증강현실
Image Processing 과제

경북대학교 컴퓨터학부
2014105018
김성현

1. 과제 개요

opencv를 활용하여 image processing을 해보는 것이 목표이다.

일단 첫 번째 프로젝트는 대상이 되는 사진에 사각형을 그리고 그 안의 사각형을 반전, 회전, 크기조절을 해보는 것이 목표이다.

두 번째 프로젝트는 웹캠을 이용하여 사람의 손을 인식하는 것이 목표이다.

세 번째 프로젝트는 두 번째 프로젝트를 이용하여 가위, 바위, 보를 판단하는 것이 목표이다.

2. 프로젝트 환경

프로젝트는 Microsoft Windows 10 운영체제에서 Microsoft Visual Studio 2013 버전을 이용하여 진행하였으며, OpenCV는 3.0버전을 사용하였다.

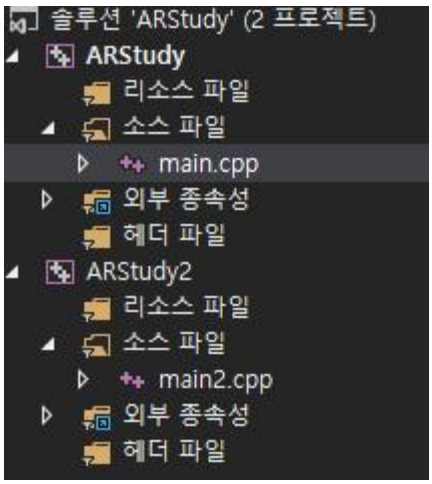
3. 사용한 사진

첫 번째 프로젝트에는 이 그림파일을 이용하였다.

4. 사용한 웹캠

이 프로젝트에 사용된 웹캠은 삼성전자 아티브 NT450R5G-X58L 노트북의 웹캠이다.

5. 프로그램 파일



ARStudy 프로젝트의 main.cpp에 첫 번째 프로젝트가 구현되었으며, ARStudy2 프로젝트의 main2.cpp에 두 번째 프로젝트가 구현되었다.

6. 첫 번째 프로젝트

1) 프로젝트 선언 부분

```
#include <iostream>
#include <fstream>
#include <cmath>

#include "opencv2₩highgui₩highgui.hpp"
#include "opencv2₩opencv.hpp"

#pragma comment(lib, "opencv_world300d.lib")

const double PI = 3.14159265;

using namespace std;
using namespace cv;

bool bLBDown = false; // 마우스 버튼 눌렀는지 체크
bool checkDrag; // 드래그가 이루어졌는지 체크
CvRect box; // 드래그로 그린 박스
```

opencv에서 필요한 라이브러리를 추가하고, 회전을 위해 cmath와 PI를 선언하였다.
그 외 flag와 사각형으로 쓸 box를 선언하였다.

2) 마우스로 사각형 그리기

```
// 사각형 그리기
void draw_box(IplImage* img, CvRect rect)
{
    cvRectangle(img, cvPoint(rect.x, rect.y),
        cvPoint(rect.x + rect.width, rect.y + rect.height),
        cvScalar(0xff, 0x00, 0x00));
}
```

이 부분은 마우스로 드래그한 정보에 따라 생성된 box 정보를 이용하여 그림에 사각형을 그리는 함수이다.

```
// 마우스 드래그
void on_mouse(int event, int x, int y, int flag, void* params)
{
    IplImage* image = (IplImage*)params;

    if (event == CV_EVENT_LBUTTONDOWN){ // 왼쪽 버튼 눌렀을 시, 박스 초기화
        bLBDown = true;
        box = cvRect(x, y, 0, 0);
    }
    else if (event == CV_EVENT_LBUTTONUP){ // 왼쪽 버튼 눌렀다가 뗐을 때, 박스의 넓이, 높이를 설정한다.
        bLBDown = false;
        checkDrag = true;
        if (box.width < 0)
        {
            box.x += box.width;
            box.width += -1;
        }
        if (box.height < 0)
        {
            box.y += box.height;
            box.height += -1;
        }
        draw_box(image, box);
    }
    else if (event == CV_EVENT_MOUSEMOVE && bLBDown){ // 드래그 중에는 박스의 넓이, 높이를 갱신한다.
        box.width = x - box.x;
        box.height = y - box.y;
    }
}
```

이 함수는 마우스 동작을 인식하는 함수이다. 처음 왼쪽 버튼을 눌렀을 때 인식하여 드래그 할 때 계속 사각형 정보를 갱신하다가 왼쪽 버튼에서 손을 떼었을 때 사각형을 그리게 한다.

