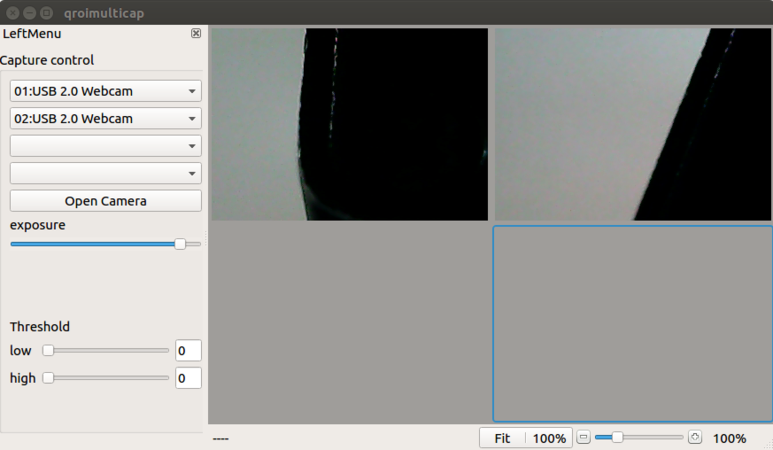
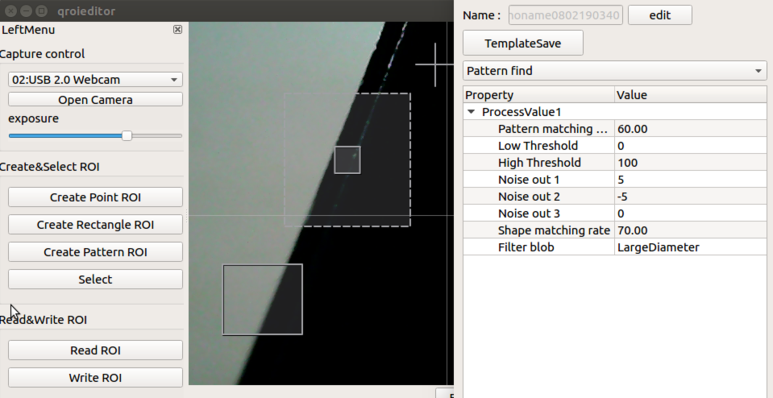
**qroilib란?**

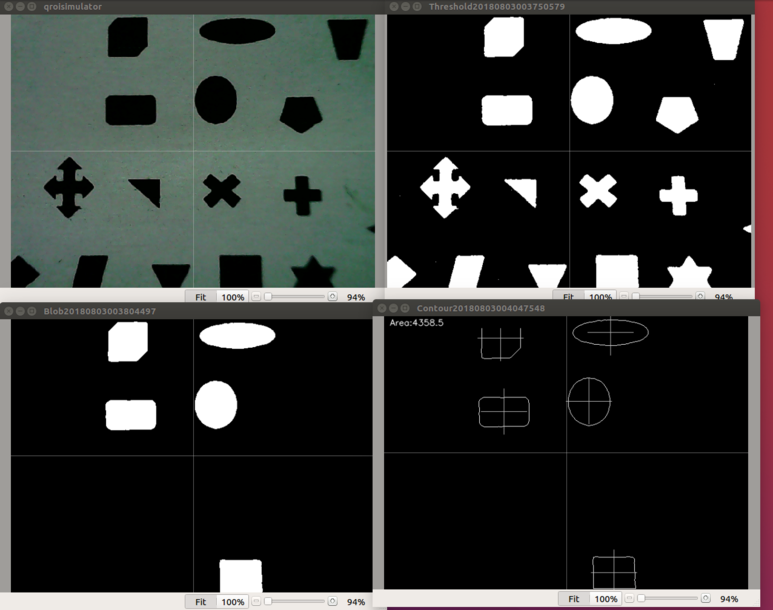
QT를 이용하여 Machine Vision 기능을 구현할때 ROI기능을 추가하기 위해서 손이 많이 가는 작업을 해야한다.  
  
단순히 camera 전체 영역 또는 특정영역을 대상으로 image 분석 및 결과를 만들어 내는 program에서는 딱히 ROI기능을 도입할 필요가 없지만 이미지 분석 영역설정을 해야하고 한개의 camera 영역내에 많은 ROI들이 존재하게 되면 ROI 관리 라는 별도의 기능이 추가 되어야한다.

[](https://blog.naver.com/PostList.nhn?from=postList&blogId=jerry1455&categoryNo=12&currentPage=2)

<multicapture>

[](https://blog.naver.com/PostList.nhn?from=postList&blogId=jerry1455&categoryNo=12&currentPage=2)

<ROI editor>

[](https://blog.naver.com/PostList.nhn?from=postList&blogId=jerry1455&categoryNo=12&currentPage=2)

<Simulator example>

### 1. qroilib를 이용한 image viewer

**1). Qroilib 시작**

qroilib는 처음 ViewMainPage instance를 만드는것으로 시작합니다.

ViewMainPage\* mViewMainPage;

mViewMainPage = new ViewMainPage(mViewStackedWidget);

Editor기능을 이용하고자한다면.아래와 같이

paramTable을 만들고 setParamTable()로 등록을 해야합니다.

extern Qroilib::ParamTable ROIDSHARED\_EXPORT paramTable[];

for (int i=0; i<2; i++)

{

DocumentView\* view = d->mViewMainPage->view(i);

if (view)

{

view->setParamTable(paramTable);

view->bMultiView = true;

connect(view, &Qroilib::DocumentView::finishNewRoiObject,

this, &MainWindow::finishNewRoiObject);

}

}

Qroilib::ParamTable paramTable[] = {

\_Inspect\_Patt\_Start, CParam(\_ProcessValue1, (""), \_IntValue, ("0")), // do not delete.

\_Patt1, CParam(\_ProcessValue1, ("Pattern matching rate"), \_DoubleValue, ("60")),

\_Patt1, CParam(\_ProcessValue1, ("Low Threshold"), \_IntValue, ("120")),

\_Patt1, CParam(\_ProcessValue1, ("High Threshold"), \_IntValue, ("255")),

\_Patt1, CParam(\_ProcessValue1, ("Noise out 1"), \_IntValue, ("-2")), // -1 : morphology Open

\_Patt1, CParam(\_ProcessValue1, ("Noise out 2"), \_IntValue, ("2")), // 1 : morphology Close

\_Patt1, CParam(\_ProcessValue1, ("Shape matching rate"), \_DoubleValue, ("70")),

\_Inspect\_Roi\_Start, CParam(\_ProcessValue1, (""), \_IntValue, ("0")), // do not delete.

