OpenCV 보고서

1) OpenCV란

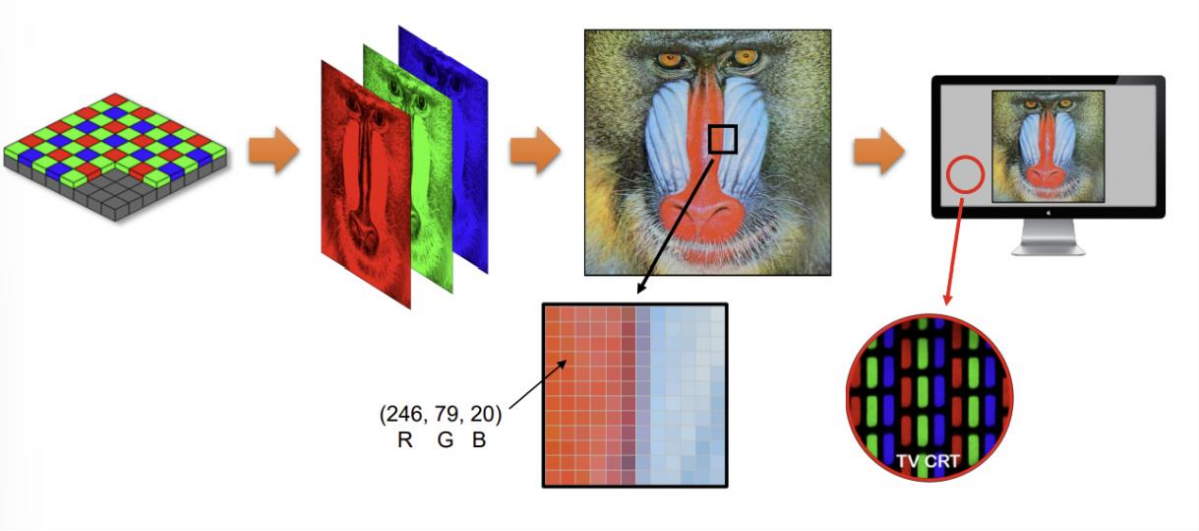
OpenCV는 Open Source Computer Vision의 약자로 영상 처리에 사용할 수 있는 오픈 소스 라이브러리이다. 컴퓨터에서 사람의 눈과 같은 역할을 할 수 있도록 해주며, 카메라 어플 등에서도 사용되고 있다. OpenCV는 오픈소스임에도 BSD(Berkely Software Distribution)라이선스를 따라, 상업적인 목적으로 사용 가능하다. 참고로 BSD라이선스란 간단히 소프트웨어의 무료 사용, 수정 및 배포를 허용하는 오픈 소스 라이선스로 많은 개발자와 회사가 이 라이선스를 이용해 사용자가 작업에 접근 가능하도록 한다.

2) Computer Vision

컴퓨터 비전은 카메라로 얻는 영상들을 모두 컴퓨터 비전이라고 한다. 인간은 빛의 반사로 눈에 들어오는 정보를 구성해 자동으로 초점과 색이 맞춰지며 사물이 비친다. 하지만 컴퓨터는 0과 1로 구성된 일련의 숫자 행렬로 사진과 영상 등의 형태를 인식한다.

이 인식 과정 즉 컴퓨터 비전의 주요 원칙은 다음 네 가지와 같다. -이미지 획득 : 이미지를 카메라, 녹화영상, 사진 등 다양한 소스를 통해 획득. -이미지 처리 : 원시 이미지 데이터를 컴퓨터가 분석할 수 있는 형태로 변환하는 작업을 포함하고, 영상 향상, 노이즈 감소 등을 포함한다. -처리된 영상에서 형상 추출 : 형상의 특징(가장자리, 모서리, 모양 등)을 통해 이미지 구별. -형상을 통해 이미지를 객체 등의 범주로 분류.

영상(Image)은 기본적으로 픽셀이 바둑판 모양의 격자에 나열되어 있는 형태인 데이터이다. 여기서 픽셀(Pixel-Picture Element)은 화소라고도 하며 영상을 이루는 가장 기본적인 단위이다. 우리가 흔히 부르는 동영상은 하나의 이미지를 의미하는 영상을 여러 개 연속적으로 재생하는 것을 뜻한다.