이 함수는 setMouseCallback 이란 opencv에서 지원하는 함수를 통해 사용된다.

3) 이미지 복사

```
// 박스내 이미지 복사
Mat copyBoxMat(Mat source)
{
    return source(box);
}
```

사각형이 그려졌을 때 사각형 안의 이미지를 변형시키기 위해 사각형 속의 이미지를 따로 빼낼 필요가 있었다. 그리하여 OpenCV에서 이미 지원하는 함수를 사용했으며 사용법은 대상이 되는 이미지가 source, 사각형이 box라 하였을 때 source(box)

로 사용할 수 있다.

```
// 이미지 복사
Mat copyMat(Mat source)
{
    // source의 Mat를 result로 복사하는 작업
    // opencv에 이미 구현이 되어있는 작업이다.
    // source.copyTo(result);
    Mat result = Mat::zeros(source.size(), source.type());
    for (int i = 0; i < source.cols; i++){
        for (int j = 0; j < source.rows; j++){
            result.at<Vec3b>(j, i) = source.at<Vec3b>(j, i);
        }
    }

    return result;
}
```

이 부분은 전체 이미지를 복사하는 함수이다. 반복문을 통해 픽셀마다 접근하여 복사를 하는 함수이다. 이 기능은 이미 opencv에 구현이 되어있으며 원본 이미지가 source, 결과 이미지가 result라고 했을 때 source.copyTo(result)와 같이 사용가능하다.

4) 반전

```
// y축반전
Mat yReflecting(Mat source)
{
    Mat result = copyMat(source);

    for (int i = 0; i < box.width; i++){
        for (int j = 0; j < box.height; j++){
            result.at<Vec3b>((box.y + j), (box.x + i)) = source.at<Vec3b>(box.y + j, (box.width + box.x - 1) - i);
        }
    }

    return result;
}

// x축반전
Mat xReflecting(Mat source)
{
    Mat result = copyMat(source);

    for (int i = 0; i < box.width; i++){
        for (int j = 0; j < box.height; j++){
            result.at<Vec3b>((box.y + j), (box.x + i)) = source.at<Vec3b>((box.height + box.y - 1) - j, (box.x + i));
        }
    }

    return result;
}
```

두 함수는 x축, y축에 대해 반전을 하는 함수이다. 반복문을 통하여 픽셀단위로 접근하여 반전을 실행한다. 사각형의 크기와 사각형의 x, y 좌표를 이용하여 이미지의 사각형 부분에 접근하여 반전을 실행하였다.

5) 회전

```
// 회전
Mat rotating(Mat source, double degree)
{
    Mat result = copyMat(source);

    int x0 = box.x + (box.width / 2);
    int y0 = box.y + (box.height / 2);
    double cosd = cos(degree*PI / 180);
    double sind = sin(degree*PI / 180);

    // 원본에 덮어씌우는 부분으로 인해 왼쪽 90도, 오른쪽 90도만 가능
    for (int i = 0; i < box.width; i++){
        for (int j = 0; j < box.height; j++){
            int x1 = (box.x + i);
            int y1 = (box.y + j);
            int x = ((cosd * (x1 - x0)) - (sind * (y1 - y0)) + x0);
            int y = ((sind * (x1 - x0)) - (cosd * (y1 - y0)) + y0);
            result.at<Vec3b>(y, x) = source.at<Vec3b>((box.y + j), (box.x + i));
        }
    }

    return result;
}
```

회전은 아래의 식을 이용하여 진행하였다.

$$x_2 = \cos\theta \times (x_1 - x_0) - \sin\theta \times (y_1 - y_0) + x_0$$

$$y_2 = \sin\theta \times (x_1 - x_0) - \cos\theta \times (y_1 - y_0) + y_0$$

sin값과 cos값은 cmath를 이용하여 계산하였으며, 미리 선언한 PI값도 이용하였다. x0, y0는 사각형의 중앙 좌표 값을 계산하여 입력하였다.

