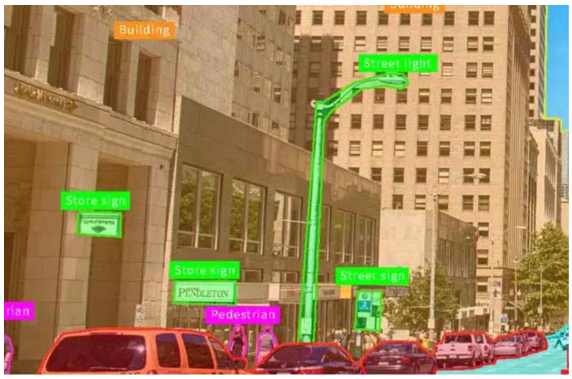
로빛18기 인턴 소재웅_ 비전 보고서

vision 이란?

비전이란, 컴퓨터가 디지털 이미지나, 비디오에서 정보를 추출하고 이를 해석하여 특정 작업을 수행하는 컴퓨터 과학 분야이다. 쉽게 말해, 기계에 '눈'을 달아주는 것으로, 컴퓨터가 이미지와 비디오 데이터를 분석하고 해석하는 기술



[비전기술을 이용해 사물을 감지함]

vision이 이용되는 분야들

1. Filtering

흐릿한 사진을 밝게 해주거나, 강조하고 싶은 곳에 포커스를 주는 필터링 기술

2. HDR

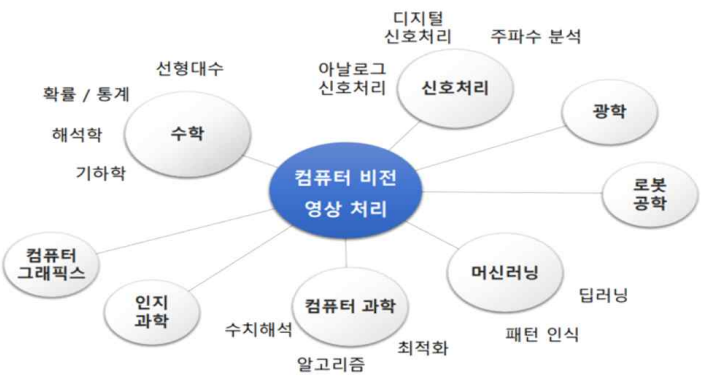
가장 밝은 곳부터 가장 어두운곳 까지 마치 사람이 눈으로 보는 것과 최대한 가깝게 밝기의 범위를 확장하는 기술

3. Image Noise Reduction

이미지의 노이즈를 최소화 하여 사진을 더욱 깨끗하고 선명하게 만들어주는 기술

4. Super Resolution

기존의 사진을 확대하였을 때 좀 더 선명하게 확대를 시켜줄 수 있는 기술



비전의 작동원리

1. 이미지 전처리

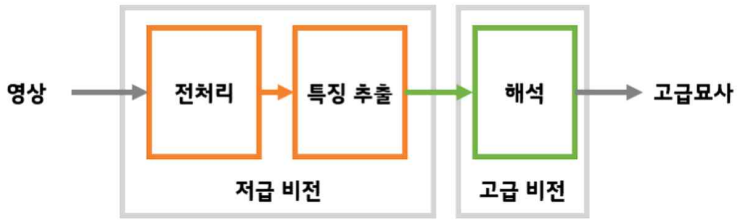
비전의 첫 단계는 원본 이미지나 영상의 전처리이다. 디 단계에서는 이미지 노이즈 제거, 명압 및 색 보정, 크기 및 해상도 조절 등을 수행하며, 처리를 위한 최적의 이미지 형태 생성 이렇게 전처리 된 이미지를 통해 비전 알고리즘이 보다 정확하게 작동할 수 있다.

2. 특징 추출

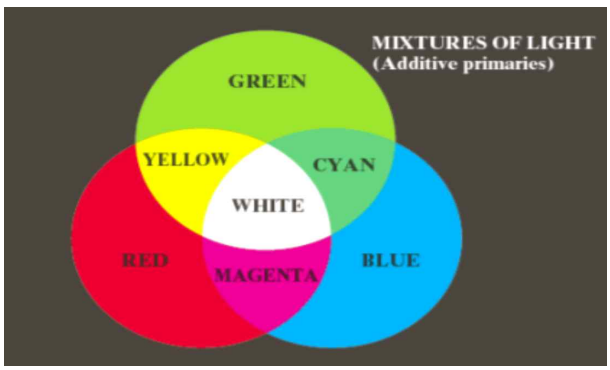
전처리 된 이미지에서 컴퓨터 비전 알고리즘은 다양한 특징 추출
이러한 특징에는 모서리, 코너, 텍스쳐, 색상, 분포 등이 포함되며, 이들 특징은 이미지 내 부분적인 패턴을 나타낸다. 특징 추출은 인식, 분류, 탐지 등의 분야애서 사용되는 핵심 구성 요소 이다.