\_Roi1, CParam(\_ProcessValue1, ("Direction"), \_ComboValue, ("0"), ("L2R,R2L,T2B,B2T,Center")),

\_Roi1, CParam(\_ProcessValue1, ("Detect method"), \_ComboValue, ("0"), ("Average, First")),

\_Roi1, CParam(\_ProcessValue1, ("Polarity"), \_ComboValue, ("0"), ("W2B, B2W")),

\_Roi1, CParam(\_ProcessValue1, ("Low Threshold"), \_IntValue, ("200")),

\_Roi1, CParam(\_ProcessValue1, ("High Threshold"), \_IntValue, ("255")),

\_Roi1, CParam(\_ProcessValue1, ("Noise out 1"), \_IntValue, ("-2")),

\_Roi1, CParam(\_ProcessValue1, ("Noise out 2"), \_IntValue, ("2")),

\_Roi1, CParam(\_ProcessValue1, ("Ramp width"), \_IntValue, ("6")),

\_Roi1, CParam(\_ProcessValue1, ("Detect method"), \_ComboValue, ("0"), ("Average, First")),

\_Inspect\_Point\_Start, CParam(\_ProcessValue1, (""), \_IntValue, ("0")), // do not delete.

\_Point1, CParam(\_ProcessValue1, ("Mark color"), \_IntValue, ("128")), // not yet implement.

\_Inspect\_Type\_End, CParam(\_FilterValue, (""), \_IntValue, ("")), // do not delete.

};

finishNewRoiObject()는 ROI를 생성한후 호출된다.

bMultiView를 true로 두면 초기 화면에 ROI 정보를 보여주지 않습니다.

**2). ViewMainPage()**

ViewMainPage를 생성할때 아래와 같이 Single 또는 Multi document를 생성합니다.

Single일때에는 bMultiView를 false, Multi일때에는 true로 합니다.

const QUrl ch1 = QUrl("file:channel1");

const QUrl ch2 = QUrl("file:channel2");

QList<QUrl> urls;

urls.append(ch1);

if (gCfg.m\_nCamNumber > 1)

urls.append(ch2);

openUrls(urls, ch1);

"[file:channel1](file:///channel1)” 과 같이 Qurl name인 중복되지 않는 이름을 부여합니다.

**4). ROI Tools**

v->selectTool(v->actSelectTool);

v->selectTool(v->actCreatePoint);

v->selectTool(v->actCreateRectangle);

v->selectTool(v->actCreatePattern);

ROI 선택 도구, Point ROI 생성도구, Rectangle ROI 생성도구, Pattern ROI 생성도구를 선택합니다.

**5). Read Image**

DocumentView\* v = currentView();

QImage img;

v->mRoi->setWidth(img.width());

v->mRoi->setHeight(img.height());

v->document()->setImageInternal(img);

v->imageView()->updateBuffer();

이미지 파일을 로드해서 Display합니다.

**6). Save Image**

DocumentView\* v = currentView();

const QImage \*pimg = v->image();

pimg->save(fileName);

현재 보여지는 이미지를 파일로 Write합니다.

아래 사이트에 qroilib를 이용한 이미지 viewer program이 있습니다.

<https://github.com/JonglockYoon/qroilib/tree/master/qroiviewer>

### 2. Capture Image

OpenCV Capture 기능을 이용하여 Camera로부터 이미지를 Capture하여 qroilib를 이용, Preview를 구현한 Program입니다.

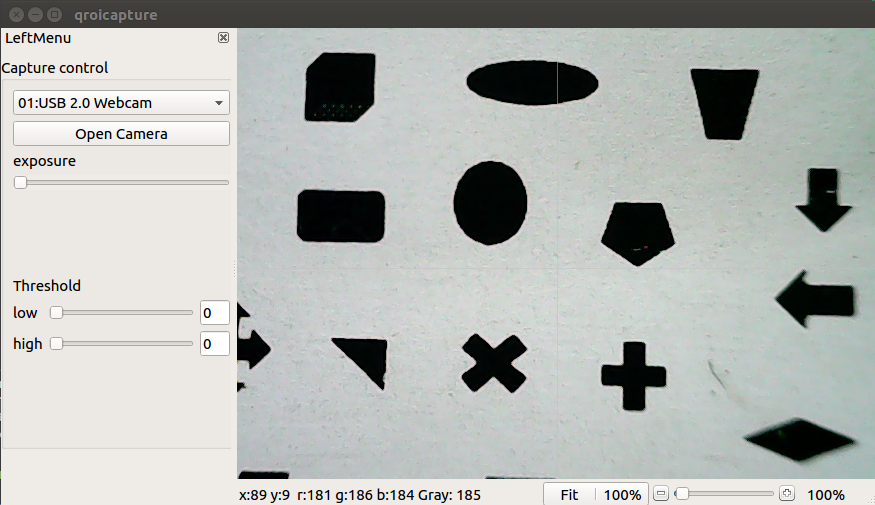
LeftDock \*mLeftDock;

mLeftDock = new LeftDock(tr("Menu"), this);

mLeftDock->setFeatures(QDockWidget::DockWidgetClosable);

addDockWidget(Qt::LeftDockWidgetArea, mLeftDock);

d->viewMenu->addAction(mLeftDock->toggleViewAction());

LeftDock 메뉴가 생성되면서 ComboBox에 현재 연결되어 있는 video device를 선택할수 있게 됩니다.

ViewMainPage\* pView = theMainWindow->viewMainPage();

str = ui->comboBoxCam1Port->currentText();

int ncam1 = str.mid(0,2).toInt();

pView->OpenCam(0, ncam1-1);

“Open Camera” 버턴을 누르면 현재 선택된 Video Device를 Open합니다.

void ViewMainPage::OpenCam(int seq, int nCam)

{

myCamCapture = new CamCapture(d->mDocumentViews[seq], seq, nCam);

QObject::connect(myCamCapture, SIGNAL(processedImage(const QImage&, int)),

this, SLOT(updatePlayerUI(const QImage&, int)));

if (myCamCapture->OpenCam() == true)

myCamCapture->Play();

}

OpenCam()함수에서 cv::VideoCapture 기능을 이용하여 카메라 Device를 Open하게 되고

capture된 이미지들은 updatePlayerUI()함수를 통하여 화면에 Display됩니다.

아래 사이트에 qroilib를 이용한 카메라 image capture program이 있습니다.

<https://github.com/JonglockYoon/qroilib/tree/master/qroicapture>

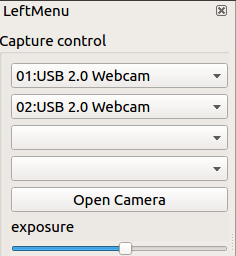
### 3. Multi Capture Image

앞에서 Single Camera Capture Program을 보았으며,

여기서는 4개의 Camera를 동시에 Capture할수 있는 Program을 보고자 합니다.

Qroilib library는 8개의 multi view를 지원하므로 화면을 최대 8분할하여 사용하수 있습니다.

여기서는 화면을 4분할 해서 4개의 카메라를 실시간 capture해서 preview할수 있는 program흐름을 보고자 합니다.



Left Dock화면을보면 카메라 선택 Combo Box가 4개로 늘어 났습니다.