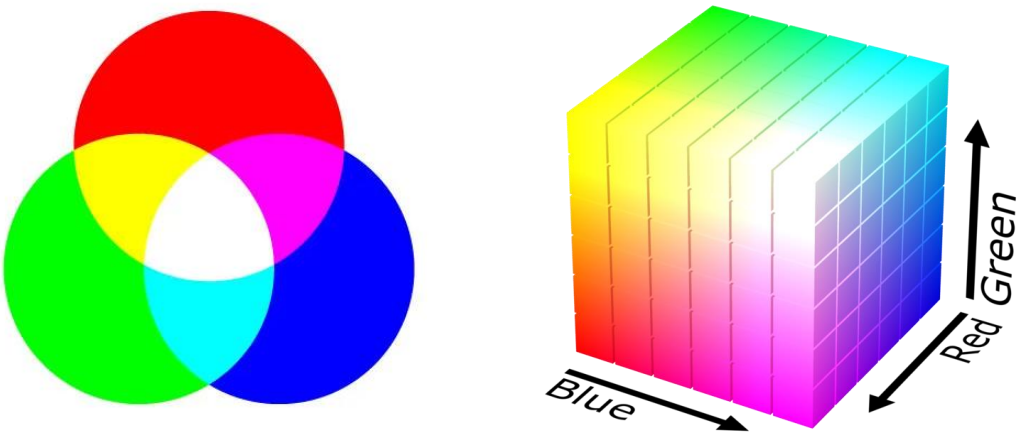


추가적으로 픽셀은 강도(Intensity)와 채널(Channel)값을 가진다. 강도는 0~255, 채널은 R, G, B로 구성된다. 이를 통해 채널이 하나인 그레이 스케일 이미지 또는 채널이 세 개인 컬러 이미지를 구성할 수 있다. 이미지는 픽셀 좌표의 강도 값을 반환하는 함수로 볼 수 있어 강도 함수(Intensity function)라고 한다.

3) 픽셀의 정보 표현 방법

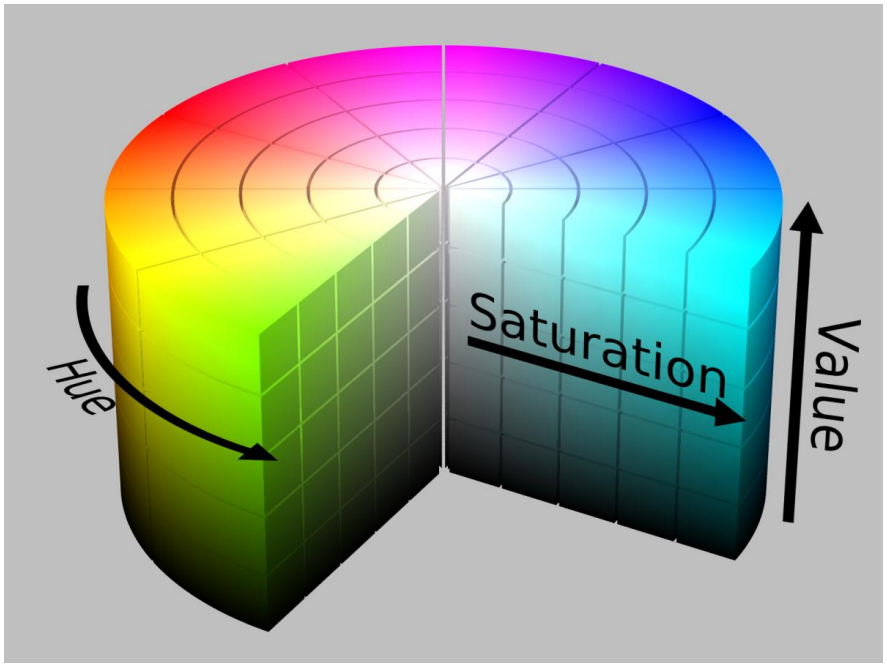
1. RGB

RGB는 빛의 삼원색인 Red, Green, Blue색으로 구성되어 있고 각 색 모두 0~255의 숫자로 표현해 다양한 색을 만들 수 있다. (참고로 OpenCV에서는 변환없이 초기 색 설정에서는 BGR순으로 입력해야 할 것이다)