6) 확대

```
// 확대
Mat scaling(Mat source, Mat boxMat, double scale)
{
    Mat result = copyMat(source);
    Mat scaleBoxMat;

    // 사각형 안의 Mat의 크기를 scale배 늘린다.
    int boxWidth = (int)(boxMat.size().width * scale);
    int boxHeight = (int)(boxMat.size().height * scale);
    cv::resize(boxMat, scaleBoxMat, Size(boxWidth, boxHeight));

    // 붙여넣을 때 시작 위치 정보를 갱신한다.
    int x = box.x - (box.width / 2);
    int y = box.y - (box.height / 2);

    for (int i = 0; i < boxWidth; i++){
        for (int j = 0; j < boxHeight; j++){
            result.at<Vec3b>((y + j), (x + i)) = scaleBoxMat.at<Vec3b>(j, i);
        }
    }

    return result;
}
```

확대는 opencv에서 제공하는 함수를 이용하여 처리하였다. resize함수를 이용하여 boxMat이란 사각형 내부의 이미지만 따로 뺀 Mat 변수를 scale배 확대하여 그것을 기존 이미지에 붙여넣는 식으로 구현하였다.

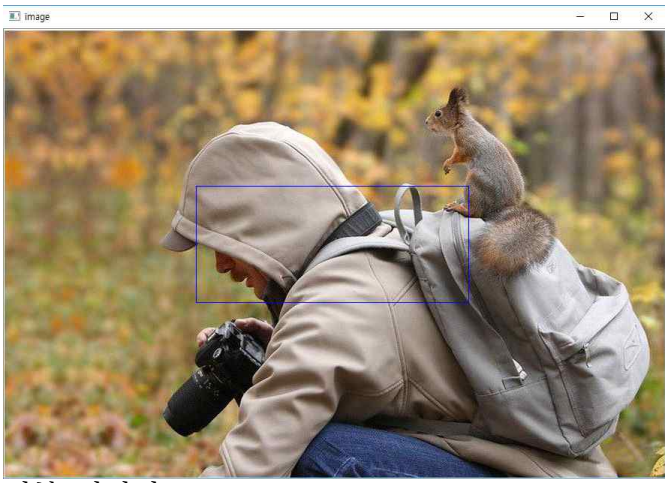
7) 메인함수

```
int main()
{
    IplImage copy;
    IplImage * resultImage;
    Mat resultMat, xReflectMat, yReflectMat, leftRotateMat, scaleMat, boxMat;

    // 이미지 불러오기
    Mat gMatImage = imread("./picture/pic.jpg", 1);

    // Mat 이미지를 IplImage 로 복사한다.
    copy = gMatImage;
    resultImage = &copy;


    checkDrag = false;
    namedWindow("image");
    setMouseCallback("image", on_mouse, resultImage);
    cvShowImage("image", resultImage);

    //드래그 대기
    while (!checkDrag){
        waitKey(100);
    }
    cvShowImage("image", resultImage);

    //사각형 추가된 사진 저장
    resultMat = cvarrToMat(resultImage);
    boxMat = copyBoxMat(resultMat);
    cout << box.x << ' ' << box.y << ' ' << box.width << ' ' << box.height << endl;

    yReflectMat = yReflecting(resultMat); // y축 반전
    xReflectMat = xReflecting(resultMat); // x축 반전
    scaleMat = scaling(resultMat, boxMat, 1.5); // 크기 변경
    leftRotateMat = rotating(resultMat, -90.0); // 90도 회전

    waitKey(2000);
    imshow("y반전 이미지", yReflectMat);
    imshow("x반전 이미지", xReflectMat);
    imshow("왼쪽 90도 회전 이미지", leftRotateMat);
    imshow("1.5배 확대 이미지", scaleMat);

    waitKey(0);

    return 0;
}
```

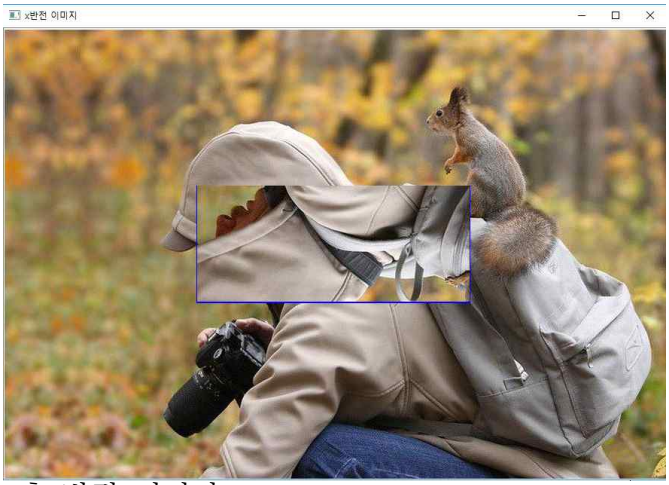
메인 함수는 일단 이미지를 불러와 드래그를 진행하여 사각형을 그리기 위해 Mat 형태의 이미지를 IplImage형태로 바꾼다. 그 후 마우스 이벤트를 입력받으며 드래그

가 되기를 기다린다. 드래그가 진행되어 사각형 정보가 나오면 반전, 회전, 확대 함수를 실행하여 각각 이미지를 생성한 후 출력한다.