연결된 카메라들을 선택후 “Open Camera”버턴을 클릭하면 선택한 카메라 대상으로

pView->OpenCam(...) 함수가 호출됩니다.

void ViewMainPage::**OpenCam**(int seq, int nCam)

{

myCamCapture[seq] = new CamCapture(d->mDocumentViews[seq], seq, nCam);

QObject::connect(myCamCapture[seq], SIGNAL(processedImage(const QImage&, int)),

this, SLOT(updatePlayerUI(const QImage&, int)));

if (myCamCapture[seq]->OpenCam() == true)

myCamCapture[seq]->Play();

}

OpenCam()함수는 카메라 한 개당 CamCapture Instance를 생성시키고 생성된 CamCapture는

각 연결된 카메라에서 이미지를 획득하고 processedImage Signal을 보내게됩니다.

**updatePlayerUI**(const QImage& img, int seq)의 seq는 view의 순서번호가 들어 있습니다.

이것으로 몇번째 view에 Display할것인지를 결정합니다.

void ViewMainPage::updatePlayerUI(const QImage& img, int seq)

{

Qroilib::DocumentView\* v = view(seq);

v->document()->setImageInternal(img);

v->imageView()->updateBuffer();

}

Tip) USB Camera들은 USB대역폭을 많이 사용하므로 USB HUB를 통하여 여러대의 카메라를 연결하면 대역폭이 모자라 Capture가 되지 않습니다.

PC의 USB Port에 직접 연결해서 사용해야합니다.

Raspberry Pi 3경우는 4개의 USB Port가 있지만 CPU Processing성능이 낮으므로 30만화소 Webcam Full Speed로 2개를 사용할수 없습니다.

2개이상의 Camera를 사용할려고 한다면 대역폭 조정이 되는 Camera를 선택 하셔야합니다.

대역폭조정)

CV\_CAP\_PROP\_FPS set을 이용해서 FPS(Frame Per Sec)설정이 되었는지 확인합니다.

대역폭 조정이 되지 않는 Camera도 있으니 get으로 확인절차가 필요합니다.

capture.set(CV\_CAP\_PROP\_FPS, val) 함수는 시스템 또는 Camera에 따라 지원되지 않는경우가 있습니다.

qroicapture program으로 확인할수 있습니다.

### 4. PiCam Capture

이 Program은 Raspberry pi에서만 동작하는 Program입니다.  
PiCam은 CSI(Camera Serial Interface) 방식으로 동작합니다.  
앞선 Program들은 USB Interface Camera들을 OpenCV함수로 Camera Image를 Capture한 반면 여기에서는 Raspberry Pi에서 제공하는 Camera Interface Program을 사용합니다.  
  
$ sudo su raspi-config  
명령을 이용해서 Pi Camera를 Enable 해야합니다. : interfaceing options -> P1 Camera [enable]   
  
다음은 picam program을 download 및 설치 절차를 거칩니다.  
$ wget <https://downloads.sourceforge.net/project/raspicam/raspicam-0.1.6.zip>   
$ unzip raspicam-0.1.6.zip   
$ cd raspicam-0.1.6   
$ mkdir build   
$ cd build   
$ cmake ..   
$ make sudo make install  
$ sudo ldconfig   
  
위의 작업으로 Raspberry Pi Camera를 사용가능하게 되었습니다.  
  
raspicam\_cv.h file을 include합니다.  
[#include](https://blog.naver.com/PostListByTagName.nhn?blogId=jerry1455&encodedTagName=include) <raspicam/raspicam\_cv.h>  
  
raspicam::RaspiCam\_Cv capture;  
  
raspicam::RaspiCam\_Cv 는 Opencv의 cv::VideoCapture class와 interface를 거의 동일하게   
만들어 두어서 OpenCV Capture Program을 조금만 변경하면 사용할수 있습니다.  
  
picapture.cpp를 보시면 약간의 차이를 구분할수 있을것으로 보여집니다.  
  
문서에는 3280 x 2464(8M) 까지 지원된다고 되어 있습니다.  
qroipicam으로 시험을 해보면 1280x1024까지가 가장 적합한 이미지를 얻을수 있습니다.  
  
3280 x 2464 color 이미지는 약 200M Byte입니다.  
Raspberry Pi 1Giga로 몇장의 이미지만 Memory에 올려도 문제가 생길수 있습니다.  
  
1280x1024 color이미지는 약 30MByte이므로 여러장의 이미지를 Memory에 올려서 처리하더라도 문제가 없습니다.  
  
OpenCV를 이용하여 Image처리를 하면 여러장의 이미지 Memory공간을 차지하게 됩니다.  
원본이미지, 이치화이미지, 모폴로지이미지 등 많은 이미지 메모리공간이 필요합니다.  
  
**EXPOSURE)**  
  
capture.set(CV\_CAP\_PROP\_AUTO\_EXPOSURE, 1); // 1-manual, 3-auto  
  
CV\_CAP\_PROP\_AUTO\_EXPOSURE 값이 1(manual)일때는 CV\_CAP\_PROP\_EXPOSURE 다음 값을 가집니다.  
-1 640 ms   
-2 320 ms  
-3 160 ms  
-4 80 ms  
-5 40 ms  
-6 20 ms  
-7 10 ms  
-8 5 ms  
-9 2.5 ms  
-10 1.25 ms  
-11 650 μs  
-12 312 μs  
-13 150 μs  
값이 적어질수록 노출이 짧아지므로 같은 조명 아래에서 이미지가 흐려지게되며, Shutter속도가 빨라지므로 고속촬영이 가능해집니다.  
  
CV\_CAP\_PROP\_AUTO\_EXPOSURE 값이 3(auto)으로 설정되어 있으면 카메라가 자동노출설정을 합니다.  
즉, 조명값이 변하게되면 CV\_CAP\_PROP\_EXPOSURE를 자동으로 변화 시켜 가장 좋은 이미지를 capture하게 설정합니다.  
  
이 option들은 camera에 따라 지원되지 않을수 있으니, 반드시 확인후 사용해야합니다.  
  
아래 사이트에 qroilib를 이용한 picam capture program이 있습니다.   
<https://github.com/JonglockYoon/QT-qroilib/tree/master/qroipicam>

### 5. qroilib Editor

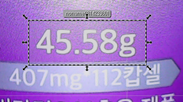
ROI(Region Of Interest, 관심영역)은 카메라에서 획득한 전체 이미지 영역중 관심영역만 검사를 하기위해서 일부 영역을 지칭하는것입니다.  
OpenCV에서도 cvSetImageROI(),cvResetImageROI() ROI API를 지원하고 있습니다.  
  