2. HSI(HSV)

HSV는 RGB와는 다르게 색을 나타내는 방법으로 색조(Hue), 채도(Saturation), 명도(Value / Intensity)로 구성되어 있다. 각 채널의 OpenCV에서의 값 범위는 H는 0~180, S는 0~255, V는 0~255이다. 여기서 H는 기본적으로 360(각도)이지만 OpecnCV 이미지 변수들은 8bit로 설정되어 있어 최대 255까지만 표현할 수 있기 때문에 360/2인 180이 최대값이 된다. 추가적으로 OpenCV 또는 비전 개발 시 RGB 이미지에서 HSV 이미지로 변환하는 경우가 많은데 이는 HSV 이미지에서 H가 순수한 색 정보를 가지고 있기 때문이다. (+RGB는 빛에 의한 명도를 판단하지 못 하지만 HSV는 가능)



3. Grayscale

Grayscale은 간단히 컬러로 되어있는 사진이나 동영상을 흑백(화면)으로 바꾸는 것을 말한다. 다시 말해, 색상정보를 제외하고 명암정보만을 남기는 것을 의미한다. 그렇기 때문에 빛의 강도를 기준으로 검은색과 흰색 사이 0부터 255까지의 숫자를 이용해 표현한다. 추가로 RGB에 비해 이미지가 가지는 정보량이 1/3 가량 적기에 이미지 연산 시 상대적으로 이득을 볼 수 있다.

4. Binary

Binary 즉 이진화는 말 그대로 이미지를 이진화 하는 것을 말한다. 정확히는 이미지 또는 영상 속 특정 값을 경계로 강도(Intensity)를 흑(0)과 백(1 / 255)으로만 구분지어 두 값으로만 표현한 이미지를 말한다. 당연하게도 앞선 어느 이미지 정보 표현 방법보다 이미지 연산 시 이득을 볼 수 있다.

4) 필터

블러링 기법 – 블러링(bluring)은 초점이 맞지 않는 사진처럼 영상을 부드럽게 만드는 필터링 기법이다. 이는 인접한 픽셀 간의 픽셀 값 변화를 크지 않게 만들어 부드러운 느낌을 받게 한다.

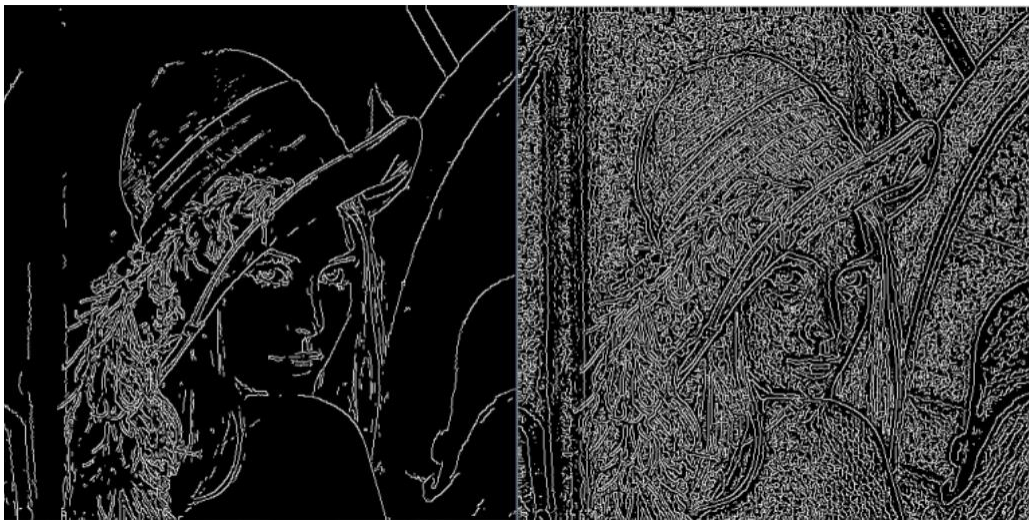
평균값 필터 – 블러링 기법 필터 중 가장 단순하고 구현하기 쉬운 필터이다. 특정 픽셀과 그 주변 픽셀들의 산술평균을 결과 영상 픽셀 값에 설정하는 필터이다.

가우시안 필터 – 가우시안 분포 함수를 근사하여 생성한 필터를 사용하는 필터링 기법이다. 여기서 가우시안 분포는 평균을 중심으로 좌우 대칭의 종 모양을 갖는 확률 분포이다. 이 분포 모양에 따라 필터 행렬(가중치)은 중앙부에서 큰 값을, 사이드로 갈수록 작은 값을 가진다. 가우시안 커널 표준 편차인 sigma에 따라 우리가 볼 때 흐려지는 정도 및 픽셀의 블러처리 정도가 강해지는 것을 볼 수 있다.