3. 패턴 분석 및 분류

추출된 특징을 바탕으로 비전 알고리즘은 패턴을 분석하고 객체를 분류한다. 여기서 머신 러닝, 딥러닝 등 다양한 방법이 사용 된다.



빛의 삼원색 - BLUE, GREEN, RED
색의 삼원색 - Magenta, Yellow, Cyan
무채색 - 색상 정보가 존재하지 않는, 각각의 color component의 비율이 같은 경우.(rgb의 비율이 같다.) <-> 유채색

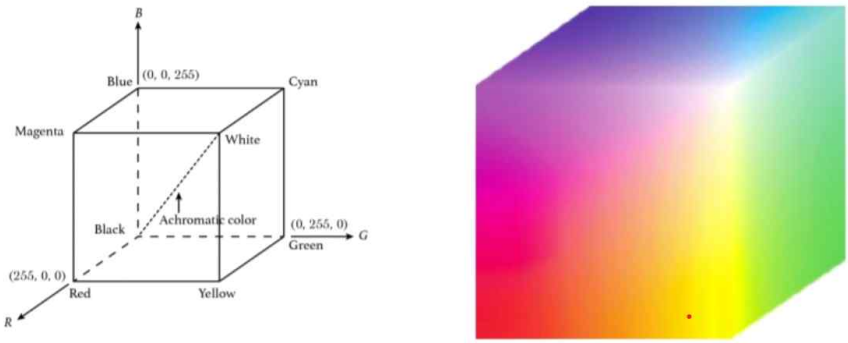


[빛의 혼합물]

RGB란?

비전, 즉영상처리 분야에서는 색상을 표현할 때, 빛의 3원색(RGB)를 이용한다.
R-channel, G-channel, B-channel로 이루어져있다.
intensity level은 일반적으로 256이므로 [0,255]로 표현 가능하다.

Red = (255,0,0)
White = (255,255,255) (무채색이기 때문에 비율이 같다.)
Black = (0,0,0)
(이때, open CV에서는 bgr순서대로 값을 입력한다.)
-> 이로 인해 Open cv에서는 이미지에 .rgbswap을 해주어야 한다.



RGB 모델. 무채색은 왼쪽 점선에 위치해있음.

RGB의 단점
--> 어두운색에서는 값의 변화가 잘 나타나지 않는다. 그래서 사용되는 것이 바로 HSV이다.

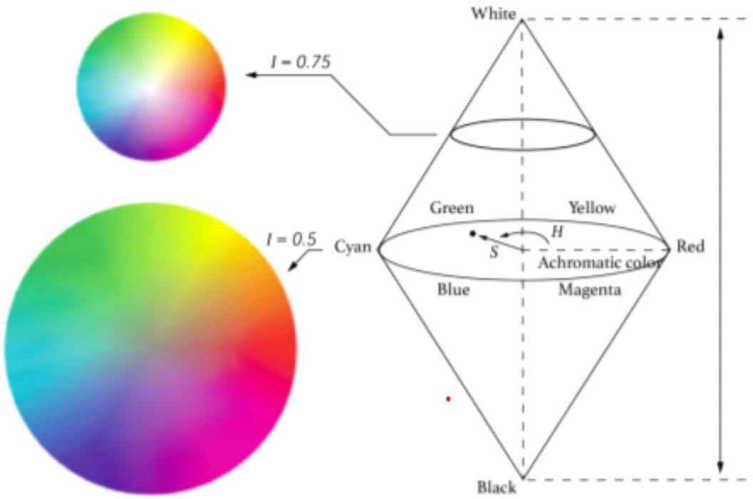
Grayscale image란?

Hue나 saturation 픽셀의 값이 모두 0인 것. 즉, 각각의 픽셀이 밝기만 표현 가능한 것으로 흑백 영상을 의미한다. lightness(or brightness)의 픽셀값이 0에 가까울수록 검정, 255에 가까울수록 흰색이다.

HSI(HSV)란?

