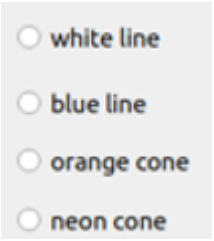
OPENCV DAY2
과제 1번 보고서

2023741024
로봇학부 박건후

목차

(1) ui 설명

프로그램을 실행하면 생성되는 윈도우창입니다. 6개의 영상이 나올 창이 있고



왼쪽의 부분을 보면 white_line, blue_line, orange_cone, neon_cone 이라는 라디오 박스를 만들었습니다. 이 라디오 박스를 선택하면 현재 상태의 hsv각각의 high, low의 값을 볼 수 있고 슬라이더를 통해 그 값을 제어할 수 있도록 하였습니다.

우하단에는 왼쪽 사진처럼 neon 꼬깔이 어디에 있는지, orange꼬깔이 어디에 있는지 lineEdit을 통해 텍스트 출력을 하여 알려주게 됩니다.

(2) main_window.hpp파일 설명

```
struct slider{
    bool flag; int huehigh; int huelow;
    int sathigh; int satlow; int valhigh; int vallow;
};
```

slider라는 구조체를 선언한 부분입니다. 변수명을 보시면 예측하실 수 있듯이 hue에 대한 low, high범위, saturation에 대한 low, high범위, value에 대한 low, high 범위를 저장할

int 변수와 어떤 항목에 대한 슬라이드의 hue, saturation, value를 변환할 것인지에 대한 flag 변수를 선언한 것입니다. 이는 슬라이더를 white_line, blue_line, orange_cone, neon_cone에 대해 각각 처리하고 각각의 슬라이더 값을 저장하며 특정 라디오 박스를 눌렀을 경우 4개 중 어떤 항목에 대한 것을 바꿀지에 대한 flag변수까지 선언한 것입니다.

```
void startCamera();   // 카메라 오픈 및 초기화 메서드
void grabFrame(); // 한 프레임씩 읽어서 라벨에 사진 띄우기
void white_click();// white_line 라디어 박스 눌렀을 때
void blue_click();  //blue_line 라디오 박스 눌렀을 때
```

슬롯 메서드를 선언한 부분입니다. startCamera를 만들어에 대한 기본 세팅을 하고자 하였습니다. grabFrame에서는 카메라 영상의 사진을 가져와서 해당 사진에 대해 모든 처리를 하게 됩니다. 이 grabFrame을 타이머를 통해 짧은 시간마다 계속 호출되어 그 시간마다 나오는 사진을 통해 바이너리 이미지를 만들고 사각형을 그리며, 꼬깔의 위치를 계산하는 것을 모두 수행하는 메서드를 만드는 방식으로 설계한 것입니다.

white_click과 blue_click 말고도 neon_click, orange_click 메서드가 있습니다. 이 메서드들은 white_line, blue_line, orange_cone, neon_cone이라는 라디오 박스를 눌렀을 때 각기 호출되는 슬롯 메서드들입니다. 특정 라디오 박스를 눌렀다면 그 라디오 박스에 해당되는 슬롯 메서드를 호출되고 각 메서드에서는 slider 객체에 저장되어 있는 멤버들로 슬라이더 위치를 최신화하게 됩니다. 이전에 설정했던 슬라이더 값을 저장했다가 다시 그 라디오 박스를 누르게 되었을 때 그 상태로 복원해야 하기 때문입니다.

```
void any_slider_changed(int val); //슬라이더 바뀌었을 때,
```

그리고 사용자는 그렇게 슬라이더가 최신화되었을 때 슬라이더를 직접 조작하여 움직이게 할 것입니다. 이렇게 슬라이더를 직접 움직이게 할 때에 대한 슬롯 메서드인 any_slider_changer 메서드를 만들었습니다. 슬라이더에는 valueChanged라는 시그널이 있지만 slideMoved라는 시그널 또한 존재합니다. 이 슬롯 메서드는 slideMoved라는 시그널에 대해 호출되도록 하였고, 각각의 슬라이더가 바뀔 때 각각의 슬롯 메서드를 만들면 메서드가 매우 많아지고 복잡해지기에 이 하나의 슬롯 메서드로 처리되도록 설계했습니다.

```
QTimer *camTimer = nullptr; //한 프레임 읽을 때의 타이머
VideoCapture cap; //카메라 영상 정보 가져오기 위해
```

private에 있는 멤버들을 말씀드리겠습니다. getFrame 메서드를 일정 시간 간격으로 호출하게 할 것이므로 QTimer에 대한 포인터를 선언하였고, 카메라 영상의 정보를 얻어와야 하므로 VideoCapture 클래스의 객체를 선언하였습니다.

```
Mat neon_cone(Mat neon);
Mat orange_cone(Mat orange);
```