5) 함수 설명

- Mat : 이미지 데이터를 저장하는 형식(ex : cv::Mat image;)

- Canny : 외곽선/경계선 추출(ex : cv::Canny(src, dst, 50, 200, 30); 여기서 매개변수들은 순서대로 각각 입력 이미지, 결과영상이 저장될 이미지, 낮은 경곗값, 높은 경곗값, 커널 크기) 아래는 커널 크기 3과 5의 사진이다.



- Inrange : 이미지 이진화(ex : cv::inrange(src, cv::Scalar(minH, minS, minV), cv::Scalar(maxH, maxS, maxV), dst); 여기서 매개변수들은 순서대로 각각 입력 이미지, HSV하한값, HSV상한값, 결과가 저장될 이미지)

- Resize : 이미지의 크기 조정(ex : cv::resize(src, dst, cv::Size(width, height)); 여기서 매개변수들은 순서대로 각각 입력 이미지, 결과가 저장될 이미지, 넓이/높이가 설정된 대로 크기 변경)

- Rectangle : 이미지에서 사각형을 그릴 수 있는 함수(ex : cv::rectangle(dst, cv::Rect(x, y, width, height), cv::Scalar(H, S, V), L); 여기서 매개변수들은 각각 사각형이 그려질 이미지, 사각형의 위치 및 넓이 높이, 사각형의 색, 선의 굵기)

- HoughLinesP : 이진화 이미지에서 선을 검출(ex : cv::HoughLinesP(src, dst, 1, CV_PI/180, 50, 30, 80); 여기서 각각의 매개변수는 이진화된 타겟 이미지, 직선 속성 변수(vector<Vec4i>타입의 어레이 변수), r방향변위값(경계값), 회전방향각도(경계값), 최소픽셀수, 직선 최소 길이, 픽셀 간 허용 최대 차이 참고자료 -> https://fwanggu-lee.tistory.com/27)

- cvtColor : 이미지의 색 표현 방식 변환(ex : cv::cvtColor(src, dst,

cv::COLOR_BRG2RGB); 여기서 각 매개변수들은 입력 이미지, 색 변환을 적용시켜 저장할 이미지, 색 표현 방식)

- Erode : 이미지 침식(ex : cv::erode(src, dst, cv::Mat(), cv::Point(-1, -1), 3); 여기서 각 매개변수들은 입력 이미지, 변환 후 저장될 이미지, 구조요소/Mat()이면 3X3 사각형 구조 요소를 사용, 고정점 위치/(-1, -1)이면 중앙을 고정점으로 사용)
- Dilate : 이미지 팽창(ex : cv::dilate(src, dst, cv::Mat(), cv::Point(-1, -1), 3); 여기서 각 매개변수들은 위 Erode 요소와 같다)

6) OpenCV 설정

OpenCV 사용을 위해 CMakeLists.txt와 package.xml을 설정해준다. CMakeLists.txt에서는 find_package에서 cv_bridge를 추가해주고, find_package(Qt5 REQUIRED COMPONENTS Widgets Network Gui Core)과 같이 Widgets 뒤에 Network를 써 준다. find_package(OpenCV 4.5.2 REQUIRED)를 적어주고 include_directories에서 OpenCV와 관련된 ${Qt5Network_INCLUDE_DIRS} ${OpenCV_INCLUDE_DIRS}를 추가해준다. 마지막으로, target_link_libraries에 Qt5::Network를 추가해주며 ${OpenCV_LIBRARIES} ${OpenCV_LIBS}를 추가해준다. Package.xml에는 find_package에 써 주었던 것들을 추가해서 각각 넣어주면 된다. OpenCV 사용 라이브러리는 다음과 같다.