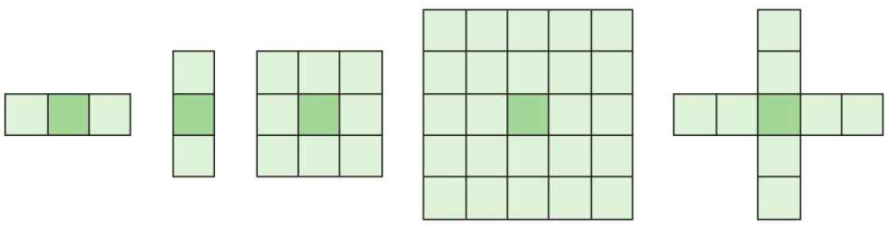
Hue-channel, Saturation-channel, Intensity-channel 값으로 구성되어 있다.
Hue : 색조. 특정한 색상을 결정짓는 우세한 파장.(색도)
Saturation : 상대적인 (색상의) 순도. 백색이 하나도 섞여있지 않은, 특정한 색상이 순수하게 특정한 하나의 파장으로 구성되어있으면 saturation이 높다고 할 수 있다. (명도)
Intensity(Brightness) : 어떤 색상의 밝기 값. 색상정보값은 포함되어있지 않고, 단지 세기를 표현 (채도)

HSI는 HSV라고도 불리는데 이때 V는 value의 약자이다.
S가 크면 선명한 색상, I가 크면 밝기가 밝은 색상이다



필터

필터링(Filtering) : 영상에서 원하는 정보만 통과 시키고 원치 않는 정보는 걸러 내는 작업
영상의 필터링은 보통 마스크(mask)라고 부르는 작은 크기의 행렬을 이용한다.
마스크는 다양한 크기와 모양으로 정의 할 수 있으며, 마스크 행렬의 원소는 보통 실수로 구성



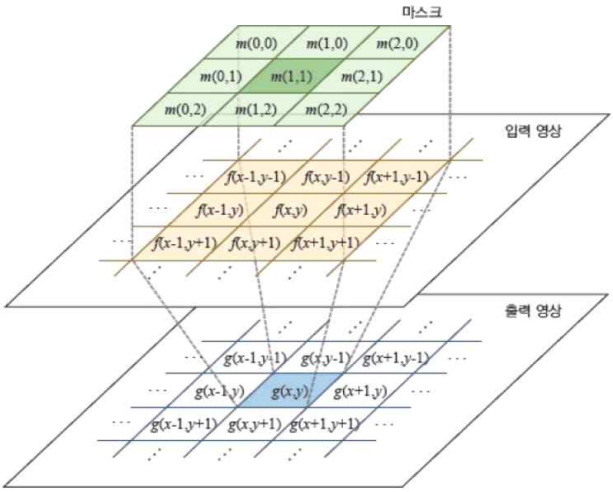
필터링 연산 방법

필터링 연산의 결과는 마스크 행렬의 모양과 원소 값에 의해 결정됨

마스크를 이용한 필터링은 입력 영상 픽셀 위로 마스크 행렬을 이동시키면서 마스크 연산을 수행하는 방식으로 작동

마스크 연산의 결과를 출력영상에서 고정점 위치에 대응되는 픽셀 값으로 설정한다.

$$
\begin{aligned}
g(x,y) = {} & m(0,0)f(x-1,y-1) + m(1,0)f(x,y-1) + m(2,0)f(x+1,y-1) \\
+ {} & m(0,1)f(x-1,y) + m(1,1)f(x,y) + m(2,1)f(x+1,y) \\
+ {} & m(0,2)f(x-1,y+1) + m(1,2)f(x,y+1) + m(2,2)f(x+1,y+1)
\end{aligned}
$$

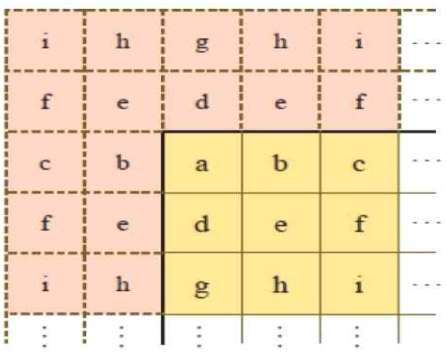


필터링 외각 처리

나중에 CNN코드에서 padding 부분과 연관이 있는 부분.

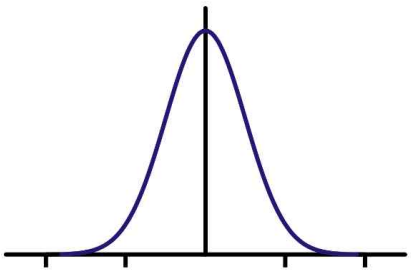
영상의 필터링 수행할 때, 영상의 가장자리 픽셀을 확장하여 영상 바깥쪽에 가상의 픽셀을 만든다.

영상의 바깥쪽 가상의 픽셀 값을 어떻게 설정하는 가에 따라 필터링 연산 결과가 달라진다.