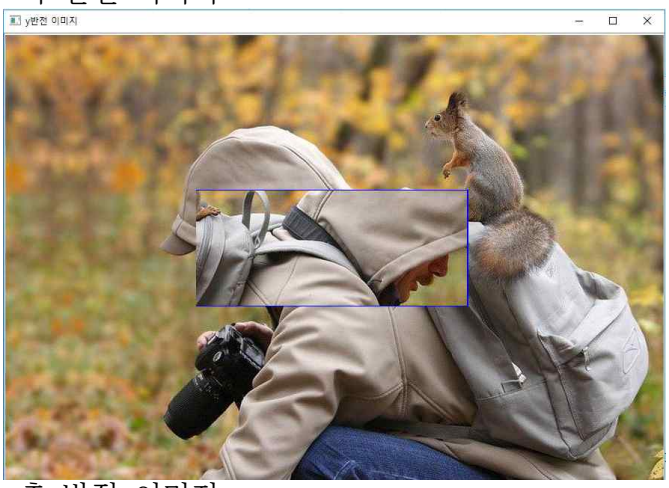
8) 실행화면



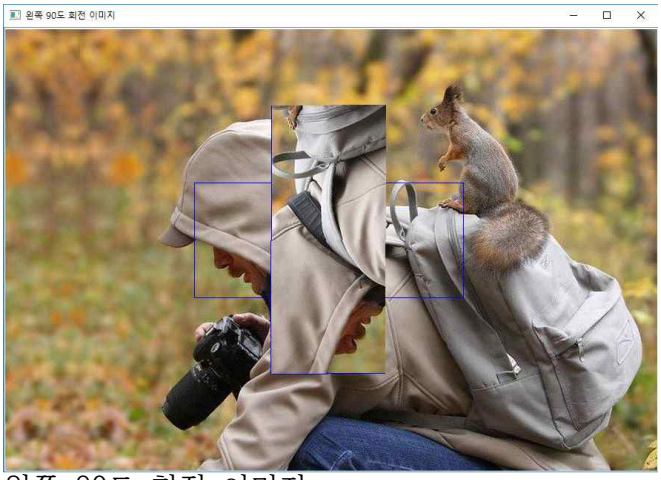
기본 이미지



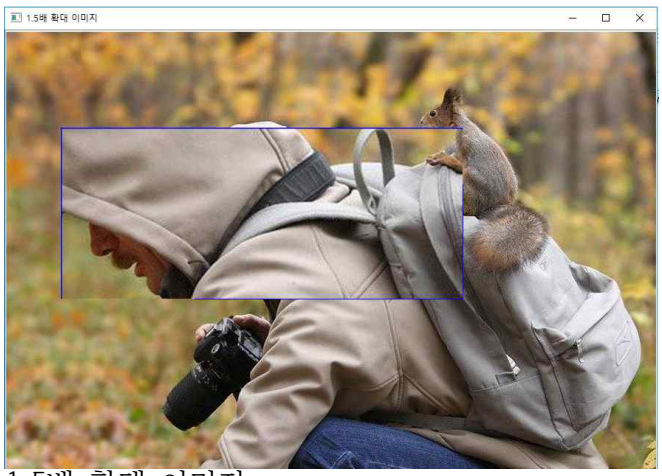
x축 반전 이미지



y축 반전 이미지



왼쪽 90도 회전 이미지



1.5배 확대 이미지

9) 실행영상

https://youtu.be/hBf1tNBh4zA

7. 두 번째, 세 번째 프로젝트

1) 프로젝트 선언 부분

```
#include <iostream>
#include <fstream>
#include <cmath>
#include <vector>

#include "opencv2₩highgui₩highgui.hpp"
#include "opencv2₩opencv.hpp"

#pragma comment(lib, "opencv_world300d.lib")

using namespace std;
using namespace cv;

// 주먹 범위
const int rock_low = 14;
const int rock_high = 20;

// 가위 범위
const int scissors_low = 21;
const int scissors_high = 29;

//보 범위
const int paper_low = 31;
const int paper_high = 41;

enum rsp {rock, scissors, paper, none};
```

opencv에서 필요한 라이브러리를 추가하였고, 가위, 바위, 보를 판단하기 위한 픽셀 퍼센트에 대한 정보를 미리 구해 정의 하였다.