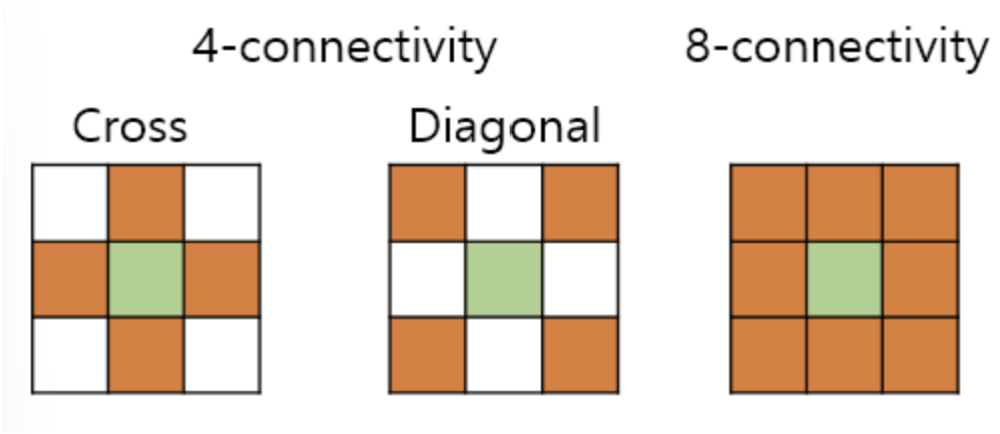
```
#include <opencv2/opencv.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/core/mat.hpp>
#include <opencv2/imgcodecs.hpp>
#include <cv_bridge/cv_bridge.h>
#include <image_transport/image_transport.h>
```

7) 라벨링

뭉쳐있는 픽셀들을 모아 한 객체로 인식하도록 하는 라벨링에 대해 알아보자.

여기서 내가 사용한 라벨링 방식 및 코드는 다음 사이트를 참고한 것이다 (https://m.blog.naver.com/dorergiverny/223075828736).

여기서는 이진화 기법을 이용해 배경과 전경을 구분하고 여기서 라벨링하고자 하는 대상을 전경으로 만들어 라벨링하는 방식이다. 라벨링 방법에는 크게 3가지가 있다. 4방향 연결과 8방향 연결, 4방향 연결 중에서도 십자 연결/대각선 연결이 있다. 이런 방식으로 픽셀들을 연결하는 것이다.



- Int connectedComponentsWithStats(src, label, stats, centroids, connectivity, ltype) : OpenCV에서 위 방식으로 라벨링하는 함수(ex : int num = cv::connectedComponents(src, label, stats, centroids); 여기서 각 매개변수는 입력 이미지(gray or binary이미지), 라벨링된 결과 저장 이미지, 각 라벨 영역 통계 정보를 잠은 행렬, 각 라벨 영역의 무게중심 좌표를 담은 행렬, 연결 방식, 출력 영상 타입)

8) 콘 위치 판별 라벨링 코드 설명

```
67  void MainWindow::ImgRec()
68  {
69    // OpenCV 이미지를 Qt 이미지로 변환
70    IMG = udp_.MatImgRcv(img, PORT, my_IP, *receivesocket);
71    cv::resize(IMG, IMG, cv::Size(430, 230));
72    cv::GaussianBlur(IMG, IMG, cv::Size(3,3), 0); //IMG에 적용 후 ROI에 저장.
73    cv::cvtColor(IMG, IMG, cv::COLOR_BGR2RGB); //ROI이미지를 BRG->RGB 변환
74
75    IMG2=IMG.clone();
76    //필터+그레이스케일
77    cv::cvtColor(IMG, BIN, cv::COLOR_BGR2GRAY);
78
79    //일반 영상
80    Usb_Cam();
81    //이진화 영상
82    Binary();
83    //주황콘 라벨링
84    OrangeLabeling();
85    //라임콘 라벨링
86    LimeLabeling();
87    //결과 화면
88    Find_object();
89    //판단
90    judge();
91
92  }
```

먼저 udp헤더파일 내 matimgrcv함수를 통해 IMG에 카메라 이미지를 받아온다. matimgrv 매개변수는 순서대로 이미지를 저장해 리턴해줄 img변수, 영상을 받아오기 위한 연결 포트, 받는 쪽의 IP, 받는 데이터 소켓이다.

그렇게 받아온 IMG를 resize함수를 통해 크기 조절을 해주고 gaussianblur함수를 통해 가우시안 필터를 적용시키며 cvtcolor함수를 통해 이미지의 색 표현 방식을 변환해준다.

다음으로 IMG2, BIN Mat객체를 만들어 IMG2에는 IMG를 복사하고 BIN에는 그레이스케일을 적용시킨 IMG를 저장한다. 마지막엔 각각의 진행을 할 함수들이다.