**5.1) Qroilib는 3가지의 ROI를 지원합니다.**  
1)     Point ROI : 기준점이 필요할 때 사용할수 있습니다.  
2)     Rectangle ROI : 영역 ROI로써 ROI영역내에서 이치화,모폴로지,Blob처리, Edge처리등을 빠르게 처리할 수 있습니다.  
3)     Pattern ROI : 검색영역, 패턴영역을 지원하여 Pattern 검색에 적합하도록 ROI를 지원합니다.  
4)     Select : 이미 작성되어 있는 ROI들을 선택하여 영역 조정, 속성조정등을 수정할수 있는 r기능을 지원합니다.  
5)     Read ROI : 파일로 저장된 ROI들을 불러옵니다.  
Write ROI : 생성 및 수정된 ROI정보들을 파일로 저장합니다.

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221339548001&categoryNo=0&parentCategoryNo=0&viewDate=&currentPage=3&postListTopCurrentPage=1&from=section&userTopListOpen=true&userTopListCount=30&userTopListManageOpen=false&userTopListCurrentPage=3)

**5.2) qroilib에서 지원되는 ROI Type들**  
1) Point ROI

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221339548001&categoryNo=0&parentCategoryNo=0&viewDate=&currentPage=3&postListTopCurrentPage=1&from=section&userTopListOpen=true&userTopListCount=30&userTopListManageOpen=false&userTopListCurrentPage=3)

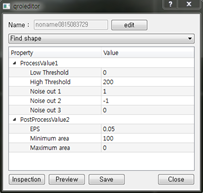
“+” 마크로 표시되며 기준점을 지정하기위해 사용됩니다.  
분석결과가 이 기준점으로부터 거리가 얼마인지 알고자할 때 많이 이용됩니다.  
  
2) Rectangle ROI

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221339548001&categoryNo=0&parentCategoryNo=0&viewDate=&currentPage=3&postListTopCurrentPage=1&from=section&userTopListOpen=true&userTopListCount=30&userTopListManageOpen=false&userTopListCurrentPage=3)

지정 영역내 이미지를 분석할 때 이용합니다.  
위의 ROI의 이미지를 분석하면 “45.58g”의 글자를 추출할수 있을것입니다.  
  
3) Pattern ROI

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221339548001&categoryNo=0&parentCategoryNo=0&viewDate=&currentPage=3&postListTopCurrentPage=1&from=section&userTopListOpen=true&userTopListCount=30&userTopListManageOpen=false&userTopListCurrentPage=3)

 바깥 영역이 검색영역이며 안쪽 사각영역이 패턴입니다.  
패턴이미지를 등록후 검색영역 내에서 등록된 패턴이미지를 찾는 기능을 구현하기 쉽도록 지원합니다.  
  
**5.3) ROI Editor기능을 사용하기위한 준비작업**  
**1) paramTable 등록**  
setParamTable()을 이용해서 paramTable을 등록하는것으로 시작합니다.  
“1. Qroilib를 이용한 image viewer” paramTable을 참조하세요.  
extern Qroilib::ParamTable ROIDSHARED\_EXPORT paramTable[];  
DocumentView\* v = currentView();  
v->setParamTable(paramTable);  
v->selectTool(v->actSelectTool);  
**2) ROI 생성**  
v->selectTool(v->actCreatePoint);  
v->selectTool(v->actCreateRectangle);  
v->selectTool(v->actCreatePattern);  
위 함수를 호출해서 원하는 Type의 ROI를 생성할수 있습니다.  
생성방법은 함수호출후 Left Mouse버턴을 누른채로 드래깅을 하면 됩니다.  
   
**3) ROI선택,삭제,수정**  
v->selectTool(v->actSelectTool);  
위 함수를 호출한후 Left Mouse버턴을 누르면 누른 위치의 ROI가 선택됩니다.  
여러 개의 ROI가 겹쳐있을 수도 있으므로 ROI경계선을 클릭합니다.  
   
선택된 ROI를 삭제할려면 Del 버턴을 눌러서 삭제할수 있습니다.  
수정할려면 Left Mouse 버턴을 DoubleClick하면됩니다.  
   
DoubleClick할 때 샐행할 함수를 미리 아래와 같이 등록합니다.  
connect(view, &Qroilib::DocumentView::editCurrentObject, this, &ViewMainPage::editRoiProperty);  
  
여러 개의 view가 생성(qroilib는 8개의 view까지 지원)될수 있으므로 각 view마다 함수등록이 필요합니다.  
   
**5.4) Editor Dialog**  
아래와 같이 ROI Editor Dialog를 생성할수 있습니다.  
5.3)항에서 등록된 paramTable의 ROI Property들을 수정할수 있습니다.  
  
void ViewMainPage::editRoiProperty()  
{  
    Qroilib::DocumentView \*v = currentView();  
    Qroilib::RoiObject \*mObject;  
    if (v->mSelectedObjects.size() == 1) {  
          mObject = v->mSelectedObjects.first();  
    } else return;  
    if (roipropertyeditor != nullptr) {  
        delete roipropertyeditor;  
    }  
    roipropertyeditor = new RoiPropertyEditor(this, mObject);  
    roipropertyeditor->show();  
    roipropertyeditor->raise();  
}  
  
Inspect\_Roi\_Find\_Shape,  CParam(\_ProcessValue1, ("Low Threshold"), \_IntValue, ("0")),  
\_Inspect\_Roi\_Find\_Shape,  CParam(\_ProcessValue1, ("High Threshold"), \_IntValue, ("200")),  
\_Inspect\_Roi\_Find\_Shape,  CParam(\_ProcessValue1, ("Noise out 1"), \_IntValue, ("1")),  
\_Inspect\_Roi\_Find\_Shape,  CParam(\_ProcessValue1, ("Noise out 2"), \_IntValue, ("-1")),  
\_Inspect\_Roi\_Find\_Shape,  CParam(\_ProcessValue1, ("Noise out 3"), \_IntValue, ("0")),  
\_Inspect\_Roi\_Find\_Shape,  CParam(\_DecideValue, ("EPS"), \_DoubleValue, ("0.05")),  
\_Inspect\_Roi\_Find\_Shape,  CParam(\_DecideValue, ("Minimum area"), \_IntValue, ("100")),  
\_Inspect\_Roi\_Find\_Shape,  CParam(\_DecideValue, ("Maximum area"), \_IntValue, ("0")),

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221339548001&categoryNo=0&parentCategoryNo=0&viewDate=&currentPage=3&postListTopCurrentPage=1&from=section&userTopListOpen=true&userTopListCount=30&userTopListManageOpen=false&userTopListCurrentPage=3)

위의 \_Inspect\_Roi\_Find\_Shape ROI Type이 Dialog상에 나타난 결과입니다.  
Dialog상에서 Value값을 변경하여 상황에 알맞은 이미지 처리를 할수 있습니다.  
   