코드의 가독성을 위해 enum형태를 정의하여 사용하였다.

2) 초기 픽셀 체크

```
// 살색 픽셀 수 체크
int checkNoise(int * noise, int noise_index, int pixel)
{
    // 프로그램 시작 후 10 번만 실행한다.
    if (noise_index > 10){
        return noise_index;
    }
    // 10번 실행 후 살색 픽셀 수의 평균값를 계산하여 살색 픽셀수를 계산해 놓는다.
    else if (noise_index == 10){
        *noise = *noise / noise_index;
        return noise_index + 1;
    }

    *noise += pixel;
    return noise_index + 1;
}
```

프로그램을 실행하면 분명 화면에 내 손이 없음에도 불구하고 픽셀수가 0이 아닌

경우가 있다. 손의 색과 비슷한 색을 가진 물체들이 원인인데, 이것을 미리 계산하여 가위, 바위, 보 판단에 사용하기 위해 함수를 구현하였다. 프로그램을 시작한 후 10번 0이 아닌 픽셀의 퍼센트를 계산하였고, 그 평균을 노이즈 픽셀 퍼센트로 하였다.

3) 픽셀 수 체크

```
// 0이 아닌 픽셀 수 체크
int checkPixel(Mat pic)
{
    int ret = 0;
    // 0값을 넣은 픽셀 하나를 선언한다.
    Mat zpic = Mat::zeros(Size(1, 1), pic.type());

    // 0 픽셀과 비교하여 아니면 덧셈하여 정보가 있는 픽셀 수를 계산한다.
    for (int i = 0; i < pic.size().width; i++){
        for (int j = 0; j < pic.size().height; j++){
            if (pic.at<Vec3b>(j, i) != zpic.at<Vec3b>(0, 0)){
                ret++;
            }
        }
    }

    return (double)ret / (double)(pic.size().width * pic.size().height) * 100;
}
```

이 프로그램은 손의 색과 비슷한 색을 가진 픽셀의 데이터만을 남겨놓고 아니면 전부 0으로 만들어버린다. 그러므로 결과 데이터에서 0이지 않은 픽셀 수를 세면 손의 면적이 나오게 되는데, 그러기 위해 0인 픽셀 하나를 선언하여 이 픽셀과 다른 픽셀 수를 세어 비율을 계산하였다.

4) 가위, 바위, 보 판단

```
// 가위 바위 보 판단
rsp check_rsp(int pixel, int noise){
    // 기본적으로 계산해 놓은 데이터들에 의해 가위 바위 보를 판단한다.(픽셀수에 의해 결정된다)
    // 프로그램 시작시 계산한 노이즈 수를 더함으로써 노이즈에 의한 에러를 줄인다.
    if (noise + rock_low <= pixel && pixel <= noise + rock_high){
        return rock;
    }
    else if (noise + scissors_low <= pixel && pixel <= noise + scissors_high){
        return scissors;
    }
    else if (noise + paper_low <= pixel && pixel <= noise + paper_high){
        return paper;
    }

    return none;
}
```

여러 번의 시행착오로 가위, 바위, 보일 때의 화면 상의 픽셀 퍼센트를 계산한 값을 이미 선언을 해두었는데, 이것을 이용하여 현재 화면에 가위, 바위, 보 중 어느 것이 있는지 판단하는 함수이다. 이 함수에는 선언한 데이터 외에 노이즈 데이터도 들어가는데 현재 픽셀에는 프로그램 시작 시 처음부터 보이던 손의 색 픽셀도 들어있다

라는 가정 하에 선언한 데이터에 이 값을 더해 가위, 바위, 보 판단의 정확도를 높였다.

반환은 선언해둔 enum형식을 사용하였으며 코드의 가독성을 높였다.