함수들에 대해 차례로 설명해보겠다.

```
94  void MainWindow::Usb_Cam()
95  {
96    QImage QImage(IMG.data, IMG.cols, IMG.rows, IMG.step, QImage::Format_RGB888);
97    ui.Cam_1->setPixmap(QPixmap::fromImage(QImage));
98  }
```

먼저 UsbCam함수는 간단히 실시간으로 받고 있는 영상의 IMG를 RGB색 표현 방식으로 QPixmap을 이용해 label Cam_1에 띄우고 있는 함수이다.

```
151  void MainWindow::Binary()
152  {
153    cv::cvtColor(IMG, BIN, cv::COLOR_BGR2RGB); //ROI이미지를 BRG->HSV 변환
154    cv::cvtColor(BIN, BIN, cv::COLOR_RGB2HSV);
155    cv::Scalar lower_white = cv::Scalar(minH, minS, minV); //최소 범위 각각 H/S/V
156    cv::Scalar upper_white = cv::Scalar(maxH, maxS, maxV); //최대 범위 HSV
157    cv::inRange(BIN, lower_white, upper_white, BIN); //범위에 따라 검/흰(0/255) 출력
158
159    QImage QImage(BIN.data, BIN.cols, BIN.rows, BIN.step, QImage::Format_Indexed8);
160    ui.Cam_4->setPixmap(QPixmap::fromImage(QImage));
161  }
```

Binary함수에 대해 설명해보자면 함수 이름 그대로 이진화 작업을 하는 함수이다. cvtcolor함수를 통해 BRG에서 RGB로, RGB에서 HVS로 변환하는 과정을 거친다. 그리고 Scalar함수를 통해 HSV 각 요소의 최댓값, 최솟값을 클래스 변수로 놓아 슬라이더를 통해 조절할 수 있도록 하였다.

이렇게 각 HSV에 넣어진 값들과 inrange함수를 통해 위 범위 내에 있는 색은 모두 흰색 아니면 검은색으로 이진화시켜준다. 이렇게 만들어진 이진화 이미지를 Cam_4에 띄우는데, 중요한점은 QPixmap의 QImage방식이 RGB가 아닌 Indexd8 이진화라는 것이다.

```
196  cv::Mat MainWindow::Region_of_interest(cv::Mat& image, cv::Point* points)
197  {
198    cv::Mat img_mask = cv::Mat::zeros(image.rows, image.cols, CV_8UC1);
199    const Point* ppt[1] = {points};
200    int npt[] = {4};
201
202    cv::fillPoly(img_mask, ppt, npt, 1, Scalar(255, 255, 255), LINE_8);
203
204    cv::Mat img_masked;
205    bitwise_and(image, img_mask, img_masked);
206
207    QImage QImage(img_mask.data, img_mask.cols, img_mask.rows, img_mask.step, QImage::Format_Indexed8);
208    ui.Cam_3->setPixmap(QPixmap::fromImage(QImage));
209
210    return img_masked;
211  }
```

Region_of_interest함수에서는 관심영역을 설정해주고 이를 제외한 나머지를 고려하지 않도록 한다. Mat::zeros(); 함수를 통해 이미지 데이터를 저장할 행렬을 Mat으로 만들어주며 새로운 행렬 생성 시 모든 원소 값이 0으로 초기화해준다. const Point*~ 부분은 points라는 배열을 포함하는 포인터 배열 ppt를 생성하는 것이다. 참고로 points에는 뒤에 나오겠지만 다각형의 꼭짓점 좌표를 담고 있다. Int npt[] = {4}; 부분은 다각형을 형성하는 꼭짓점의 개수를 나타내는 배열을 생성하는 코드이다. 이런 설정을 통해 cv::fillPolt함수에 전달되었을 때 하나의 다각형