**5.5) RoiPropertyEditor Class**  
RoiPropertyEditor Class에서 ROI Property를 변경하는 Dialog를 지원합니다.  
**1)     RoiPropertyEditor::RoiPropertyEditor()함수**  
ROI Type별로 구분해서 선택 ComboBox를 준비합니다.  
  
switch(mObject->shape())  
{  
case Qroilib::RoiObject::Pattern:  
    for(int i=\_Inspect\_Patt\_Start+1; i<\_Inspect\_Patt\_End; i++){  
        theClassNames.append(g\_cRecipeData->m\_sInspList[i]);  
    }  
    nCbIdx = mObject->mInspectType - (\_Inspect\_Patt\_Start + 1);  
    break;  
case Qroilib::RoiObject::Rectangle:  
    for(int i=\_Inspect\_Roi\_Start+1; i<\_Inspect\_Roi\_End; i++){  
        theClassNames.append(g\_cRecipeData->m\_sInspList[i]);  
    }  
    nCbIdx = mObject->mInspectType - (\_Inspect\_Roi\_Start + 1);  
    break;  
   
case Qroilib::RoiObject::Point:  
    for(int i=\_Inspect\_Point\_Start+1; i<\_Inspect\_Point\_End; i++){  
        theClassNames.append(g\_cRecipeData->m\_sInspList[i]);  
    }  
    nCbIdx = mObject->mInspectType - (\_Inspect\_Point\_Start + 1);  
    break;  
}  
  
**2)     RoiPropertyEditor::ObjectListSet() 함수**  
QtTreePropertyBrowser \*variantEditor; 에 ROI 속성들을 등록하는 작업을 하며  
variantEditor->raise()로 화면에 보여지게합니다.  
   
ROI Value 속성들에는 \_BoolValue,\_IntValue,\_DoubleValue,\_StringValue,\_ComboValue Type들을 지원합니다.  
  
int size = mObject->m\_vecParams.size();  
for (int i = 0; i < size; i++)  
{  
CParam \*c = &mObject->m\_vecParams[i].param;  
...  
if (c->valueType == \_BoolValue) {  
    QtVariantProperty \*item = variantManager->addProperty(QVariant::Bool, c->Name);  
    item->setValue(c->Value.toInt());  
    topItem->addSubProperty(item);  
}  
else if (c->valueType == \_IntValue) {  
...  
}  
else if (c->valueType == \_DoubleValue) {  
...  
}  
else if (c->valueType == \_StringValue) {  
...  
}  
else if (c->valueType == \_ComboValue) { // ,로 구분된 선택 Table처리  
...  
}  
}  
if (topItem != nullptr) variantEditor->addProperty(topItem);  
variantEditor->raise()  
  
**3)     RoiPropertyEditor:: ObjectListGet () 함수**  
Dialog상에서 수정된 값들을 paramTable로 반영합니다.  
Memory Update가 되었으므로 수정된 ROI 정보를 파일로 저장할수 있습니다.  
  
  
**4)     RoiPropertyEditor::clickedSaveButtonSlot() 함수**  
qroilib의 RoiWrite class를 이용해서 xml type 파일로 ROI정보를 저장합니다.  
  
QString fileName = "./RecipeData.roi";  
RoiWriter writer;  
   
while (true) {  
Qroilib::DocumentView\* v = pMainView->view(seq);  
if (v == nullptr)  
    break;  
   
writer.writeRoiGroup(v->mRoi);  
seq++;  
}  
   
writer.writeRoiGroupEnd();  
  
**5)     RoiPropertyEditor:: SaveTemplelateImage () 함수**  
Qroilib::RoiObject::Pattern Type의 ROI인경우 패턴영역의 이미지를 패턴매칭에 이용하기 위해 파일로 보관해야합니다.  
현재 패턴영역의 이미지를 패턴이미지파일로 저장하는 기능을 담당합니다.  
   
**6)     RoiPropertyEditor:: clickedInspButtonSlot () 함수**  
MainWindow::InspectOneItem()함수를 호출하여 ROI Property들을 이용해서 이미지 분석을 진행합니다.  
OpenCV를 이용한 이미지 분석은 이후 다른 포스팅 글들에서 다룰 예정입니다.  
   
아래 사이트에 qroilib를 이용한 ROI Editor program이 있습니다.   
[https://github.com/JonglockYoon/QT-qroilib](https://github.com/JonglockYoon/QT-qroilib/tree/master/qroieditor)

### 6. qroilib Align

카메라로 이미지를 획득하고 분석을 할 때 이미지 분석전 얼라인 단계를 거치는 경우가 많습니다.  
얼라인은 기구적인 얼라인 과 Software얼라인으로 구분할수 있습니다.  
고 정밀도를 요구하는 검사 Program은 기구적인 얼라인 스테이지가 필요합니다.  
이미지를 한번 획득해서 X,Y,Theta 이동값을 계산하고 기구스테이지를 이동해서 제품을 얼라인하는 방식입니다.  
**1)카메라로 이미지 획득**

[https://postfiles.pstatic.net/MjAxODA4MTlfMjcg/MDAxNTM0NjgyNTQ1MTI3.qJbLwKPJzAQXW_FEUmjrTD5UzF5caKvG13Eocel0XO0g.a7fXFbzy973RNjYSTcfcKleQ1-raqbFnvIyvjPBut4cg.PNG.jerry1455/image.png?type=w773](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221341913377&categoryNo=0&parentCategoryNo=0&viewDate=&currentPage=3&postListTopCurrentPage=1&from=section&userTopListOpen=true&userTopListCount=30&userTopListManageOpen=false&userTopListCurrentPage=3)

**2)이미지분석 및 기구이동(X,Y,Theta)**

[https://postfiles.pstatic.net/MjAxODA4MTlfMjE3/MDAxNTM0NjgyNTYzMzM2.07JZL9tR7Kvi0rDyPOrmDEskas6To-RagpZGaDQsWsIg.Bx3Js0QqOT8bPrYgxhGAND0n8MVogPmD9LfPGIV-QRMg.PNG.jerry1455/image.png?type=w773](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221341913377&categoryNo=0&parentCategoryNo=0&viewDate=&currentPage=3&postListTopCurrentPage=1&from=section&userTopListOpen=true&userTopListCount=30&userTopListManageOpen=false&userTopListCurrentPage=3)

**3)카메라로 이미지 획득**

[https://postfiles.pstatic.net/MjAxODA4MTlfMTk5/MDAxNTM0NjgyNTc1ODIz.Jk4Vr4OIUECbPjPxBUCmlbuNc5Hr_bZHWA9DPosVLEkg.UYyW99oKAleFPRzxwv6Ogs2c74RuzhjZpivvVvy98osg.PNG.jerry1455/image.png?type=w773](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221341913377&categoryNo=0&parentCategoryNo=0&viewDate=&currentPage=3&postListTopCurrentPage=1&from=section&userTopListOpen=true&userTopListCount=30&userTopListManageOpen=false&userTopListCurrentPage=3)

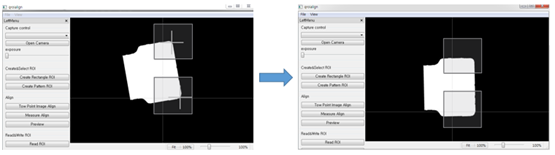
**4)분석**  
   
위 철차를 거쳐 얼라인된 이미지를 Capture하여 분석진행을 합니다.  
  
