jerry1455/image.png?type=w773](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221365405520&parentCategoryNo=&categoryNo=10&viewDate=&isShowPopularPosts=false&from=postList)

StdDev : 41.19

위 두개 이미지에서 표준편차가 크면 blob내부에 이미지 변화량이 많다는 것을 나타냅니다.  
   
**Join** : 여러 개의 Blob을 한개의 blob으로 합칩니다

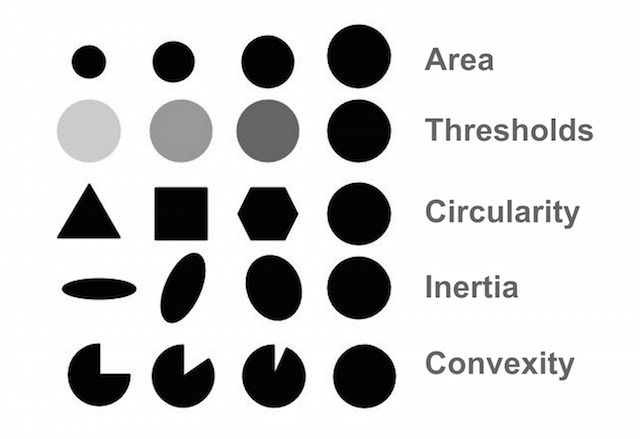
[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221365405520&parentCategoryNo=&categoryNo=10&viewDate=&isShowPopularPosts=false&from=postList)

CBlob \*blob1 = blobs.GetBlob(0);  
for (int i = 1; i < nBlobs; i++) {  
                  CBlob \*blob = blobs.GetBlob(i);  
                  **blob1->JoinBlob(blob);**  
}  
CvRect rect = blob1->GetBoundingBox();  
cvDrawRect(tmp, CvPoint(rect.x, rect.y),CvPoint(rect.x+rect.width, rect.y+rect.height), CvScalar(128,128,128), 1, 8);  
   
두개의 blob이 한 개의 blob으로 합쳐진 것을 표현한것입니다.  
  
   
**평균값**  
   Mean() : Blob냐 Pixel들의 평균값을 구합니다  
  
   
**최대, 최소 좌표**  
  MinX(), MinY(), MaxX(), MaxY() : Blob의 최소, 최대 X, Y 좌표를 구합니다

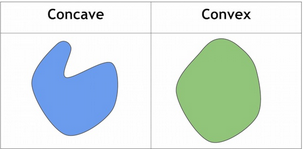
[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221365405520&parentCategoryNo=&categoryNo=10&viewDate=&isShowPopularPosts=false&from=postList)

### 5. OpenCV#5(OpenCV Blob APIs)

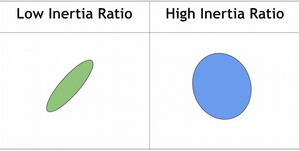
이번 Post는 OpenCV에서 제공되는 Blob기능에 대해 알아보고자합니다.   
  
**SimpleBlobDetector**

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221368080208&parentCategoryNo=&categoryNo=10&viewDate=&isShowPopularPosts=false&from=postList)

'Satya Mallick‘ 님의<https://github.com/spmallick/learnopencv/tree/master/BlobDetector>  와   
<https://www.learnopencv.com/blob-detection-using-opencv-python-c/> 글을 참고합니다.   
  
  
**By Color**  :'Satya Mallick’님의 말에 의하면 이기능은 제대로 동작하지 않는다고 합니다.   
  
**By Size(Area)** :  filterByArea = 1 을 설정하고  minArea 및 maxArea에 적절한 값을 설정하여 Pixel 크기를 기준으로 필터링 할 수 있습니다.   
  
**By Shape(Circularity)** :  filterByCircularity = 1을 설정 하고 minCircularity 와와 maxCircularity를 를 0 과1사이의 값으로 설정해서 진원도를 기준으로 필터링 할 수 있습니다.   
원의 원형도는 1이고 사각형의 원형도는 0.785입니다.  정육각형은 정사각형보다 진원도가 높습니다. 4πArea / (Perimeter\*Perimeter) 수식으로 진원도를 구합니다.   
  