5) 색 채우기

```
// 화면 색 채우기
void fillMat(Mat * mat, int n)
{
    // 필요로 하는 곳의 데이터를 255로 만들어 빨간, 파란, 초록 색 중 하나로 변환시킨다.
    Mat pic = Mat::zeros((*mat).size(), (*mat).type());
    for (int i = 0; i < (*mat).size().width; i++){
        for (int k = 0; k < (*mat).size().height; k++){
            pic.at<Vec3b>(k, i)[n] = 255;
        }
    }

    // 색을 입힌 이미지와 원래 mat이미지를 bitwise하여 기본 이미지의 색을 바꾼다.
    bitwise_and(*mat, pic, *mat);
}
```

가위, 바위, 보에 따라 손에 색을 입히는 작업을 하는 함수이다.

pic이라는 임의의 Mat 변수의 배열에 255를 입힘으로서 색을 설정할 수 있다. [0]는 빨간색, [1]은 초록색, [2]는 빨간색을 가리키며 한 곳이 255고 다른 곳이 0 이면 셋 중 한 색을 가리키게 된다.

이렇게 생성한 색 영상을 bitwise_and로 기존 이미지에 덮어씌우면 기존 이미지의 0이 아닌 부분은 해당 색으로 변하게 된다.

5) 메인 함수

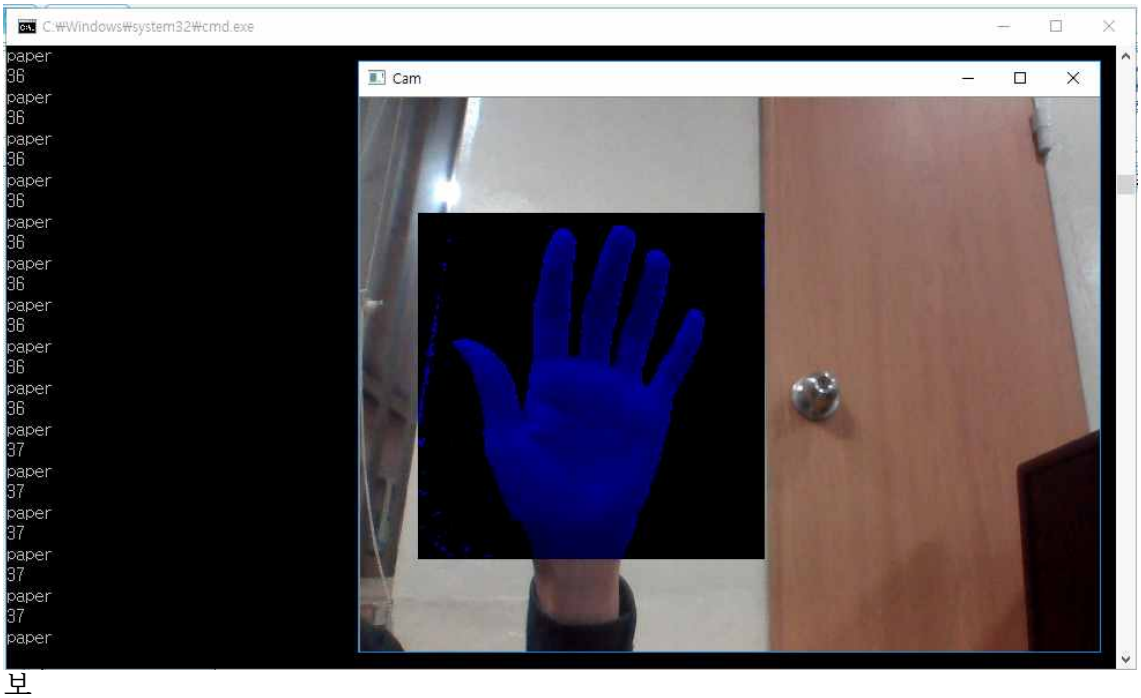
```
        // 각각 떼어낸 이미지에서 손의 색과 비슷한 부분만을 추출한다.
        Mat crMat, cbMat;
        inRange(splitMatV[1], Scalar(77), Scalar(150), crMat);
        inRange(splitMatV[2], Scalar(133), Scalar(173), cbMat);

        // bitwise_and를 이용하여 Cr과 Cb 정보중 남아 있는 것를 and 연산하여 손의 이미지를 확정짓는다.
        Mat grayMat;
        bitwise_and(crMat, cbMat, grayMat);

        // 이미지를 BGR형태로 변환
        Mat bgrMat;
        cvtColor(grayMat, bgrMat, CV_GRAY2BGR);

        // 손의 픽셀 정보를 담고있는 bgrMat과 원래 이미지를 담고있는 frame을 and연산하여 손의 이미지만 남긴다.
        Mat handMat;
        bitwise_and(bgrMat, frame, handMat);

        // 손을 인식하여 손의 부분의 픽셀 개수를 센다.
        pixel = checkPixel(handMat);
        cout << pixel << endl;

        // 살색 픽셀 수 체크
        noise_index = checkNoise(&noise, noise_index, pixel);
```