   
얼라인 스테이지를 이용하지 않고 Software만으로의 얼라인은 다음 절차를 거칩니다.  
**1)카메라로 이미지 획득**

[https://postfiles.pstatic.net/MjAxODA4MTlfMTY2/MDAxNTM0NjgyNTk2Njcz.yQoOfNI5DzfnCOsJ0qhmkYZfwnJ1PdVEjq-Ees2wqRQg.4Ec2odIwprQQ8NEfulInYjUMgjTJF6BoibKI0A9rrGog.PNG.jerry1455/image.png?type=w773](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221341913377&categoryNo=0&parentCategoryNo=0&viewDate=&currentPage=3&postListTopCurrentPage=1&from=section&userTopListOpen=true&userTopListCount=30&userTopListManageOpen=false&userTopListCurrentPage=3)

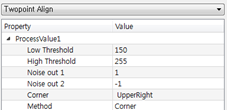
**2)이미지분석 및 이미지 이동 및 회전**

[https://postfiles.pstatic.net/MjAxODA4MTlfMTI5/MDAxNTM0NjgyNjEwMTI5.JXI0DzJTPxSZ7WCf2QdEF2tMclanuteL1GwZYZlAR7kg.qCemFpwohxV_kBVBtb3jd1RV5-FYreb_J15USxQpx2Ig.PNG.jerry1455/image.png?type=w773](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221341913377&categoryNo=0&parentCategoryNo=0&viewDate=&currentPage=3&postListTopCurrentPage=1&from=section&userTopListOpen=true&userTopListCount=30&userTopListManageOpen=false&userTopListCurrentPage=3)

**3)분석**  
Software align을 할때의 단점은 이미지회전을 할 때 원본이미지에 약간의 손상이 일어납니다.

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221341913377&categoryNo=0&parentCategoryNo=0&viewDate=&currentPage=3&postListTopCurrentPage=1&from=section&userTopListOpen=true&userTopListCount=30&userTopListManageOpen=false&userTopListCurrentPage=3)

**5.1) TwoPoint Align**  
두개의 얼라인 ROI를 이용해서 수직으로 이미지를 Rotate시킨후 X,Y 이미지 Shift를 시킵니다.

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221341913377&categoryNo=0&parentCategoryNo=0&viewDate=&currentPage=3&postListTopCurrentPage=1&from=section&userTopListOpen=true&userTopListCount=30&userTopListManageOpen=false&userTopListCurrentPage=3)

ROI Editor 기능을 이용하여 두개의 Align Point를 구할 Parameter를 설정합니다.  
위는 150~255사이의 Gray Level을 흰색으로 이치화시킨후 Noise를 제거하기위해 Morphology Close,Open후 Corner를 구합니다.  
CImgAlign::SingleROICorner() 함수에서 EdgeCorner() 와 EdgeCornerByLine()을 지원합니다.  
   
Qroilib에서 blob처리는 <https://github.com/OpenCVBlobsLib/opencvblobslib> Library를 이용합니다.  
EdgeCorner()는 Blob GetBoundingBox()함수를 이용해서 Blob의 코너를 구합니다.  
EdgeCornerByLine()은 가로 와 세로로 나누어 양쪽 45%영역을 버리고 중간 10% Blob테두리위치를 구해서 직선을 그어 만나는 지점을 구합니다.

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221341913377&categoryNo=0&parentCategoryNo=0&viewDate=&currentPage=3&postListTopCurrentPage=1&from=section&userTopListOpen=true&userTopListCount=30&userTopListManageOpen=false&userTopListCurrentPage=3)

Line Fitting을 이용하여 가로세로 만나는 지점을 이용하는방법은 qroisimulator에서 소개할 예정입니다

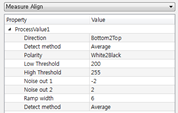
[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221341913377&categoryNo=0&parentCategoryNo=0&viewDate=&currentPage=3&postListTopCurrentPage=1&from=section&userTopListOpen=true&userTopListCount=30&userTopListManageOpen=false&userTopListCurrentPage=3)

**[Two Point Align절차]**  
1. git clone을 이용해서 qroilib program을 download합니다    
   git clone <https://github.com/jonglockyoon/qt-qroilib>   
  
2. qroilibalign.pro를 실행합니다.   
   QT 개발환경이 설치되어 있어야합니다. winlib에는 64bit lib 및 dll들이 있습니다.   
   windows는 관련 winlib아래의 dll들을 실행파일 위치 또는 PATH로 지정된 위치에 copy해야합니다.   
  
3. alignimage.png이미지를 로드합니다.   
   file -> read image -> alignimage.jpg   
  
4. "Create Rectangle ROI"을 클릭후 모서리 위치에서 왼쪽 Mouse Button을 클릭한채 Dragging합니다.   
   생성된 ROI를 Double Click합니다.   
   "Twopoint Align" ComboBox Item을 선택합니다.   
   Corner에 UpperRight 또는 BottomRight를 선택합니다.   
   Method는 Corner로 둡니다.   
      
   두번찌 ROI도 위 절차대로 생성합니다.   
      
5. "Two Point Image Align" 버턴을 클릭해서 Align이 정상적으로 실행되는지 확인합니다.

**5.2) Measure Align**

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221341913377&categoryNo=0&parentCategoryNo=0&viewDate=&currentPage=3&postListTopCurrentPage=1&from=section&userTopListOpen=true&userTopListCount=30&userTopListManageOpen=false&userTopListCurrentPage=3)

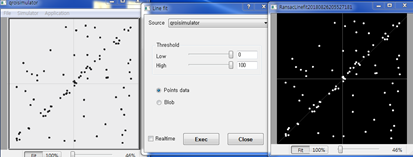
세개의 얼라인 ROI를 이용해서 수직 또는 수평으로 이미지를 Rotate시킨후 X,Y 이미지 Shift를 시킵니다.

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221341913377&categoryNo=0&parentCategoryNo=0&viewDate=&currentPage=3&postListTopCurrentPage=1&from=section&userTopListOpen=true&userTopListCount=30&userTopListManageOpen=false&userTopListCurrentPage=3)

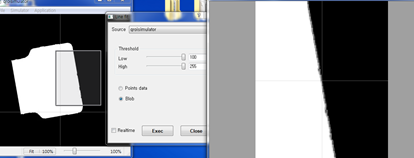
위 ROI는 수평 한개, 수직 두개 총 세개의 ROI이며 CImgAlign::MeasureAlignImage()함수에서 처리됩니다.  
   
한 개의 Measure Align ROI만 사용하면 수평 또는 수직으로 이미지를 이동하는 작업만 실행합니다.  
  
  
**아래 사이트에 qroilib를 이용한 align program이 있습니다.   
[https://github.com/JonglockYoon/QT-qroilib/tree/master/qroialign](https://github.com/JonglockYoon/QT-qroilib/tree/master/qroialign" \t "_blank)**

### 7. qroilib Simulator (linefitting)

Qroilib simulator중 cvFitLine Test 부분입니다.  
   
