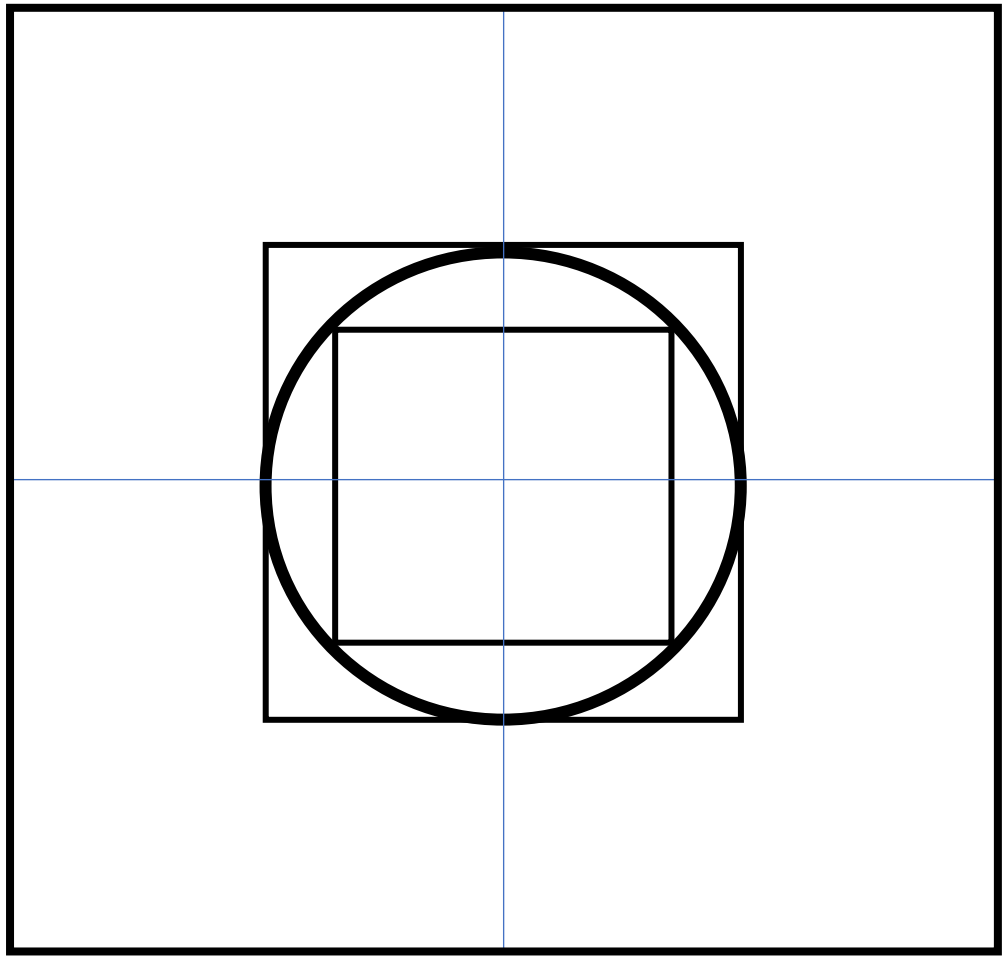
# 바둑알 검출 프로그램

컴퓨터과학과
201733005
김신영

# 목차

# 1. 프로젝트 이해 배경지식 – 사용한 사각형 영역들



사각형 1 – 바둑알 내부의 사각형

사각형 2 – 바둑알을 감싸는 사각형

사각형 3 – 바둑알을 둘 수 있는 한 칸의 범위를 나타내는 사각형.