**Convexity** :filterByConvexity = 1로 설정하고 minConvexity 와 maxConvexity를 0 과 1사이의 값으로 설정해서 볼록정도를 기준으로 필터링 할수 있습니다.   
Convexity는 (Blob의 면적 / convex hull의 면적)으로 정의됩니다. 

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221368080208&parentCategoryNo=&categoryNo=10&viewDate=&isShowPopularPosts=false&from=postList)

**Inertia Ratio** :  filterByInertia = 1로 설정하고 minInertiaRatio 와 maxInertiaRatio를 0 과 1사이의 값으로 설정합니다.   형상이 얼마나 길쭉한지를 측정하며  원의 경우이 값은 1이며, 타원의 경우 0과 1 사이 이고, 줄의 경우 0입니다. 

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221368080208&parentCategoryNo=&categoryNo=10&viewDate=&isShowPopularPosts=false&from=postList)

**<SimpleBlobDetector 사용예.>**  
  
// Setup SimpleBlobDetector parameters.   
SimpleBlobDetector::Params params;   
    
// Change thresholds   
params.minThreshold = 10;   
params.maxThreshold = 200;   
    
// Filter by Area.   
params.filterByArea = true;   
params.minArea = 1500;   
    
// Filter by Circularity   
params.filterByCircularity = true;   
params.minCircularity = 0.1;   
    
// Filter by Convexity   
params.filterByConvexity = true;   
params.minConvexity = 0.87;   
    
// Filter by Inertia   
params.filterByInertia = true;   
params.minInertiaRatio = 0.01;   
  
// Set up detector with params   
Ptr<SimpleBlobDetector> detector = SimpleBlobDetector::create(params);   
  
detector->detect(im,keypoints);   
  
// 추출된 결과를im\_with\_keypoints에 원을 그려서 출력해줍니다.   
Mat im\_with\_keypoints;   
drawKeypoints( im, keypoints, im\_with\_keypoints, Scalar(0,0,255), DrawMatchesFlags::DRAW\_RICH\_KEYPOINTS );   
  
//drawKeypoints는 pt 와 size정보를 이용해서 원을 그려줍니다.   
for(inti=0;i<keypoints.size();i++){   
KeyPointk=keypoints[i];   
      //k.pt : 추출 결과위 중심 - Blob의 중심이 아닙니다.   
      //k.size : 추출결과의 diameter. - Blob의 직경이 아닙니다.   
}   
  
  
  
**connectedComponentsWithStats**  
  
  
Blob Pixel 영역의 크기와 Blob의 Min,Max 좌표정보를 추출할수 있습니다.   
  
Matlabels,stats,centroids;   
intnumOfLables=connectedComponentsWithStats(img,labels, stats,centroids,8,CV\_32S);   
for(intidx=0;idx<numOfLables;idx++){   
   intarea=stats.at<int>(idx,CC\_STAT\_AREA);   
   intleft=stats.at<int>(idx,CC\_STAT\_LEFT);   
   inttop=stats.at<int>(idx,CC\_STAT\_TOP);   
   intwidth=stats.at<int>(idx,CC\_STAT\_WIDTH);   
   intheight=stats.at<int>(idx,CC\_STAT\_HEIGHT);   
}   
  
  
\* 아직까지 OpenCV 자체에서 제공되는 Blob처리 기능은 만족하지 못하는 수준으로 보여지며, 저는 현재까지도OpenCVBlobsLib Class 기능을 즐겨사용하고 있습니다.   
  
시간이 허락하는대로 SimpleBlobDetector 에서 제공하고 있는 By Shape, Convexity, Inertia Ratio기능을OpenCVBlobsLib로 가져 와서 사용할려고 생각중입니다.