그리고 멤버함수로 바이너리 이미지를 구해올 메서드를 각기 선언하였습니다. 이 바이너리 이미지를 qt 화면에 각기 출력해야 되기에 neon_cone, orange_cone, blue_line, white_line이라는 멤버함수를 각기 만들어 각각의 범위에 맞게 바이너리 이미지를 생성되도록 만들기 위해 각각의 메서드를 만든 것입니다.

```
Mat draw_rectangle(Mat mask, Mat origin, Scalar color,std::string s);
void pos_cal();
```

사각형 그리는 알고리즘을 만들어야 했기에 새로운 메서드를 만들고 구현했습니다. draw_rectangle로 사각형을 만들었고, pos_cal에서는 이처럼 만들어진 사각형 정보를 가지고 선 위치에 대한 꼬깔의 위치가 up, down, right, left인지를 알아내어 lineEdit에 출력하게 했습니다.

```
slider white_slide;
slider blue_slide;
slider neon_slide;
slider orange_slide;
```

이 slider객체를 각 라디오 버튼에 대응되게 선언하여 특정 라디오 박스 상황에서의 슬라이더 값을 각각 대응되는 huehigh, huelow, sathigh, satlow, valhigh, vallow에 각각 저장되도록 한 것입니다.

(3) main_window.cpp 파일 설명

MainWindow의 생성자에서는 setupUi와 타이머 객체를 동적할당하고 기본적인 connect를 통해 시그널과 슬롯을 연결했습니다.

connect한 것에 대해 말로 설명드리자면, 라디오 버튼이 눌렸을 때는 각각의 메서드를 호출하여 그 메서드에 해당되는 slider 객체에 저장된 멤버들을 통해 슬라이더를 setText로 초기화하게 하였습니다. 전에 말씀드렷듯, 모든 슬라이더에 대해 슬라이더가 움직이면(slideMoved) any_slider_changed가 호출되도록 하였습니다. 그리고 QTimer가 타임 아웃될 때 grabFrame 메서드가 호출되도록 하였습니다.

```
ui->hue_high->setRange(0, 179);
ui->hue_low-> setRange(0,179);
```

그리고 슬라이더의 objectName이 각각의 슬라이더에 대응되는 이름인 hue_high, hue_low, saturation_high, saturation_low, value_high, value_low으로 만들었고 이 슬라이더의 setRange로 각각의 슬라이더에 대해 범위를 지정한 것입니다. 이렇게 한 이유는 hsv에 값의 범위가 존재하기 때문에 이 범위에 맞게 만들고자 했습니다. 괜히 코드로 알고리즘을 짜서 범위를 매핑하기 보다는 이 방법이 훨씬 쉽고 간단합니다. Hue를 설정할 때는 0~179, Saturation을 설정할 때는 0~255, Value를 설정할 때도 0~255로 설정하였습니다.

```
neon_slide.huelow = 20;
neon_slide.huehigh = 30;
```

이렇게 각각의 범위를 설정한 이후에는 각각의 slider 객체 멤버변수를 초기값으로 설정해주었습니다. 제일 인식 잘되는 범위의 값을 설정해준 것입니다. 이를 생성자에서 정의해주었기에 프로그램을 시작하여 라디오 박스를 누르면 바로 해당 값으로 슬라이더가 설정된 모습을 보실 수 있습니다. 그리고 생성자의 마지막 코드로 startCamera라는 메서드를 호출하여 카메라를 열고 기본적인 해상도 처리를 하고 타이머를 시작시킵니다.

```
Mat MainWindow:: neon_cone(Mat neon){
  Mat temp, result;
```

neon_cone이라는 메서드입니다. 이 메서드는 바이너리 이미지를 생성하는 메서드로 이전에 말씀드렸듯, 각각의 설정된 hsv값으로 각각 바이너리 이미지를 생성하도록 서로 다른 메서드로 각각 만들었습니다.

```
Scalar lower_yellow(neon_slide.huelow, neon_slide.satlow, neon_slide.vallow);
Scalar upper_yellow(neon_slide.huehigh, neon_slide.sathigh, neon_slide.valhigh);
cvtColor(neon, temp, COLOR_BGR2HSV);
inRange(temp, lower_yellow, upper_yellow, result);
```