# 2. 바둑알에 대한 정보 얻기 – 바둑알 검출하기

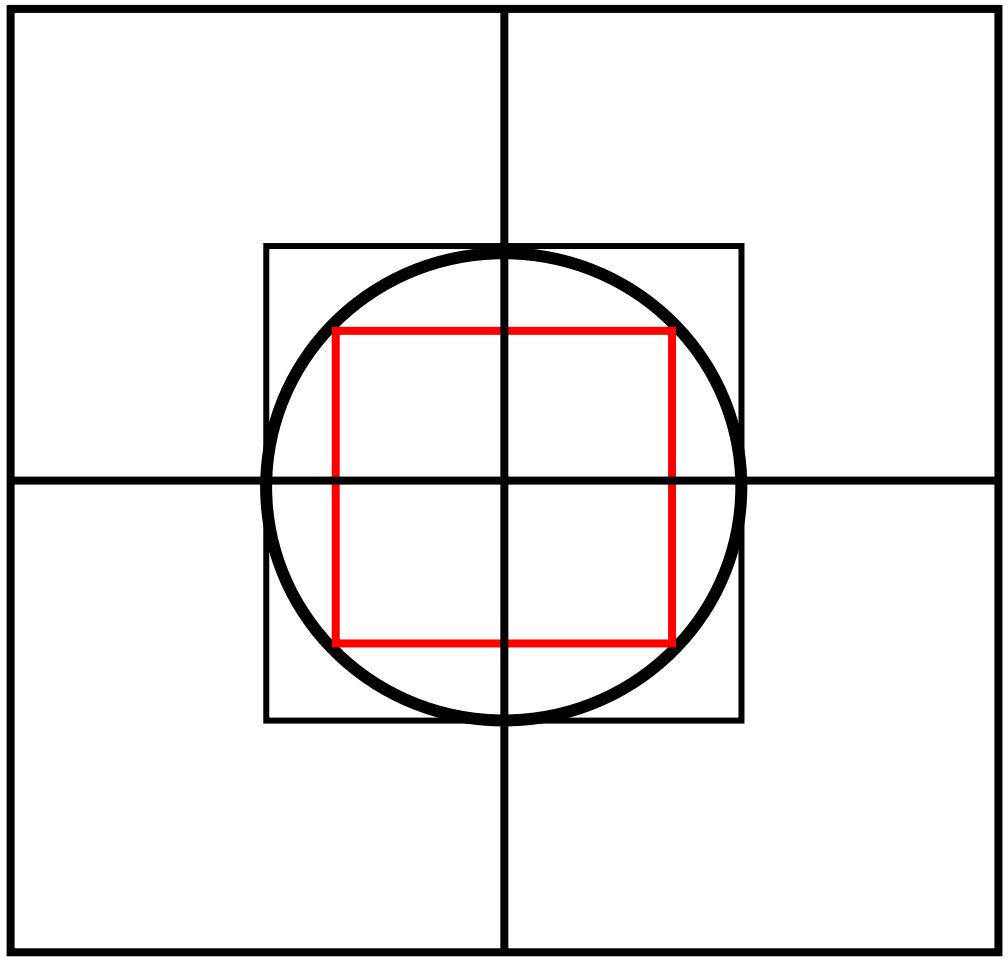
```
void go_find_eggs(string link, int cnt)
{
    Mat src = imread(link, IMREAD_GRAYSCALE);

    if (src.empty()) {
        cerr << "Image load failed!" << endl;
        return;
    }

    Mat blurred;
    blur(src, blurred, Size(3, 3));
    Mat dst;
    cvtColor(src, dst, COLOR_GRAY2BGR);

    // 바둑알 검출
    vector<Vec3f> circles;
    HoughCircles(blurred, circles, HOUGH_GRADIENT, 1, 1, 150, 30, 10, 12);
```

링크로 받아온 이미지 파일을 읽기. 허프 원 검출을 사용하여 circles 벡터에 모든 원에 대한 정보 저장.

처음 원이 잘 검출되지 않고, 큰 원들까지 검출되어 여러 파라미터 값들을 변경하는 데에 시간을 엄청 소비.

결국 원의 최소, 최대 반지름 값인 마지막 두 번째 파라미터만 수정하면서 해결.

