9nd Week Lab Assignment

12201928 이상혁

Task 1

coin.png 의 동전 개수를 알아내는 프로그램을 구현

이를 구현하기 위해 실습시간에서 사용했던 BlobDetection 을 사용하였다. Blob 이라는 이미지 내에서 주변보다 더 밝거나 어두운 영역을 검출하여 흰 배경과 동전으로 이루어진 사진 파일에서 개수를 검출하였다.

실습시간에 진행했던 파라미터 대로 진행을 하니 특정 몇 개의 동전만 검출됨을 확인할 수 있었다.

다양한 블롭을 검출하기 위해 원래 설정된 mincircularity 의 값을 조금 더 낮춰주었다.

이렇게 했더니 아까보다는 많은 동전이 검출되는 동시에 작은 블롭까지 검출되었다.

이후 minArea 를 10 -> 100 지정하여서 작은 블롭들을 제거하였다.

500 원을 제외한 모든 동전이 인식되었다.

이후 maxArea 를 10000 정도로 설정해주니 모두다 잘 blob 이 검출됨을 확인할 수 있었다.

이후 putTEXT 를 사용하여 KEPOINTS 의 SIZE 의 값을 이미지에 넣어주어 동전의 개수를 확인할 수 있도록 설정하였다.


```
SimpleBlobDetector::Params params;
params.minThreshold = 10;
params.maxThreshold = 500;
params.filterByArea = true;
params.minArea = 100;
params.maxArea = 10000;
params.filterByCircularity = true;
params.minCircularity = 0.3;
params.filterByConvexity = true;
params.minConvexity = 0.9;
params.filterByInertia = true;
params.minInertiaRatio = 0.01;
// Set blob detector
Ptr<SimpleBlobDetector> detector = SimpleBlobDetector::create(params);

// Detect blobs
std::vector<KeyPoint> keypoints;
detector->detect(img, keypoints);

// Draw blobs
Mat result;
drawKeypoints(img, keypoints, result,
    Scalar(255, 0, 255), DrawMatchesFlags::DRAW_RICH_KEYPOINTS);

int Coin = keypoints.size();


putText(result, "COIN: " + to_string(Coin), Point(10, 30), FONT_HERSHEY_SIMPLEX, 1, Scalar(0, 0, 0), 2);
imshow("coin", result);
waitKey(0);
destroyAllWindows();
```

Task 2

OpenCV 의 1. corner detection 과 2. circle detection 을 이용해 삼각형, 사각형, 오각형, 육각형의 영상을 순차적으로 읽어와 각각 몇 각형인지 알아내는 프로그램을 구현(도형 4 개는 그림판, PPT 등을 이용해 각자 생성할 것)

이번에는 coner detection 인 harris 함수와 blobdetection 을 동시에 이용하였다. 먼저 garriscorner 를 통하여 코너의 개수를 센 이후에 코너에 그려진 동그라미의 개수를 인식하여 사진 및 콘솔에 출력되도록 작성하였다. 작성된 코드는 아래와 같다.