그래서 neon_cone에 대응되는 slider 객체로 hsv 범위 값을 통해 Scalar를 만들어서 기존에 바이너리 이미지를 만드는 방식대로 HSV로 바꾸고, inRange를 하여 만들어진 바이너리 이미지를 리턴하는 형태입니다. orange_cone메서드, blue_line메서드, white_line메서드를 모두 이와 같은 원리, 방식으로 만들었습니다.

```
Mat MainWindow::draw_rectangle(Mat mask, Mat origin, Scalar color, std::string s)
```

draw_rectangle에서는 주황 콘, 네온 콘, 파란 줄, 흰 줄을 각각 명확히 얻어와야 하기에 각각에서 생성된 바이너리 이미지를 가져오는 것이 각각을 가져오는 것이 제일 명확하게 사각형을 그릴 수 있습니다. 그래서 mask라는 매개변수를 통해 바이너리 이미지를 가져오고, 그리고 원본 이미지에 이 사각형을 덧붙여야 하므로 origin 매개변수로 원본 이미지를 가져오도록 했습니다. 사각형의 색깔을 호출하는 곳에서 설정하기 위해 Scalar color를 받았습니다. 그리고 string을 설정하여 어떤 것에 대해 추출할 것인지를 호출하는 부분에서 작성해주도록 하여 만들어진 사각형을 그 스트링 값에 따라 전역으로 선언한 Rect 객체에 저장하도록 하기 위함입니다. 이 Rect 객체를 저장하는 이유는 pos_cal할 때 이 사각형 정보를 통해 꼬깔의 위치를 판단하기 때문입니다. 아래의 사진이 Rect 객체를 전역으로 선언한 부분입니다.

```
Rect neon_rect, orange_rect, blue_rect, white_rect;
```

```
std::vector<std::vector<Point>> contours; //바이너리 이미지에 있는 점들
선언
findContours(mask, contours, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);
```

사각형을 그려주는 메서드인 draw_rectangle메서드 내부입니다. 사각형을 그리기 위해서는 바이너리 이미지에 있던 모든 흰색 부분들을 가져와야 하고, 흰색으로 된 점들의 위치를 잡기 위해 contours 라는 이차원 벡터를 선언했습니다. opencv에 있는 Rect, Point들이 이차원에서 정의되는 거처럼 화면이 이차원 좌표로 구성되기에 이차원 배열처럼 할당한 것입니다. 이 drawRectangle 메서드를 만들 때 이 점들을 이차원 벡터에 저장하고 그 중 어느 정도 크기가 되는 것들을 사각형으로 그려주도록 만드는 방식을 생각했고 이를 라이브러리를 통해 구현하였습니다. findContours라이브러리로 외곽의 윤곽선만 보며, 이 좌표를 저장하도록 설정하고 contours 벡터에 잡힌 것들의 좌표를 저장하게 하였습니다.

```
Rect rect;
for (size_t i = 0; i < contours.size(); ++i) {
    double a = contourArea(contours[i]); //각각
```

contourArea를 통해 각 점의 영역을 구하고

```
if (a > 600.0) {  //어느 정도 크기가 된다
사각형 그림
   rect = boundingRect(contours[i]);
   rectangle(origin, rect, color, 2);
}
```

그 픽셀 영역이 600으로 어느 정도 크기가 된다는 것만 뽑아서 사각형을 만들고 rectangle 함수를 통해 사각형을 원본에 그리는 것을 만든 것입니다.

```
if(s== "neon") neon_rect= rect;
else if(s=="orange") orange_rect = rect;
else if(s=="blue") blue_rect = rect;
else if(s=="white") white_rect = rect;
return origin;
```

그리고 이 사각형을 저장해야 pos_cal에서 이를 통해 위치를 비교할 수 있으므로 어떤 것을 구한 것인지를 호출하는 곳에서 스트링으로 넘겨주어 그것을 통해 전역으로 선언한 Rect 객체에 저장하는 것으로 코딩한 것입니다.

```
int cx = neon_rect.x + neon_rect.width / 2; /
int cy = neon_rect.y + neon_rect.height / 2;
```

pos_cal의 구현부분입니다. 네온 꼬깔 인식을 한 사각형 전역 객체를 통해 중간 좌표를 구했습니다. 그리고 이 중간 좌표를 통해 파란선과 흰색선의 좌표와 비교하며 비교를 하는 것입니다.

```
if (cy < blue_rect.y) {
    ui->neon_ver->setText("up");
}
else if (cy > blue_rect.y + blue_rect.height) {
```