이 정의되어 그려진다. 그리고 bitwise_and(); 함수는 OpenCV에서 제공하는 비트 단위의 AND연산 수행 함수이다. 때문에 이 코드에선 image 이미지 행렬과 img_mask 이미지 행렬을 AND연산하여 img_masked에 저장하는 방식이다. 결과 Canny에 의해 외곽선이 그려진 이미지에 AND연산을 통해 중앙 흰선만 보이게 되는 모습이 보여질 것이다.

```
164  {
165    //윤곽선 이미지 CAN
166    cv::Canny(IMG, CAN, 50, 200, 3);
167
168    points[0] = cv::Point(30, 125);
169    points[1] = cv::Point(30, 105);
170    points[2] = cv::Point(340, 125);
171    points[3] = cv::Point(340, 145);
172
173    CAN = Region_of_interest(CAN, points);
174
175    std::vector<Vec4i> lines;
176    cv::HoughLinesP(CAN, lines, 1, CV_PI/180, 50, 30, 80);
177    for(size_t i = 0; i < lines.size(); i++){
178      cv::Vec4i l = lines[i];
179      cv::line(IMG2, cv::Point(l[0], l[1]), cv::Point(l[2], l[3]), cv::Scalar(0,0,255), 2, cv::LINE_AA);
180    }
181
182    QImage QImage(IMG2.data, IMG2.cols, IMG2.rows, IMG2.step, QImage::Format_RGB888);
183    ui.Cam_2->setPixmap(QPixmap::fromImage(QImage));
184  }
```

이 Find_object함수에서는 외곽선 그리기, 라벨링 등의 결과값들이 카메라에 그려지도록 한 함수이다. 먼저 IMG에 Canny(외곽선)적용 후 CAN 이미지에 저장하였고 Point points 객체들을 통해 위에서 이 바로 위에서 설명한 관심영역 좌표를 설정해준다. 그렇게 위에서 설명한 대로 Region_of_interest함수를 통해 원하는 부분만 외곽선이 그려진 이미지가 CAN 이미지에 저장된다. 이후 HoughLinesP함수와 line 등을 이용해 원하는 부분의 외곽선을 선명하게 그려준다.

```
100  void MainWindow::OrangeLabeling()
101  {
102    cv::cvtColor(IMG, ORA, cv::COLOR_BGR2RGB); //ROI이미지를 BRG->HSV 변환
103    cv::cvtColor(ORA, ORA, cv::COLOR_RGB2HSV);
104    cv::Scalar lower_white = cv::Scalar(90, 63, 127); //최소 범위 각각 H/S/V
105    cv::Scalar upper_white = cv::Scalar(255, 255, 255); //최대 범위 HSV
106    cv::inRange(ORA, lower_white, upper_white, ORA); //범위에 따라 검/흰(0/255) 출력
107
108    QImage QImage(ORA.data, ORA.cols, ORA.rows, ORA.step, QImage::Format_Indexed8);
109    ui.Cam_5->setPixmap(QPixmap::fromImage(QImage));
110
111    cv::Mat label, stats, centroids;
112
113    int num = cv::connectedComponentsWithStats(ORA, ORA, stats, centroids, 8, CV_32S);
114
115    for (int i = 1; i < num; i++) {
116        Ox = static_cast<int>(stats.at<int>(i, cv::CC_STAT_LEFT));
117        Oy = static_cast<int>(stats.at<int>(i, cv::CC_STAT_TOP));
118        int width = static_cast<int>(stats.at<int>(i, cv::CC_STAT_WIDTH));
119        int height = static_cast<int>(stats.at<int>(i, cv::CC_STAT_HEIGHT));
120
121        cv::rectangle(IMG2, cv::Rect(Ox, Oy, width, height), cv::Scalar(255, 127, 0), 4);
122    }
123  }
```