# 2. 바둑알에 대한 정보 얻기 – 흑백 판단



바둑알 내부의 사각형을 구하고, 그 사각형에 포함되는 모든 픽셀들의 그레이스케일 값의 평균을 구하려고 했음.

바둑알이 짝수 개일 경우 그레이스케일 값 상위 n/2개를 백, 나머지를 흑으로 처리.

바둑알이 홀수 개일 경우 흑의 선공(보통 바둑은 흑이 선공이므로) 기준으로 그레이스케일 값 하위 (n+1)/2개를 흑, 나머지를 백으로 처리.

평균 그레이스케일 값 구하는 데에 실패.

사각형을 그려보고 숫자의 영향을 적게 받는 점 clean_gray를 생각했음.

원의 중심으로부터 내부 사각형의 한 면의 절반 만큼 y + 방향으로 수직이동한 점이 clean_gray

# 2. 바둑알에 대한 정보 얻기 – 흑백 판단

```
vector<Vec3f> egg(circles.size(), Vec3f(3, 0));
int eggcount = 0;

// 검출한 원 정보 egg벡터로 저장하기.
for (Vec3f c : circles) {
    int dotx = cvRound(c[0]) - cvRound(c[2]) / r2;
    int doty = cvRound(c[1]) - cvRound(c[2]) / r2;
    int radius = cvRound(c[2]);

    // 바둑알
    Rect in_egg_rect(dotx, doty, 2 * radius / r2, 2 * radius / r2);

    int size_egg_rect = in_egg_rect.height * in_egg_rect.width;

    Point clean(cvRound(c[0]), cvRound(c[1]) - radius / r2);
    drawMarker(dst, clean, Scalar(0, 0, 255), 0, 2, 2, 8);
    int clean_gray = blurred.at<uchar>(clean);

    if (clean_gray < 128) {
        egg[eggcount].val[0] = c[0] - c[2];
        egg[eggcount].val[1] = c[1] - c[2];
        egg[eggcount].val[2] = 0;
        rectangle(dst, in_egg_rect, Scalar(0, 0, 255), 0.6);
    }
    else {
        egg[eggcount].val[0] = c[0] - c[2];
        egg[eggcount].val[1] = c[1] - c[2];
        egg[eggcount].val[2] = 1; // 백돌에 1 부여.
        rectangle(dst, in_egg_rect, Scalar(255, 0, 0), 0.6);
    }
    eggcount++;
}
```

바둑알의 흑백 정보를 포함하여 담을 egg 벡터 생성.

바둑알 내부의 사각형 정보인데, 위에서 설명했듯이 사용하는 데에 실패함.

원 내부에서 숫자의 영향을 적게 받는 점 Point clean의 그레이스케일 값을 int clean_gray값으로 저장한다.

값이 128보다 작으면 흑돌로 판단하여 val[2]값에 0을,
값이 128보다 크면 백돌로 판단하여 val[2]값에 1을 부여한다.

val[0]에는 원의 x좌표에서 반지름 길이를 뺀 값,
val[1]에는 원의 y좌표에서 반지름 길이를 뺀 값을 저장한다.
원을 감싸는 사각형을 그리기 위한 준비

# 3. 바둑판에 대한 정보 얻기 – 바둑판의 네 꼭지점

```
// 바둑판 검출
Mat gray = src.clone();

Mat bin;
threshold(gray, bin, 200, 255, THRESH_BINARY_INV | THRESH_OTSU);

vector<vector<Point>> contours;
findContours(bin, contours, RETR_EXTERNAL, CHAIN_APPROX_NONE);

// 근사화된 외곽선 저장할 벡터.
vector<Point> approx;
for (vector<Point>& pts : contours) {
    if (contourArea(pts) < 400)
        continue;
    // 근사 점 마다 벡터에 저장.
    approxPolyDP(pts, approx, arcLength(pts, true) * 0.02, true);
}
for (int i = 0; i < 4; i++) {
    cout << approx[i]; // 좌상, 좌하, 우하, 우상 순서임.
}
int fx = (int)approx[3].x;
int ly = (int)approx[3].y;
cout << fx << " " << ly << endl;
```

프로젝트 초기에는 마우스 클릭 이벤트로 좌표를 얻는 방식으로 사각형의 네 좌표를 구했는데, 이후 외곽선 검출 방법으로 사각형을 검출해서 바둑판을 검출하는 방식을 추가했다.