필터 예시 - Gaussian filter

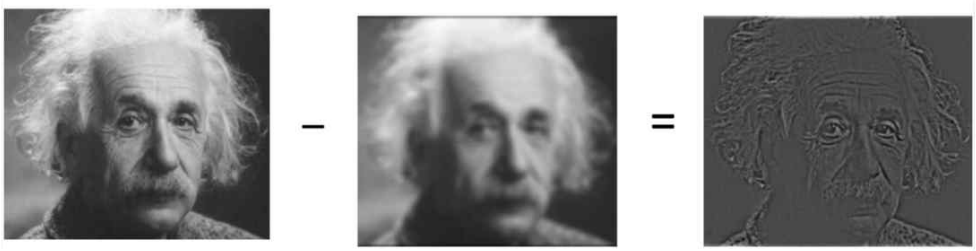
gaussian은 distribution을 확인했을 때 중심을 기준으로 대칭을 이룬다.
따라서 gaussian filter의 kernel 예시를 확인하면 중심을 기준으로 더 큰 weight를 갖고 있으며, 중심에서 멀어질수록 weight가 감소하는 것을 볼 수 있다.



$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Low-pass Filter, High-pass Filter

가우시안 필터는 Low-pass Filter이다.
Low-pass filter는 이미지로부터 "high-frequency"를 제거하는 필터이다.
이미지에 가우시안 필터를 convolve하면 블러링된 이미지를 얻을 수 있다.
이때, 블러링된 이미지를 원래 이미지에서 빼면 이미지의 디테일한 부분을 얻을 수 있다.



이렇게 이미지의 디테일한 부분을 추출하는 연산을 High-pass Filter라고 합니다.

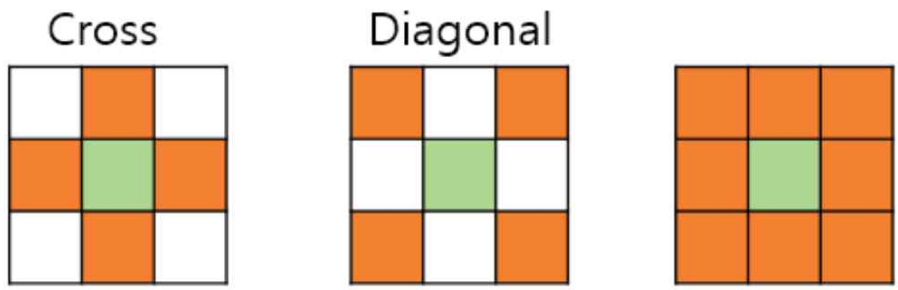
--> 라벨링을 시키기 위해서는 꼭 써주어야한다.

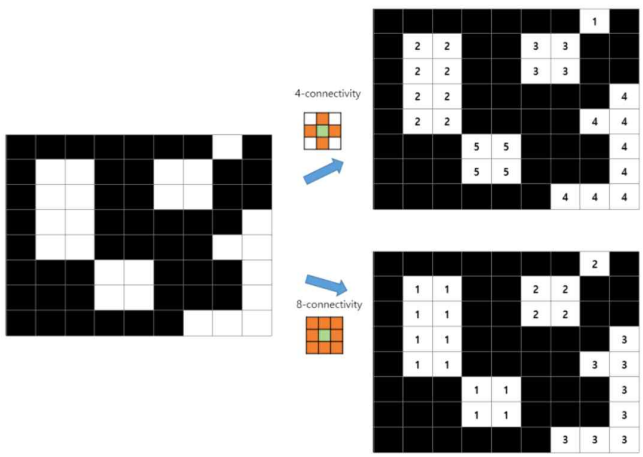
영상 라벨링(Image Labeling) 이란?

영상 라벨링은 영상 내에서 주위 같은 밝기의 픽셀값을 가지는 픽셀들을 그룹화하여 그룹별로 번호를 매기는 방법을 말한다. object detection, segmentation 등에 많이 사용되는 기법이다. 전경을 객체(Object)라고 표현할 수도 있다. 하나의 객체들은 덩어리로 있기 때문에 몇개의 객체가 있는지를 확인하기 위해서는 연결된 덩어리를 구분할 필요가 있다. 이렇게 연결된 구성요소(connected component) 레이블링을 하는 것을 CCL(Connected Components Labeling)이라고 부릅니다.

라벨링 방법은?

픽셀의 연결관계를 정의하는 방법은 크게 3가지가 있다. 4방향 연결과 8방향 연결이 있고, 4방향 연결 방법 중에는 십자가 모양과 대각선 모양이 있다. 하지만 대각선은 많이 사용되지 않아 OpenCV에서는 지원하고 있지 않다.





Open_cv

OpenCV는 Open Source Computer Vision의 약자로 영상 처리에 사용할 수 있는 오픈 소스 라이브러리 입니다. 컴퓨터가 사람의 눈처럼 인식할 수 있게 처리해주는 역할을 하기도 하며, 우리가 많이 사용하는 카메라 어플에서도 OpenCV가 사용하기도 한다. (추가로 자율주행자동차에서 자동차의 눈이 되주는 것이 카메라와 OpenCV가 합작해서 해낸 일이다.) 추가로 사용되는 예로는 공장에서 제품 검사할 때 의료 영상 처리 및 보정 그리고 판단
CCTV영상, 로보틱스 등 다양한 범위에서 사용되고 있다. 카메라로 찍어서 할 수 있는 모든 일은 OpenCV로 처리할 수 있다. 여기에 머신 러닝과 A.I를 활용해서 그 활용도를 더욱 넓혀가고 있는 중이다.