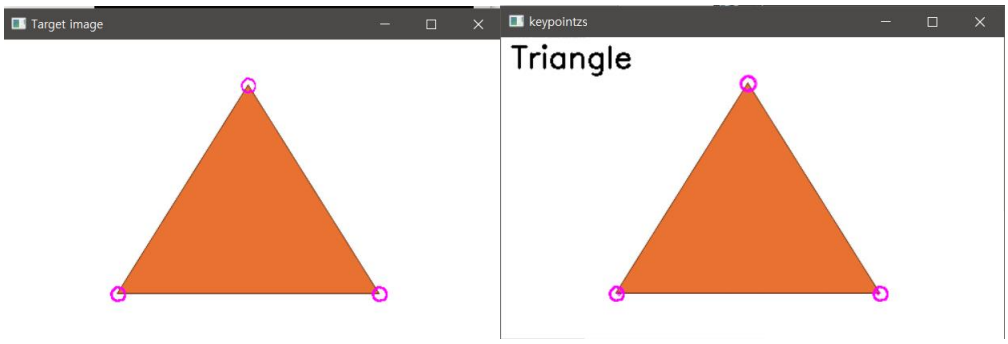
```
Mat cvHarrisCorner(Mat img) {
    resize(img, img, Size(500, 300), 0, 0, INTER_CUBIC);

    Mat gray;
    cvtColor(img, gray, COLOR_BGR2GRAY);
    Mat harr;
    cornerHarris(gray, harr, 2, 3, 0.05, BORDER_DEFAULT);
    normalize(harr, harr, 0, 255, NORM_MINMAX, CV_32FC1, Mat());
    Mat harr_abs;
    convertScaleAbs(harr, harr_abs);

    int thresh = 125;
    Mat result = img.clone();
    for (int y = 0; y < harr.rows; y += 1) {
        for (int x = 0; x < harr.cols; x += 1) {
            if ((int)harr.at<float>(y, x) > thresh)
                circle(result, Point(x, y), 7, Scalar(255, 0, 255), 0, 4, 0);
        }
    }

    imshow("Source image", img);
    imshow("Harris image", harr_abs);
    imshow("Target image", result);
    waitKey(0);
    destroyAllWindows();
    return result;
}
```

```
}
void tesk2cvBlobDetection(Mat img) {
    // Set params
    SimpleBlobDetector::Params params;
    params.minThreshold = 10;
    params.maxThreshold = 400;
    params.filterByArea = true;
    params.minArea = 10;
    params.maxArea = 1000;
    params.filterByCircularity = true;
    params.minCircularity = 0.5;
    params.filterByConvexity = true;
    params.minConvexity = 0.8;
    params.filterByInertia = true;
    params.minInertiaRatio = 0.01;

    // Set blob detector
    Ptr<SimpleBlobDetector> detector = SimpleBlobDetector::create(params);

    // Detect blobs
    std::vector<KeyPoint> keypoints;
    detector->detect(img, keypoints);

    // Draw blobs
    Mat result;
    drawKeypoints(img, keypoints, result,
        Scalar(255, 0, 255), DrawMatchesFlags::DRAW_RICH_KEYPOINTS);

    int Coin = keypoints.size();
    string ans;
    if (Coin == 3) ans = "Triangle";
    if (Coin == 4) ans = "Rectangle";
    if (Coin == 5) ans = "Pentagon";
    if (Coin == 6) ans = "Hexagon";

    putText(result, ans , Point(10, 30), FONT_HERSHEY_SIMPLEX, 1, Scalar(0, 0, 0), 2);
    cout << "이미지 정답 : " << Coin << "각형" << endl;
    imshow("keypointzs", result);
    waitKey(0);
    destroyAllWindows();
}
```

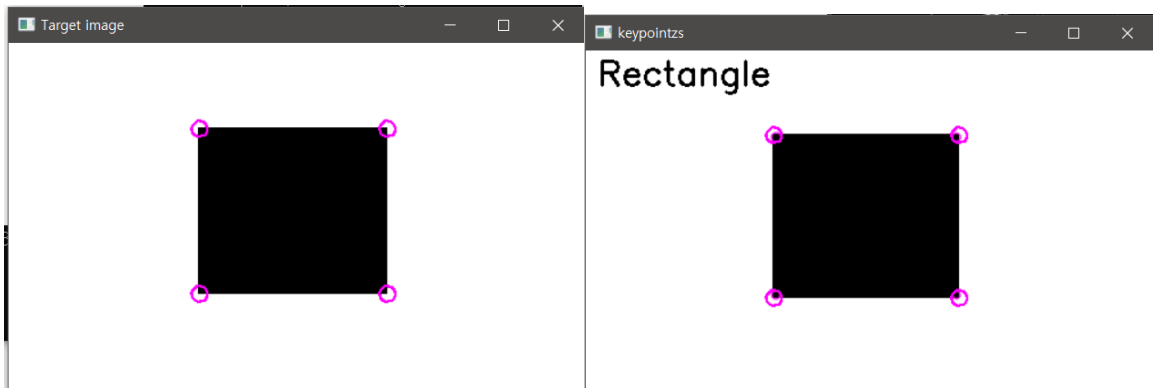
이후 ppt를 통하여 제작한 각 도형을 불러와서 실행시켰다.

```
//2
Mat img = imread("9/1.jpg", IMREAD_COLOR);
Mat dst = cvHarrisCorner(img);
tesk2cvBlobDetection(dst);

Mat img1 = imread("9/2.jpg", IMREAD_COLOR);
Mat dst1 = cvHarrisCorner(img1);
tesk2cvBlobDetection(dst1);

Mat img2 = imread("9/3.jpg", IMREAD_COLOR);
Mat dst2 = cvHarrisCorner(img2);
tesk2cvBlobDetection(dst2);

Mat img3 = imread("9/4.jpg", IMREAD_COLOR);
Mat dst3 = cvHarrisCorner(img3);
tesk2cvBlobDetection(dst3);
```