마우스로 검출한 사각형 꼭지점은
(33,33), (33, 467), (467, 33), (467, 467)

외곽선 검출과 근사화를 통해서 바둑판 사각형의 네 꼭지점 좌표 구하기.

네 꼭지점을 출력하여 확인해 보았다.

# 3. 바둑판에 대한 정보 얻기 – 바둑판의 모든 빈칸(점)들

```
vector<Vec2i> dot(19 * 19, Vec2i(2, 0));
int dotcount = 0;
int a = (467 - 33) / 18;
for (int i = 33; i < 467; i += a) {
    for (int j = 33; j < 467; j += a) {
        dot[dotcount].val[0] = j;
        dot[dotcount].val[1] = i;
        dotcount++;
    }
}
```

위에서 구한 fx 값과 ly 값을 사용하면 완벽하게 작동되지 않아, 467과 33을 직접 사용했다. 이 점은 해결을 못했다.

점마다 두 개의 값을 저장할 수 있는 dot 벡터를 생성하였고, (33,33)에서 시작하여 (467,467)까지 한 칸의 길이 만큼 더하면서 각 점을 dot 벡터에 저장하였다.

# 지금까지 얻어낸 정보들

바둑알을 감싸는 사각형의 왼쪽 위 꼭지점-> vector<Vec3f> egg

바둑알의 흑백 여부 -> egg.val[2]

바둑알이 위치할 수 있는 바둑판의 점들 -> vector<Vec2i> dot

# 4. 점과 흑백 구분하여 출력 구현

```
int dotnumber = 0;
for (Vec2i d : dot) {
    dotnumber++;
    cout << ".";
    fout << ".";
    Rect blank_area(d.val[0] - a, d.val[1] - a, a * 2, a * 2);
    rectangle(dst, blank_area, Scalar(0, 0, 255), 1, 8);
    for (Vec3f e : egg) {
        Rect egg_area(e.val[0] - e.val[2], e.val[1] - e.val[2],
            average_r * 2 + 2.5, average_r * 2 + 2.5);
        rectangle(dst, egg_area, Scalar(0, 0, 0), 1, 8);
        Rect k = blank_area & egg_area;
        if (k == egg_area) {
            if (e.val[2] == 0) {
                cout << "\bb";
                fout << "\bb";
            }
            else if (e.val[2] == 1) {
                cout << "\bw";
                fout << "\bw";
            }
        }
    }
    if ((dotnumber % 19) == 0) {
        cout << "\n";
        fout << "\n";
    }
}
```

앞에서 얻어낸 바둑판의 모든 점들에 대해 for문으로 연산을 수행한다.

일단 빈칸으로 가정하여 점을 찍는다.

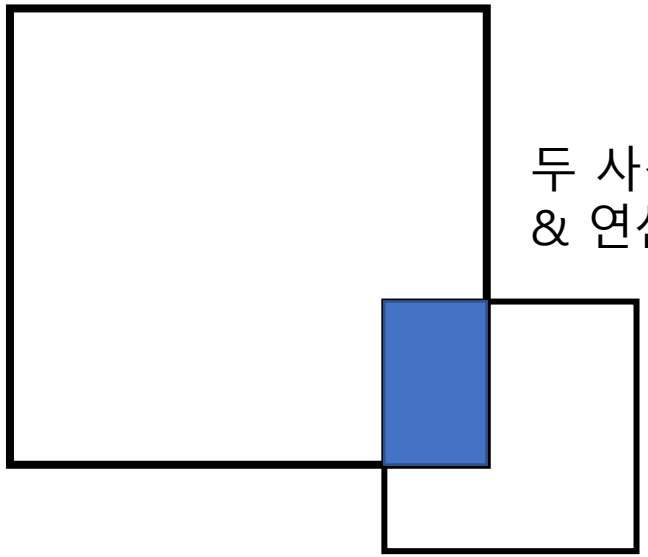
빈칸에 해당하는 영역인 큰 사각형 blank_area 사각형을 만든다.