위의 사진처럼 파란선보다 위에 있으면 up으로 관련 lineEdit을 setText로 출력하도록 했습니다. 아래에 있다면 else if 로 down을 출력하게 하고 만약 위 2개가 아닌 경우는 파란선 사이에 있는 경우이기에 else로 center를 출력하게 한 것입니다.

```
if (cx < white_rect.x) {
    ui->neon_hor->setText("left");
}
else if (cx > white_rect.x + white_rect.width) {
```

위 사진의 경우에는 네온 꼬깔에 대해 흰선을 기준으로 또 판단하는 경우입니다. x좌표를 통해 흰 선의 왼쪽 줄보다 값이 작으면 왼쪽, 그리고 오른쪽 줄보다 x좌표가 크면 오른쪽, 그게 아니라면 center로 출력하게 하였습니다.

```
cx = orange_rect.x + orange_rect.width / 2; /
cy = orange_rect.y + orange_rect.height / 2;
```

그리고 오렌지 꼬깔의 경우에도 같은 방식으로 해주었습니다

```
blue_slide.flag = false;
neon_slide.flag  = false;
orange_slide.flag = false;

white_slide.flag=true;
```

라디오 박스를 클릭했을 때 호출되는 메서드 중 white_click 메서드 내부입니다. slider 객체의 flag를 white의 경우에만 true로 한 것입니다. 그 이유는 white_line이라는 라디오 버튼을 누른 것이고 이는 즉 white_slide에 대한 슬라이드 값만 바꿀 수 있고 또 그 값이 white_slide에 저장된다고 하기 위해서입니다.

```
ui->hue_high->setValue(white_slide.huehigh);
ui->hue_low-> setValue(white_slide.huelow);
```

그리고 라디오 박스를 눌렀으니 슬라이더를 모두 최신화해줘야 하므로 slider 객체에 있는 값들로 모두 슬라이더 값을 초기화해주었습니다. 이를 hue_~~말고도 saturation_high, low 그리고 value_high, value_low에 대해서도 각각 해주었습니다. 그리고 각 라디오 박스에 대한 각 슬롯 메서드가 존재하기에 각 슬롯 메서드에 대해 같은 원리, 같은 방식으로 작업을 해주었습니다.

```
slider* t;
//여기서 flag 변수 사용함 어떤 것의 hsv변수가 바뀌는
if(neon_slide.flag==true) t= &neon_slide;
```

any_slider_changed라는 메서드 내부입니다. slider 포인터를 선언했습니다. 그 이유는 라디오 박스를 누른 후에 슬라이더를 ui로 변경하게 될 때 그 변경한 것을 어디에 저장할 것인가를 포인터를 통해 그 객체를 가리키기 위함입니다. 그래서 if 문에서 flag가 true일 경우에 그 객체를 포인터로 잡아서 바뀐 슬라이드 값을 그 객체의 hsv high, low값을 저장하는 멤버에 각각 넣어주는 것입니다.

```
int hl = ui->hue_low->value();
int hh = ui->hue_high->value();
int sl = ui->saturation_low->value();
```

그래서 이처럼 슬라이더의 값을 각각 가져옵니다.

```
t->huelow = hl;
t->huehigh = hh;
t->satlow = sl;
```

그렇게 가져온 값을 포인터를 통해 멤버에 접근하여 값을 각각 대입해주는 것입니다. 이로써 이 슬롯 메서드는 마무리되며 각각의 슬라이더에 각각의 슬롯 메서드를 만들지 않았으므로 하나의 슬라이더가 바뀔 때마다 모든 슬라이더의 값을 모든 멤버에 대입해주는 코드를 짠 것입니다. 그리고 이 슬롯 메서드의 시그널을 sliderMoved로 설정한 이유는 setValue라는 것을 라디오 박스의 슬롯 메서드에서 사용하기 때문입니다. setValue를 하면 valueChanged라는 시그널이 발생하게 되는데 이와 같은 시그널을 이 슬롯 메서드에 사용하게 된다면 실행이 꼬이게 되어 정상적으로 작동하지 않았습니다. 그래서 sliderMoved로 바꾸어 코딩하게 되었습니다.

아래는 제가 이번 과제를 하며 참고한 사이트들입니다. 감사합니다.

https://yonghello.tistory.com/entry/Open-CV-%ED%95%A8%EC%88%98%EC%A0%95%EB%A6%AC

https://m.blog.naver.com/dorergiverny/223077740655

https://blog.naver.com/dorergiverny/223077650900

https://m.blog.naver.com/sees111/222343911932