OpenCV 로고

코드 내에서 이용한 Open_cv 함수

--> cv::라는 namespace를 싸주어야한다.

Ros에서 Open_cv를 이용하기 위해서는 Cmake 파일을 수정해야 한다.

```
##############################################################################
#########
# Catkin
##############################################################################
#########

# qt_build provides the qt cmake glue, roscpp the comms for a default talker
find_package(catkin REQUIRED COMPONENTS
  roscpp
  std_msgs geometry_msgs
  cv_bridge
  )
```

---> find_pakage 내에 cv_bridge를 추가해 준다.

```
#############################################################################
#########
# Qt Environment
#############################################################################
#########
# included via the dependency call in package.xml
find_package(Qt5 REQUIRED COMPONENTS Widgets Network Gui Core)
find_package(OpenCV 4.5.2 REQUIRED)
include_directories(${Qt5Widgets_INCLUDE_DIRS}
${Qt5Core_INCLUDE_DIRS}${Qt5Network_INCLUDE_DIRS}${OpenCV_INCLUDE_DIRES})
```

--> Qt Environment에서 find_package(OpenCV 4.5.2 REQUIRED)를 추가해준다.
그리고 밑에 ${OpenCV_INCLUDE_DIRES}도 추가해준다.

코드 내에서 이용된 Open_cv함수 정리

1. **cv::Mat** – 이미지의 class로 이미지를 저장할 수 있다.
ex) cv:: Mat img_HSV;

2. **cv::resize** – 받아온 이미지의 크기를 수정해 줄 수 있다.
resize(크기를 조정할 이미지, 크기가 조정된 이미지, cv::Size(원하는 가로,원하는세로));
ex) cv::resize(clone_mat2, clone_mat2, cv::Size(320, 240));

3. **cv::GaussianBlur** – 가우시안 필터를 입힐 수 있다.
cv::GaussianBlur(필터를 씌우려는 이미지, 필터를 씌우고 난 이미지,cv::Size(필터의 크기),값);
ex) cv::GaussianBlur(clone_mat2,clone_mat2,cv::Size(),1);

4. **cv::cvtColor** – 컴퓨터에서 이미지를 불러오게 되면 BGR 이미지로 저장이 되기 때문에 HSV를 사용하기 위해서는 BGR에서 HSV로 전환을 시켜주어야한다. 그러기위해 이용되는 함수이다.
cv::cvtColor(변환하고자하는 이미지,변환한 이미지,cv:원래의 상태 -> 변환할 상태 (ex)BGR->HSV));
ex) cv::cvtColor(clone_mat2,img_HSV,cv::COLOR_BGR2HSV);

5. **cv::inRange –** 이미지를 이진화 시켜주는 함수이다. (검은색과 흰색만 나타나게 됨.)
cv::inRange(변환할 이미지,cv::Scalar(MIN – HSV의 값), cv::Scalar(MAX – HSV의 값), 변환할 이미지);
cv::inRange(clone_mat2,cv::Scalar(202,200,192),cv::Scalar(243,233,242),img_HSV2);

6. **cv::erode(침식), cv::dilate(팽창)** - HSV값을 조절하다보면 엄청작은 픽셀들이 사라지지 않는 경우가 있다. 이때 erode를 사용하면 그 작은 픽셀들이 사라지게 된다. 그리고 반대로 이제 원하는 HSV값을 조절을 했을 때 그 값이 너무 작게 나올 때 dilate를 이용해서 팽창시켜 그 부분을 크게 만들어 줄 수 있다.
cv::erode(원하는 이미지,erode된 이미지,mask,cv::Point(erode할 좌표),적용할 값);
cv::dilate(i원하는 이미지,dilate된 이미지 ,mask,cv::Point(dilate할 좌표),적용할 값);
이때 mask는 설정을 해주어야하는데
cv::Mat mask = cv:: getStructuringElement(cv::MORPH_RECT,cv::Size(3,3),cv::Point(1,1));
이런식으로 사용자 정의 kernel을 만들어주어야한다. 만들어 주기 위해서는 getStructuringElement() 함수를 사용해야한다. getStructuringElement() 괄호 안에는 int(shape), Size(ksize), Point(anchor)가 들어간다.

ex)cv::Mat mask = cv:: getStructuringElement(cv::MORPH_RECT,cv::Size(3,3),cv::Point(1,1));
ex) cv::erode(img_HSV3,img_HSV3,mask,cv::Point(-1,-1),4);
ex) cv::dilate(img_HSV3,img_HSV3,mask,cv::Point(-1,-1),4);

---> 적당하게 쓸 부분을 지정해주는 것이 좋다. erode와 dilate의 순서도 중요하다.