한 점 마다 바둑알의 개수만큼 반복하여 바둑알의 존재 여부를 검사하게 되는데, & 연산을 사용하였다. 뒷장에서 자세히 설명.

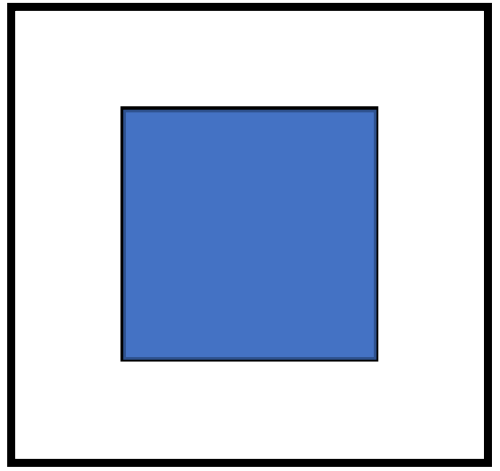
전에 바둑알 벡터의 val[2]값으로 저장해 두었던 흑백 여부를 활용하여 미리 출력해둔 점을 지우고 b나 w를 출력한다. 콘솔 창에서는 정확히 확인이 되는데 텍스트 파일로 내보내는 과정에서는 ₩b연산이 의도대로 되지 않아 점이 알파벳과 붙어서 출력되었다.

한 줄의 연산이 끝나면 줄 바꿈을 한다.

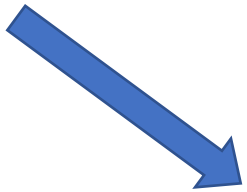
# 4. 점과 흑백 구분하여 출력 구현

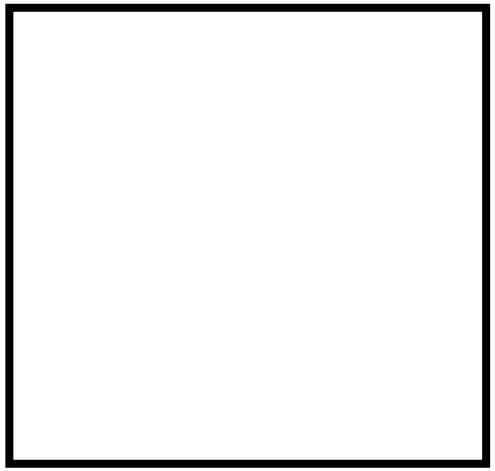


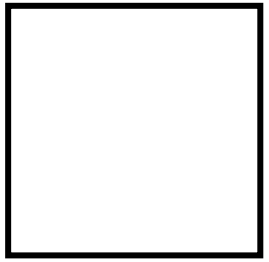
두 사각형에 대한
& 연산의 결과



영역 내에 존재하는 경우,
&연산의 결과는 내부의 사각형







영역 내에 존재하지 않는 경우,
&연산의 결과는 Rect(0,0,0,0)

# 4. 점과 흑백 구분하여 출력 구현

```
Rect k = blank_area & egg_area;
```

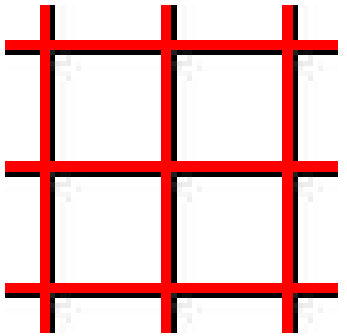
빈칸을 나타내는 blank_area 사각형과 바둑알의 범위를 나타내는 egg_area 사각형의 & 연산의 결과인 Rect k를 바탕으로 빈칸과 바둑알이 있는 칸 여부를 판단하게 된다.

```
if (k == egg_area)
```