메인 함수는 일단 손을 인식할 범위인 Box를 선언한다. 그 후 해야할 일을 while 문으로 무한정 돌리게 된다. while문 안에서는 우선 웹캠 화면을 받아온 뒤, 손을 인식할 부분을 따로 떼어낸다. 이 부분의 영상을 YCrCb 형식으로 바꾼 뒤, split함수로

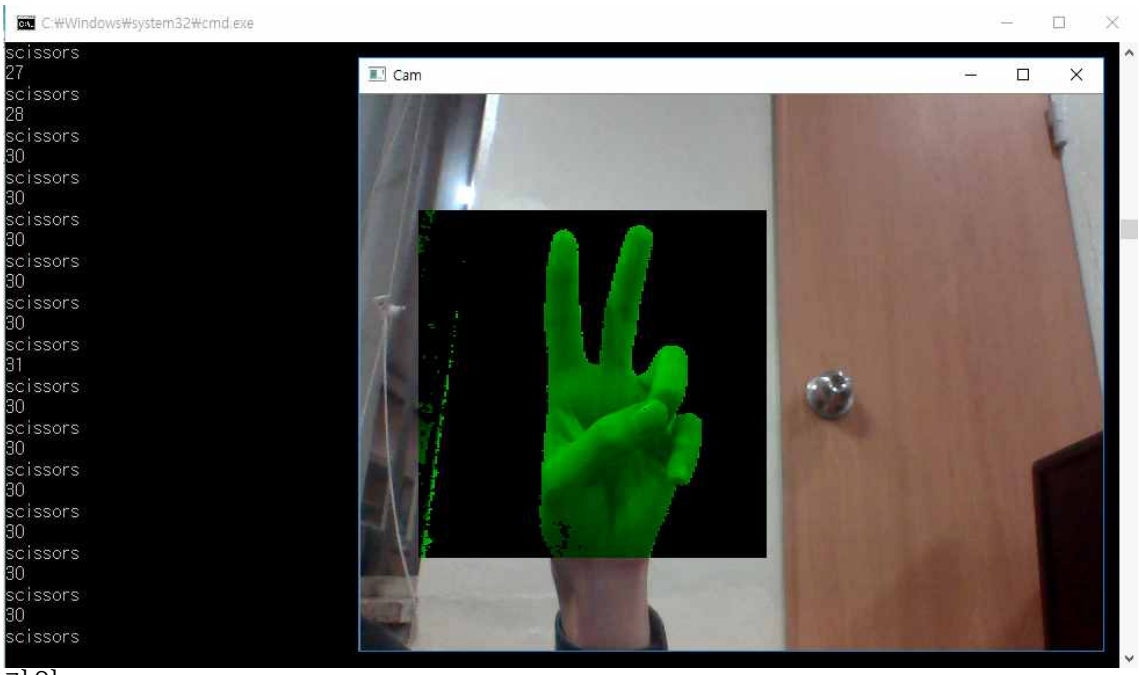
```
int main()
{
    VideoCapture capture(0);

    // 실제로 사람의 손을 인식할 사각형 범위를 지정할 사각형
    Rect box;
    box.x = 50;
    box.y = 100;
    box.height = 300;
    box.width = 300;

    int noise_index, noise, pixel;
    noise_index = 0, noise = 0;

    while (true) {
        // 웹캠 정보를 받아온다.
        Mat frame, camFrame;
        capture >> camFrame; // get a new frame from webcam
        frame = camFrame(box); // 웹캠 정보에서 손을 인식할 부분만 따로 떼어낸다.

        // 기존 이미지를 YCrCb 3개의 정보를 저장한 형태로 변환
        Mat rgbMat, yCbCrMat;
        cvtColor(frame, yCbCrMat, CV_RGB2YCrCb);

        // split으로 3개가 합쳐진 정보를 떼어낸다.
        // splitMatV[0] : Y, splitMatV[1] : Cr, splitMatV[2] : Cb
        vector <Mat> splitMatV;
        split(yCbCrMat, splitMatV);
        // 픽셀 수와 노이즈 정보를 이용하여 가위 바위 보를 인식하여 그에 영상에 색을 입힌다.
        rsp temp = check_rsp(pixel, noise);
        if (temp == rock){
            cout << "rock" << endl;
            fillMat(&handMat, 2);
        }
        else if (temp == scissors){
            cout << "scissors" << endl;
            fillMat(&handMat, 1);
        }
        else if (temp == paper){
            cout << "paper" << endl;
            fillMat(&handMat, 0);
        }

        // 색까지 입혀 완성된 이미지를 원본 이미지 frame에 입혀 원본 이미지를 변환 시킨다.
        addWeighted(frame, 0.0, handMat, 1.0, 0.0, frame);
        imshow("Cam", camFrame);

        if (waitKey(50) >= 0) break;
    }

    return 0;
}
```