7. cv::Point – x,y좌표를 동시에 담고 있다.
cv::Point(x좌표, y좌표);
ex) cv::Point(-1,-1)

8. cv::HoughLinesP - 서로 직선 관계를 갖는 픽셀들만 골라내는 것이 허프 선 변환의 핵심입니다. OpenCV에서는 허프 변환을 위해 아래와 같은 함수를 제공합니다.
cv::HoughLinesP(하프라인 시킬 이미지, 직선 속성변수(vector<cv::VEc4i>타입의 어레이 변수), r 방향변위값(경계값), 회전방향각도(경계값),최소픽셀수(경계값), 직선최소길이, 픽섹간 허용최대갭(동일직선상))
ex) cv::HoughLinesP(line_img,white_line,1,CV_PI/180,50,30,60);

---> std::vector<cv::Vec4i> 각 선은 시작점과 끝점을 나타내는 4개의 정수로 표현되는 cv::Vec4i형식으로 저장이된다. 이러한정보를 이용해 line함숭로 선을 그릴 수 있다.

9. cv::line – 원하는 좌표에 선을 그려주는 함수이다.
cv::line(원하는 이미지, cv::Point(x,y좌표), cv::Point(x,y좌표),cv::Scalar(원하는 색깔),선의 두께);
ex) cv::line(clone_mat3,cv::Point(L[0],L[1]),cv::Point(L[2],L[3]), cv::Scalar(255, 0, 0),3,8);

10. cv::connectedComponentsWithStats – 라벨링을 해주는 함수이다.
cv::connectedComponentsWithStats(입력영상, 라벨링 되어서 나온 영상, (4빙향 연결성, 8방향 연결성, 기본값 8), (출력 영상type,CV_16S또는 CV_32S,기본값 CV_32S)) = 반환값(레이블 개수)
ex) **int num_red = cv::connectedComponentsWithStats(img_HSV3, img_HSV3, stats, centroids, 8, CV_32S);**

```
for (int i = 1; i < num_red; ++i) {
        x = static_cast<int>(stats.at<int>(i, cv::CC_STAT_LEFT));
        y = static_cast<int>(stats.at<int>(i, cv::CC_STAT_TOP));
        width = static_cast<int>(stats.at<int>(i, cv::CC_STAT_WIDTH));
        height = static_cast<int>(stats.at<int>(i, cv::CC_STAT_HEIGHT));
  cv::rectangle(clone_mat3, cv::Rect(x, y, width, height), cv::Scalar(0, 0, 255),4);
```

---> 이런 식으로 함수를 이용하여 라벨링하고 그 라벨링된 좌표를 따서 그에 맞게 직사각형을 그려주면 라벨링된 사물에 대하여 사각형을 자동적으로 그릴 수 있게된다.

11. cv::Mat::zeros - 주어진 크기와 타입을 갖는 모든 원소가 0으로 초기화된 행렬을 생성합니다.
cv::Mat cv::Mat::zeros(행, 열, 행렬데이터의 타입을 나타내는 매개변수);
ex) img_mask = cv::Mat::zeros(img_edges.rows,img_edges.cols,CV_8UC1);

12. cv::fillPoly – 다가형을 채우는데 사용되는 함수 행렬사에 해당 다각형을 채우는데 사용된다.
cv::fillPoly(넣을 이미지또는 행렬, 다각형의 꼭짓점 좌표를 담은 배열, 다각형이 닫혀있는지 요부를 나타내는 플래그(1), 채울 색깔, 다각형의 선의 유형을 나타내는 변수)
ex) cv::fillPoly(img_mask,ppt,npt,1,cv::Scalar(255,255,255),cv::LINE_8);

13. cv::bitwise_and - 두 개의 이미지나 행렬 간에 각 픽셀당 AND 비트 연산을 수행하는 함수이다.
cv::bitwise_and(첫번째 이미지, 두 번째 이미지 , 결과 이미지);
ex) cv::bitwise_and(img_edges,img_mask,img_masked);

코드 분석

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

    setWindowIcon(QIcon(":/images/icon.png"));
    qnode.init();
    MY_IP = QHostAddress("192.168.0.79");
    NOOK_IP = QHostAddress("192.168.0.51");
    PORT=8888;
img_socket = new QUdpSocket(this);
     if(img_socket->bind(MY_IP,PORT))
    { std::cout<<"3"<<std::endl;
        QObject::connect(img_socket, SIGNAL(readyRead()),this,SLOT(Load()));
    }
    QObject::connect(&qnode, SIGNAL(rosShutdown()), this, SLOT(close()));