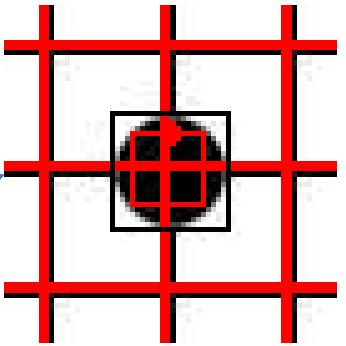
Rect k가 바둑알의 영역과 같다면, 바둑알의 영역이 빈칸의 영역 내부에 존재한다는 의미이다.

blank_area & egg_area
==
egg_area 인 경우

If문에 해당하지 않는 경우 점만 출력하고 다음 점으로 넘어간다.

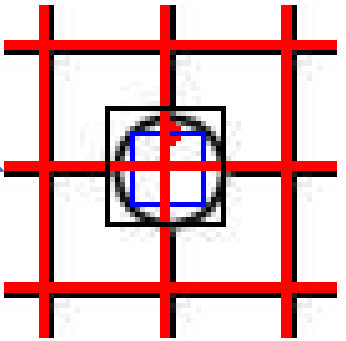


e.val[2] == 0



b 출력

e.val[2] == 1



w 출력

# 5. 특정 폴더의 모든 이미지 파일 처리하기 구현

```
String input_path = "C:\\Users\\newyo\\Documents\\OpenCV_goEgg_detecting\\egg_image_input\\*";
String output_path_txt = "C:\\Users\\newyo\\Documents\\OpenCV_goEgg_detecting\\egg_image_output\\img";


int main()
{
    String path(input_path);

    vector<String> str;

    glob(path, str, false);

    for (int cnt = 0; cnt < str.size(); cnt++) {
        go_find_eggs(str[cnt], cnt);
    }

    return 0;
}
```

접근하려는 폴더의 경로를 String 형태로 저장해 놓고, glob() 함수를 사용하여 폴더 내의 이미지 파일들을 벡터에 저장한다.

이미지 파일의 수 만큼 go_find_eggs() 함수를 수행하면 모든 이미지 파일에 대해 처리를 할 수 있다.

# 6. 처리한 이미지 파일들 특정 파일에 각기 다른 이름으로 저장하기 구현

```
String result_image_name = output_path_txt + to_string(cnt) + ".jpg";

vector<int> params;
params.push_back(IMWRITE_JPEG_QUALITY);
params.push_back(95);
imwrite(result_image_name, dst, params);
```

output_path 문자열에, 함수의 인자로 넘겨받은 함수의 반복 횟수와 .jpg 문자열을 포함하여 서로 다른 이미지 파일로 저장되도록 imwrite() 함수에 사용하였다.

# 7. 미흡한 점과 아쉬운 점

점을 구하는 반복문에 사각형 검출을 통해 얻어낸 x, y 좌표를 바로 사용할 경우 출력에 문제가 생기는데 원인을 찾지 못하여 해결을 못함. 좌표를 직접 입력해야 한다는 한계.

₩b(문자열 한 칸 지우기) 입력이 fout 방식에서는 작동하지 않아 fout 방식으로 텍스트 파일로 출력하였을 때 cout과 같은 결과를 내지 못함. fout 에서 문자열 한 칸 지우는 연산을 찾을 수 없었다. 사용할 수 있는 다른 방식이 존재할 것 같은데, 구현하지 못함.

흑백 판단 구현에서 평균 그레이스케일 값 사용하여 연산하는 것을 활용하지 못함. 숫자의 영향을 적게 받는 Point clean_gray 를 사용하는 것이 img3, 4, 10에 대해서는 문제가 없지만, 또 다른 많은 경우에서는 매우 정확히 동작하지 않을 가능성도 있다.

# 질문

감사합니다