Y, Cr, Cb 세 가지로 나눈다. 이 중, Cr, Cb에서 손의 색 부분만을 추출한 뒤, 두 영상을 bitwise_and함수를 이용하여 합친다. 이 영상을 다시 BGR 형식으로 바꾼 뒤 원래 영상과 합치면 일단 손만을 추출해낸 영상이 나온다. 그 후 영상의 픽셀 수를 계산한 뒤, 처음 10번 동안은 영상의 손의 색 픽셀을 계산하게 된다. 이후 계산한 픽셀 수를 기반으로 가위, 바위, 보를 판단하여 색을 입힌 뒤 이것을 원본 영상에 씌운 후 영상을 출력하게 된다.

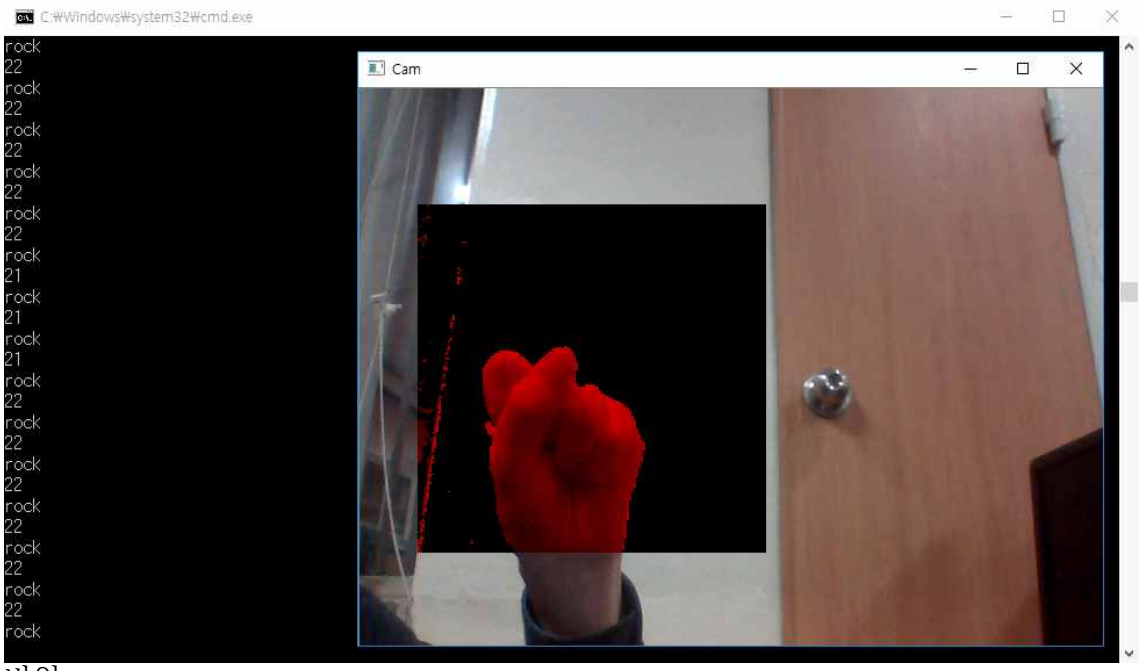
6) 실행 화면


C:₩Windows₩system32₩cmd.exe
paper
36
paper
36
paper
36
paper
36
paper
36
paper
36
paper
36
paper
36
paper
36
paper
36
paper
37
paper
37
paper
37
paper
37
paper
37
paper
Cam

보


C:₩Windows₩system32₩cmd.exe
scissors
27
scissors
28
scissors
30
scissors
30
scissors
30
scissors
30
scissors
30
scissors
31
scissors
30
scissors
30
scissors
30
scissors
30
scissors
30
scissors
30
scissors
Cam

가위



바위

7) 실행 영상

https://youtu.be/7Pwxm_fOAFE