  }
clone_mat = udp_.MatImgRcv(img, PORT, NOOK_IP, *img_socket);
```

udp를 이용해서 사진을 전송받아야 했기 때문에 udp의 읽는 부분을 만들어 주었다. 만약에 나의 소켓에서 읽혔다면 내가 지정된함수를 실행시키도록하여서 Load라는 함수를 실행시키도록 하였다. 그리고 udp 헤더파일을 이용해서 이미지를 전달 받도록 하였다.

---> udp를 이용하기 위해서는 나 자신의 소켓, port, 나의 IP를 선언해주고 그 IF문 부분에는 나의 IP를 써주고 이제 udp에 있는 받는 함수에는 udp로 이미지를 보내고 있는 사람의 IP를 써주어 통신을 한다.

```
void MainWindow::Load(){
cv::Mat mask = cv::    getStructuringElement(cv::MORPH_RECT,cv::Size(3,3),cv::Point(1,1));
    clone_mat = udp_.MatImgRcv(img, PORT, NOOK_IP, *img_socket);

        cv::Mat clone_mat2 = clone_mat;
         cv::Mat clone_mat3 = clone_mat;
         cv::resize(clone_mat3, clone_mat3, cv::Size(320, 240));
      cv::resize(clone_mat2, clone_mat2, cv::Size(320, 240));

      cv::GaussianBlur(clone_mat2,clone_mat2,cv::Size(),1);
      cv::cvtColor(clone_mat2,img_HSV,cv::COLOR_BGR2HSV);
```

처음에 Load 함수에서는 udp로 받은 이미지를 Mat clone_mat2의 이미지로 저장을 하고 원본의 이미지를 건들지 않도록 하였다. 원본사진을 띄우기 위해서 clone_mat3에도 담아주었다. 여기서 clone_mat2이미지는 이진화하여 사용하는 이미지 이기 때문에 맨처음 이미지로 사용할 수 없었다. 그리고 그후 resize 함수를 이용해 적당한 크기를 조절해 주었고 가우시안필터를 씌워주고 BGR을 HSV로 전환해주었다.

```
cv:: Mat line_img;
cv::inRange(clone_mat2,cv::Scalar(ui.horizontalSlider->value(),ui.horizontalSlider_2->value(),ui.
horizontalSlider_3->value()),cv::Scalar(ui.horizontalSlider_4->value(),ui.horizontalSlider_5->valu
e(),ui.horizontalSlider_6->value()),img_HSV);
cv::inRange(clone_mat2,cv::Scalar(202,200,192),cv::Scalar(243,233,242),img_HSV2);
cv::inRange(clone_mat2,cv::Scalar(0,100,153),cv::Scalar(111,184,255),img_HSV3);
cv::inRange(clone_mat2,cv::Scalar(57,137,92),cv::Scalar(139,225,191),img_HSV4);
cv::erode(img_HSV,img_HSV,mask,cv::Point(-1,-1),1);
cv::dilate(img_HSV,img_HSV,mask,cv::Point(-1,-1),4);
cv::dilate(img_HSV2,img_HSV2,mask,cv::Point(-1,-1),5);
cv::erode(img_HSV2,img_HSV2,mask,cv::Point(-1,-1),1);
cv::erode(img_HSV3,img_HSV3,mask,cv::Point(-1,-1),4);
cv::dilate(img_HSV3,img_HSV3,mask,cv::Point(-1,-1),4);
cv::erode(img_HSV4,img_HSV4,mask,cv::Point(-1,-1),4);
```

HSV값으로 전환을 해주었다면 이제는 이진화를 시켜주어야한다. 이진화를 INRANGE함수를 이용하여 시주는데 여기서 img_HSV는 툴바를 이용해서 원하는 사물의 HSV값을 얻어내기 위해서 만든 이미지이고 img_HSV_2, img_HSV_3, img_HSV_4는 주황색 고깔, 네온색 고깔, 선색깔에 맞는 HSV값들을 입력한 값들이다. 그리고 HSV값을 조절하는데 완전한 이진화가 되지않아 erode하고 dilate로 픽셀을 수축 팽창 시켜 주었다.