OrangeLabeling()함수는 오렌지콘을 라벨링하는 함수이다. 113까지의 코드는 앞서 라벨링 방식 등에서 설명한 것과 같고, for문 내에 있는 코드는 113줄에서의 라벨링을 통해 라벨링된 객체의 왼쪽 위 모서리부분에서의 좌표와 객체 라벨링의 넓이 높이를 알아내어 준다. 여기서 나는 이 좌표를 기준으로 콘의 위치를 판별하고자 하였기에 x, y좌표를 클래스 변수로 두어 어디에서든 접근할 수 있도록 하였다. 참고로 아래 코드 사진과 같이 라임색 콘 라벨링 또한 오렌지색 콘 라벨링과 방식이 같다.

```
125  void MainWindow::LimeLabeling()
126  {
127    cv::cvtColor(IMG, LIM, cv::COLOR_BGR2RGB); //ROI이미지를 BRG->HSV 변환
128    cv::cvtColor(LIM, LIM, cv::COLOR_RGB2HSV);
129    cv::Scalar lower_white = cv::Scalar(0, 42, 127); //최소 범위 각각 H/S/V
130    cv::Scalar upper_white = cv::Scalar(90, 255, 255); //최대 범위 HSV
131    cv::inRange(LIM, lower_white, upper_white, LIM); //범위에 따라 검/흰(0/255) 출력
132    cv::erode(LIM, LIM, cv::Mat(), cv::Point(-1, -1), 3);
133
134    QImage QImage(LIM.data, LIM.cols, LIM.rows, LIM.step, QImage::Format_Indexed8);
135    ui.Cam_6->setPixmap(QPixmap::fromImage(QImage));
136
137    cv::Mat label, stats, centroids;
138
139    int num = cv::connectedComponentsWithStats(LIM, LIM, stats, centroids, 8, CV_32S);
140
141    for (int i = 1; i < num; i++) {
142        Lx = static_cast<int>(stats.at<int>(i, cv::CC_STAT_LEFT));
143        Ly = static_cast<int>(stats.at<int>(i, cv::CC_STAT_TOP));
144        int width = static_cast<int>(stats.at<int>(i, cv::CC_STAT_WIDTH));
145        int height = static_cast<int>(stats.at<int>(i, cv::CC_STAT_HEIGHT));
146
147        cv::rectangle(IMG2, cv::Rect(Lx, Ly, width, height), cv::Scalar(161, 223, 110), 4);
148    }
149  }
```

```cpp
void MainWindow::judge()
{
  ui.lcdNumber->display(Oy);
  ui.lcdNumber_2->display(Ly);
  if(Oy >= 83){
    ui.lineEdit->setText("ORANGE UP");
  }
  else if(Oy <= 52){
    ui.lineEdit->setText("ORANGE DOWN");
  }
  else{
    ui.lineEdit->setText("ORANGE MIDDLE");
  }
  if(Ly >= 106){
    ui.lineEdit_2->setText("LIME UP");
  }
  else if(Ly <= 79){
    ui.lineEdit_2->setText("LIME DOWN");
  }
  else{
    ui.lineEdit_2->setText("LIME MIDDLE");
  }
}
```

마지막으로 콘의 위치를 판별하는 judge()함수이다. 이는 알고리즘적으로 매우 간단하다. 위에서 얻은 콘의 x, y좌표값을 이용해 중앙선 위에 있었을 때를 if에, 선 아래 있을 때를 else if에, 그 외는 중앙선에 걸친다고 출력하도록 했다.

```cpp
MainWindow::MainWindow(int argc, char** argv, QWidget* parent) : QMainWindow(parent), qnode(argc,
{
  ui.setupUi(this);  // Calling this incidentally connects all ui's triggers to on_...() callbac

  my_IP = QHostAddress("192.168.0.132");
  receivesocket = new QUdpSocket;
  PORT = 8888;

  setWindowIcon(QIcon(":/images/icon.png"));

  qnode.init();

  if(receivesocket->bind(my_IP, PORT)){
    connect(receivesocket, SIGNAL(readyRead()), this, SLOT(ImgRec()));
  }

  // ui.label = new QLabel(this);
  // ui.label->setGeometry(10, 10, 1080, 720);  // 적절한 크기와 위치 설정
  // ui.label->setScaledContents(true);

  QObject::connect(ui.Slider_1, SIGNAL(valueChanged(int)), this, SLOT(SliderValueChanged(int)));
  QObject::connect(ui.Slider_2, SIGNAL(valueChanged(int)), this, SLOT(SliderValueChanged(int)));
  QObject::connect(ui.Slider_3, SIGNAL(valueChanged(int)), this, SLOT(SliderValueChanged(int)));
  QObject::connect(ui.Slider_4, SIGNAL(valueChanged(int)), this, SLOT(SliderValueChanged(int)));
  QObject::connect(ui.Slider_5, SIGNAL(valueChanged(int)), this, SLOT(SliderValueChanged(int)));
  QObject::connect(ui.Slider_6, SIGNAL(valueChanged(int)), this, SLOT(SliderValueChanged(int)));
  QObject::connect(&qnode, SIGNAL(rosShutdown()), this, SLOT(close()));
}
```

추가적으로 MainWindow 즉 생성자에서는 udp를 위한 설정을 해주었고 udp를 통한 이미지 데이터가 들어왔을 때 ImgRec()함수를 실행하는 SIGNAL, SLOT을 설정해주었다. 또한 하한값, 상한값의 HSV를 슬라이더를 통해 조절 가능하도록 UI와의 연결도 해주었다.

결과는 아래 사진과 같이 나온다.