7.1. DialogLinefit::**ExecRansacLinefitPoints()**

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&Redirect=View&logNo=221346560824&categoryNo=12&isAfterWrite=true&isMrblogPost=false&isHappyBeanLeverage=true&contentLength=53212&redirect=View&widgetTypeCall=true&directAccess=false)

**File -> read image -> points.jpg 를 로드합니다  
전체 이미지 영역을 드래깅해서 rectangle ROI를 생성합니다  
Low, High값을 변경합니다  
Exec 버턴으로 여러 점들중 가징 이상적인 라인을 그려줍니다.**  
  
CBlobResult blobs;  
blobs = CBlobResult(iplImg, nullptr, 0);  
int blobCount = blobs.GetNumBlobs();  
if (blobCount <= 0)  
    return;  
  
CvPoint\* points = (CvPoint\*)malloc(blobCount \* sizeof(CvPoint));  
for (int i = 0; i < blobCount; i++) // 각 점들의 중심점을 구합니다.  
{  
   CBlob \*currentBlob = blobs.GetBlob(i);  
   double m00 = currentBlob->Moment(0,0);  
   double m01 = currentBlob->Moment(0,1);  
   double m10 = currentBlob->Moment(1,0);  
  
   CvPoint p(m10/m00, m01/m00);  
   points[i].x = p.x;  
   points[i].y = p.y;  
}  
  
//cvFitLine으로 최적의 직선을을 찾습니다.  
CvMat pointMat = cvMat(1, blobCount, CV\_32SC2, points);  
float line[4];  
cvFitLine(&pointMat, CV\_DIST\_L1, 1, 0.001, 0.001, line);  
  
float d,t;  
  
d = sqrt(line[0] \* line[0] + line[1] \* line[1]);  
line[0] /= d;  
line[1] /= d;  
t = tmp->width + tmp->height;  
  
// 영상에 선을 그립니다.  
int x0= line[2]; // 선에 놓은 한 점  
int y0= line[3];  
int x1= x0 - t\*line[0]; // 기울기에 길이를 갖는 벡터 추가  
int y1= y0 - t\*line[1];  
int x2= x0 + t\*line[0];  
int y2= y0 + t\*line[1];  
cvLine( tmp, CvPoint(x1,y1), CvPoint(x2,y2), CV\_RGB(128,128,128), 1, 8 );  
  
**7.2. DialogLinefit::ExecRansacLinefitBlob()**

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&Redirect=View&logNo=221346560824&categoryNo=12&isAfterWrite=true&isMrblogPost=false&isHappyBeanLeverage=true&contentLength=53212&redirect=View&widgetTypeCall=true&directAccess=false)

**File -> read image -> alignimage.jpg 를 로드합니다  
경사진 영역을 드래깅해서 rectangle ROI를 생성합니다  
Low, High값을 변경합니다  
Exec 버턴으로 경사진 라인을 구해서 그려줍니다.  
  
   
    CImgProcBase base;  
    base.FilterLargeArea(iplImg); // 이미지에서 가장 큰 Blob한 개만 남깁니다.  
   
    cvSmooth(iplImg, iplImg, CV\_GAUSSIAN,3,3);  
    cvCopy(iplImg, tmp);  
   
    // Blob의 테두리를 구합니다.  
    CvMemStorage\* storage = cvCreateMemStorage(0);  
    CvSeq\* contours = 0;  
    cvFindContours(iplImg, storage, &contours, sizeof(CvContour), CV\_RETR\_TREE, CV\_CHAIN\_APPROX\_SIMPLE);  
   
    float line[4];  
    while(contours)  
    {  
        // CV\_DIST\_HUBER : Distance types for Distance Transform and M-estimators  
        // 0   
        // 0.01 : 정확도  
        cvFitLine(contours, CV\_DIST\_HUBER, 0, 0.001, 0.001, 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;  
   
        // 영상에 선을 그립니다.  
        int x0= line[2]; // 선에 놓은 한 점  
        int y0= line[3];  
        int x1= x0 - t\*line[0]; // 기울기에 길이를 갖는 벡터 추가  
        int y1= y0 - t\*line[1];  
        int x2= x0 + t\*line[0];  
        int y2= y0 + t\*line[1];  
        cvLine( tmp, CvPoint(x1,y1), CvPoint(x2,y2), CV\_RGB(128,128,128), 1, 8 );  
   
        contours = contours->h\_next;  
    }  
  
전체 Source Code는 다음에서 볼수 있습니다.**[**https://github.com/JonglockYoon/QT-qroilib/tree/master/qroisimulator**](https://github.com/JonglockYoon/QT-qroilib/tree/master/qroisimulator)

### 8. qroilib Simulator (blob)

Qroilib simulator중 Blob Test 부분입니다.  
여기에서 사용된 Blob 함수들은 https://github.com/OpenCVBlobsLib 에서 가져온것입니다.  
   
이미지 처리에서 Blob은 매우 많이 사용됩니다.  
이치화, 모폴로지 단계를 거치고 나면 각 이미지는 분석이 가능한 깨끗한 이치화 이미지가 될 것입니다.  
잡음이 없는 깨끗한 이미지를 Blob으로 영역크기, 영역의 가로 세로 길이, 영역의 사각형, 둘레길이,중심점 등 다양한 Blob정보를 얻을수 있습니다.  
   
**Blob이란?**  
Binary Large Object의 약자로, 이진 이미지의 연결된 픽셀 그룹을 말합니다.  
“Large”의 의미는 특정 크기의 객체만 관심대상이고, “Small”이진 객체는 잡음을 나타냅니다.  
다음 그림은 잡음을 제거한 Blob만 추출하였습니다.

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221347271028&categoryNo=12&parentCategoryNo=0&viewDate=&currentPage=1&postListTopCurrentPage=&from=postList&userTopListOpen=true&userTopListCount=5&userTopListManageOpen=false&userTopListCurrentPage=1)

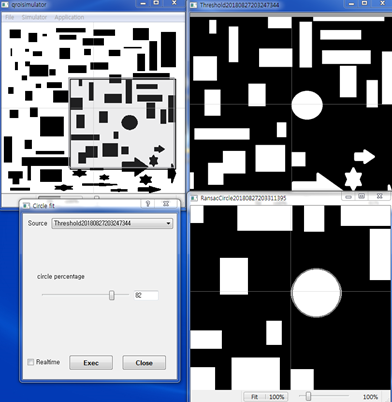
이 과정은 모폴리지 Open기능을 이용하였으며 결과로 2개의 Blob이 추출되었습니다.  
(<https://blog.naver.com/jerry1455> 참조)  
   