```
cv::Canny(img_HSV2,line_img,10,300,5);
cv::Point points[4];
points[0] = cv::Point(0,100);
points[1] = cv::Point(0,150);
points[2] = cv::Point(270,150);
points[3] = cv::Point(270,125);
line_img = region_of_interest(line_img,points);
std::vector<cv::Vec4i> white_line;
cv::HoughLinesP(line_img,white_line,1,CV_PI/180,50,30,60);
for(size_t i = 0; i<white_line.size();i++){
cv::Vec4i L = white_line[i];
cv::line(clone_mat3,cv::Point(L[0],L[1]),cv::Point(L[2],L[3]), cv::Scalar(255, 0, 0),3,8);
    }
```

```
cv::Mat MainWindow::region_of_interest(cv::Mat img_edges,cv::Point *points){
img_mask = cv::Mat::zeros(img_edges.rows,img_edges.cols,CV_8UC1);
const cv::Point*ppt[1] = {points};
int npt[]={4};
cv::fillPoly(img_mask,ppt,npt,1,cv::Scalar(255,255,255),cv::LINE_8);
cv::bitwise_and(img_edges,img_mask,img_masked);
return img_masked;
```

이제 관심영역을 지정해서 HSV로 먼저 선을 딴 img를 Canny로 선을따는 과정이다. 일단 Point를 이용해서 관심영역들을 지정해준다. 그후 region of interest에 이미지를 넣어서 그 부분만을 cv::HoughLinesP(line_img,white_line,1,CV_PI/180,50,30,60); 함수를 이용해 선을 그리도록 하였다.

```
QImage
img3((uchar*)img_HSV3.data,img_HSV3.cols,img_HSV3.rows,QImage::Format_Indexed8);
QPixmap pixmap3 = QPixmap::fromImage(img3);
ui.label_5->setPixmap(pixmap3);
QImage
mg4((uchar*)img_HSV4.data,img_HSV4.cols,img_HSV4.rows,QImage::Format_Indexed8);
QPixmap pixmap4 = QPixmap::fromImage(img4);
ui.label_6->setPixmap(pixmap4);
QImage
img5((uchar*)img_mask.data,img_mask.cols,img_mask.rows,QImage::Format_Indexed8);
QPixmap pixmap5 = QPixmap::fromImage(img5);
ui.label_3->setPixmap(pixmap5);
```

6칸의 label칸중 3개의 라벨창에다가 HSV값을 조절해준 이미지를 넣어주었다. img5는 관심영역을 label에 나오도록 한 것이다.

```
cv:: Mat stats;
cv:: Mat centroids;
cv:: Mat stats2;
cv:: Mat centroids2;
int num_red = cv::connectedComponentsWithStats(img_HSV3, img_HSV3, stats, centroids,
8, CV_32S);
 for (int i = 1; i < num_red; ++i) {
          x = static_cast<int>(stats.at<int>(i, cv::CC_STAT_LEFT));
          y = static_cast<int>(stats.at<int>(i, cv::CC_STAT_TOP));
      width = static_cast<int>(stats.at<int>(i, cv::CC_STAT_WIDTH));
          height = static_cast<int>(stats.at<int>(i, cv::CC_STAT_HEIGHT));
   cv::rectangle(clone_mat3, cv::Rect(x, y, width, height), cv::Scalar(0, 0, 255),4);
```

```
int num_green = cv::connectedComponentsWithStats(img_HSV4, img_HSV4, stats2,
centroids2, 8, CV_32S);

 for (int i = 1; i < num_green; ++i) {
            x2 = static_cast<int>(stats2.at<int>(i, cv::CC_STAT_LEFT));
            y2 = static_cast<int>(stats2.at<int>(i, cv::CC_STAT_TOP));
            width2 = static_cast<int>(stats2.at<int>(i, cv::CC_STAT_WIDTH));
            height2 = static_cast<int>(stats2.at<int>(i, cv::CC_STAT_HEIGHT));
   cv::rectangle(clone_mat3, cv::Rect(x2, y2, width2, height2), cv::Scalar(0, 255, 0),4);
```

앞에서 설명한 connectedComponentsWithStats 함수를 이용해서 HSV화 된 이미지를 라벨링 시켜준다. 그리고 그 라벨링된 왼쪽끝의 X좌표와 맨 위의 Y좌표값, 가로, 세로의 값을 찾아서 그 좌표에 사각형을 그리도록 하였다. (green,red동일하게 한다.)

```
if(y<60){
   std::cout<<"1"<<std::endl;
      ui.textEdit_2->setText("Top");
}
else if(y>85){
     std::cout<<"2"<<std::endl;
        ui.textEdit_2->setText("down");
}
else{
     std::cout<<"3"<<std::endl;
        ui.textEdit_2->setText("line");
}
if(y2<90){
   std::cout<<"1"<<std::endl;
   ui.textEdit->setText("Top");
}
else if(y2>113){
     std::cout<<"2"<<std::endl;
       ui.textEdit->setText("down");
}
else{
     std::cout<<"3"<<std::endl;
        ui.textEdit->setText("line");
}
```

그리고 이제 콘을 가져다 놓으면서 선을 기준으로 TOP값의 최댓값 bottom값의 최솟값을 구해 if문을 통해서 ui내에서 TOP BOTTOM이 뜨게하도록 하였다.

결과는 영상첨부