**Blob응용**  
‘7. qroilib Simulator (linefitting)’에서 이미 Blob을 이용하여 각 점들의 중심점을 구한 예를  
보였습니다.  
  
CBlobResult blobs;  
blobs = CBlobResult(iplImg, nullptr, 0);  
int blobCount = blobs.GetNumBlobs();  
if (blobCount <= 0)  
return;  
 CvPoint\* points = (CvPoint\*)malloc(blobCount \* sizeof(CvPoint));  
for (int i = 0; i < blobCount; i++) // 각 점들의 중심점을 구합니다.  
{  
                  CBlob \*currentBlob = blobs.GetBlob(i);  
                  double m00 = currentBlob->Moment(0,0);  
                  double m01 = currentBlob->Moment(0,1);  
                  double m10 = currentBlob->Moment(1,0);  
                  CvPoint p(m10/m00, m01/m00);  
                  points[i].x = p.x;  
                  points[i].y = p.y;  
}  
  
이처럼 Blob은 이미지 처리에서 필수 기능이라고 볼수 있습니다.  
CBlob \*currentBlob = blobs.GetBlob(i);   
double dArea = currentBlob->Area(); // 추출된 Blob의 Pixel갯수, 즉 영역크기를 얻을수 있습니다.  
currentBlob->FillBlob(grayImg, CVX\_WHITE); // Blob을 특성 color로 채움니다.  
CvRect rect = currentBlob->GetBoundingBox(); // Blob의 사각형 영역을 가져옵니다.  
CvBox2D box = currentBlob->GetEllipse(); // 경계 타원 정보를 가져옵니다.  
\* 많이 사용되는 API만 나열합니다  
추출된 Blob에서 위 와 같은 많은 정보들을 얻을수 있으며, 이와 같은 정보를 이용하여  
다양한 결과를 도출할수 있습니다.  
  
**VoronoiThinner**

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221347271028&categoryNo=12&parentCategoryNo=0&viewDate=&currentPage=1&postListTopCurrentPage=&from=postList&userTopListOpen=true&userTopListCount=5&userTopListManageOpen=false&userTopListCurrentPage=1)

cv::Mat1b m = cv::Mat1b(cv::cvarrToMat(grayImg).clone());  
 IplImage \*trans = nullptr;  
bool ok = thinner.thin(m, IMPL\_GUO\_HALL\_FAST, false);  
if (ok) {  
                  IplImage iplImage = thinner.get\_skeleton(); // <https://github.com/arnaud-ramey/voronoi> 참조.  
                   trans = &iplImage;  
}  
 if (trans)  
                  cvCopy(trans, grayImg);  
else  
                  cvZero(grayImg);  
m.release();  
  
Blob과는 거리가 있지만 각 Blob들의 골격(skeleton)을 추출하는 기능입니다.  
   
**전체 Source Code는 다음에서 볼수 있습니다.**  
<https://github.com/JonglockYoon/QT-qroilib/tree/master/qroisimulator>

### 9. qroilib Simulator (circlefit)

Qroilib simulator의 circle fit 부분입니다.  
File -> ReadImage -> testImage.png를 load합니다.  
아래처럼 circle이 포함된 영역을 드래깅해서 rectangle ROI를 생성합니다.  
Simulator -> Threshold 에서 Threshold 0 ~ 100의 입력값으로 이치화를 합니다.  
Simulator -> Circle Image 에서 이치화결과를 Source로 선택하고 circle percentage를 80으로 입력합니다.  
아래와 같이 RANSAC circle 의 결과로 80% 이상의 원형도를 가진 Blob을 추출할수 있습니다.

[](https://blog.naver.com/PostView.nhn?blogId=jerry1455&logNo=221347932308&categoryNo=12&parentCategoryNo=0&viewDate=&currentPage=1&postListTopCurrentPage=1&from=postList&userTopListOpen=true&userTopListCount=5&userTopListManageOpen=false&userTopListCurrentPage=1)

<https://stackoverflow.com/questions/26222525/opencv-detect-partial-circle-with-noise> 참조.  
CBlobResult blobs = CBlobResult(grayImgIn, nullptr, 0);  
int nBlobs = blobs.GetNumBlobs();  
for (int i = 0; i < nBlobs; i++)     // 여러개의 Blob이 있을때 한개씩 뽑아서 RANSAC을 돌린다.  
{  
        CBlob \*p = blobs.GetBlob(i);  
        CvBox2D d = p->GetEllipse();  
        cvZero(grayImg);  
        p->FillBlob(grayImg, CVX\_WHITE);          // Draw the filtered blobs as white.  
…  
}  
위 코드는 이미지 중 Blob한 개씩 뽑아내는 과정입니다.  
cv::Mat canny;  
cv::Mat gray = cv::cvarrToMat(grayImg);  
cv::Mat mask;  
cv::Canny(gray, canny, 100, 20);  
mask = canny;  
std::vector<cv::Point2f> edgePositions;  
edgePositions = getPointPositions(mask);  
cv::Mat dt;  
cv::distanceTransform(255 - mask, dt, CV\_DIST\_L1, 3);  
int maxNrOfIterations = edgePositions.size();  
RANSACIRCLE rbest;  
rbest.cPerc = 0.0;  
rbest.center.x = 0;  
for (int its = 0; its < maxNrOfIterations; ++its){  
    unsigned int idx1 = rand() % edgePositions.size();  
    unsigned int idx2 = rand() % edgePositions.size();  
    unsigned int idx3 = rand() % edgePositions.size();  
    // we need 3 different samples:  
    if (idx1 == idx2) continue;    if (idx1 == idx3) continue;     if (idx3 == idx2) continue;  
    // create circle from 3 points:  
    cv::Point2f center; float radius;  
    getCircle(edgePositions[idx1], edgePositions[idx2], edgePositions[idx3], center, radius);  
    // inlier set unused at the moment but could be used to approximate a (more robust) circle from alle inlier  
    std::vector<cv::Point2f> inlierSet;  
    //verify or falsify the circle by inlier counting:  
    float cPerc = verifyCircle(dt, center, radius, inlierSet);  
    // update best circle information if necessary  
    QRect rect = QRect(QPoint(center.x-radius, center.y-radius), QPoint(center.x+radius, center.y+radius));  
    // circle 영역안에 blob center가 포함되어야한다.  
    if (rect.contains(d.center.x, d.center.y))    {  
        RANSACIRCLE rclc;  
        rclc.cPerc = cPerc;  
        if (rbest.cPerc < rclc.cPerc) {  
            rbest.cPerc = rclc.cPerc;            rbest.center = center;             rbest.radius = radius;  
        }  
    }  
}  
if (rbest.center.x > 0) {  
    if (rbest.cPerc > bestCirclePercentage) {  
        vecRansicCircle.push\_back(rbest);  
    }  
}  
뽑아낸 한 개의 Blob씩 RANSAC 알고리즘을 이용해서 circle percenrage를 구합니다.  
  
   
**전체 Source Code는 다음에서 볼수 있습니다.**  
<https://github.com/JonglockYoon/QT-qroilib/tree/master/qroisimulator>