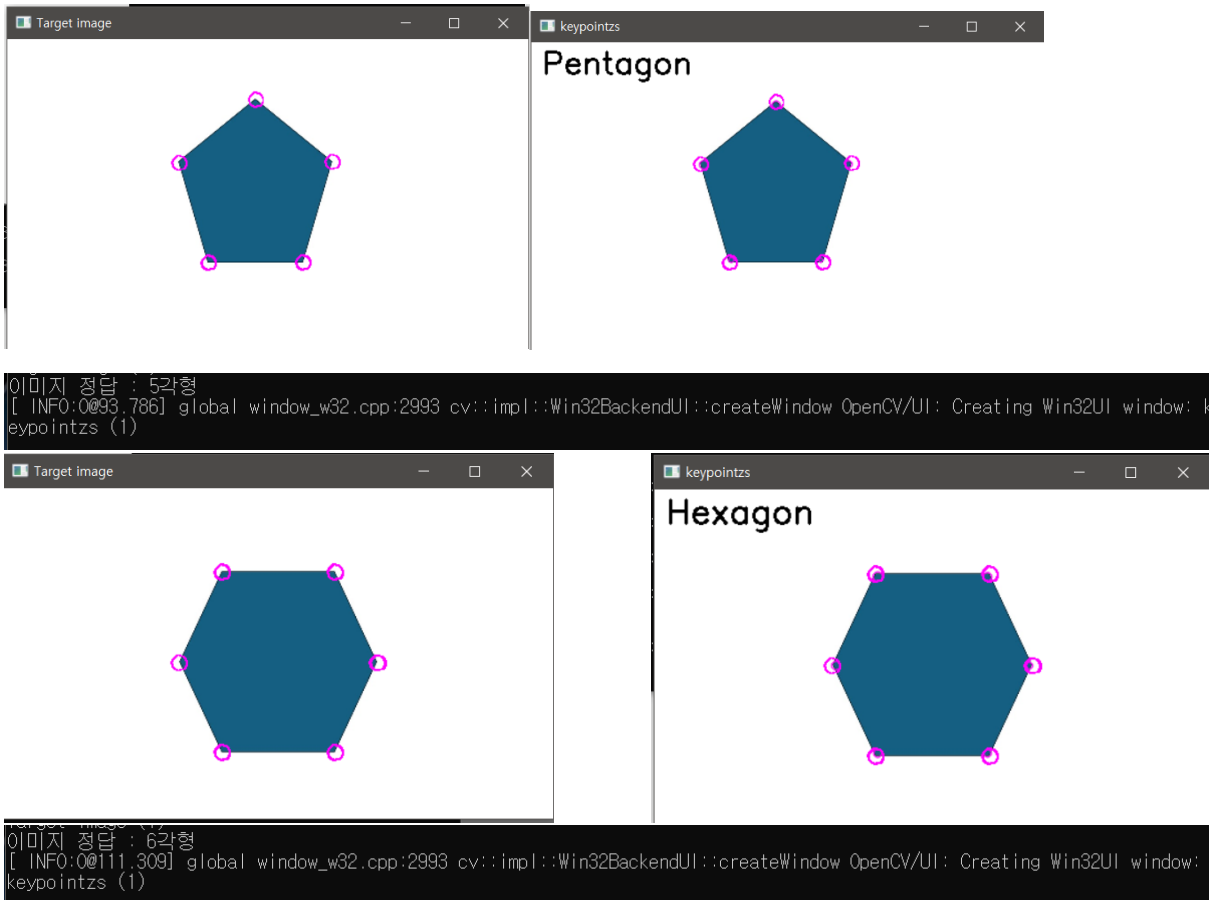
결과는 아래와 동일하다.




```
이미지 정답 : 3각형
[ INFO:0@10.195] global window_w32.cpp:2993 cv::impl::Win32BackendUI::createWindow OpenCV/UI: Creating Win32UI window:
eypointzs (1)
```




```
이미지 정답 : 4각형
[ INFO:0@61.801] global window_w32.cpp:2993 cv::impl::Win32BackendUI::createWindow OpenCV/UI: Creating Win32UI window:
eypointzs (1)
```



각 도형에 따라서 정상적으로 검출됨을 확인하였다. 이미지를 RESIZE 하는 과정에서 픽셀이 깨져 detect 하는데 가끔씩 오류가 있었으나 정상적으로 출력이 되는 지점을 찾아 이를 적용하였다.

Task 3

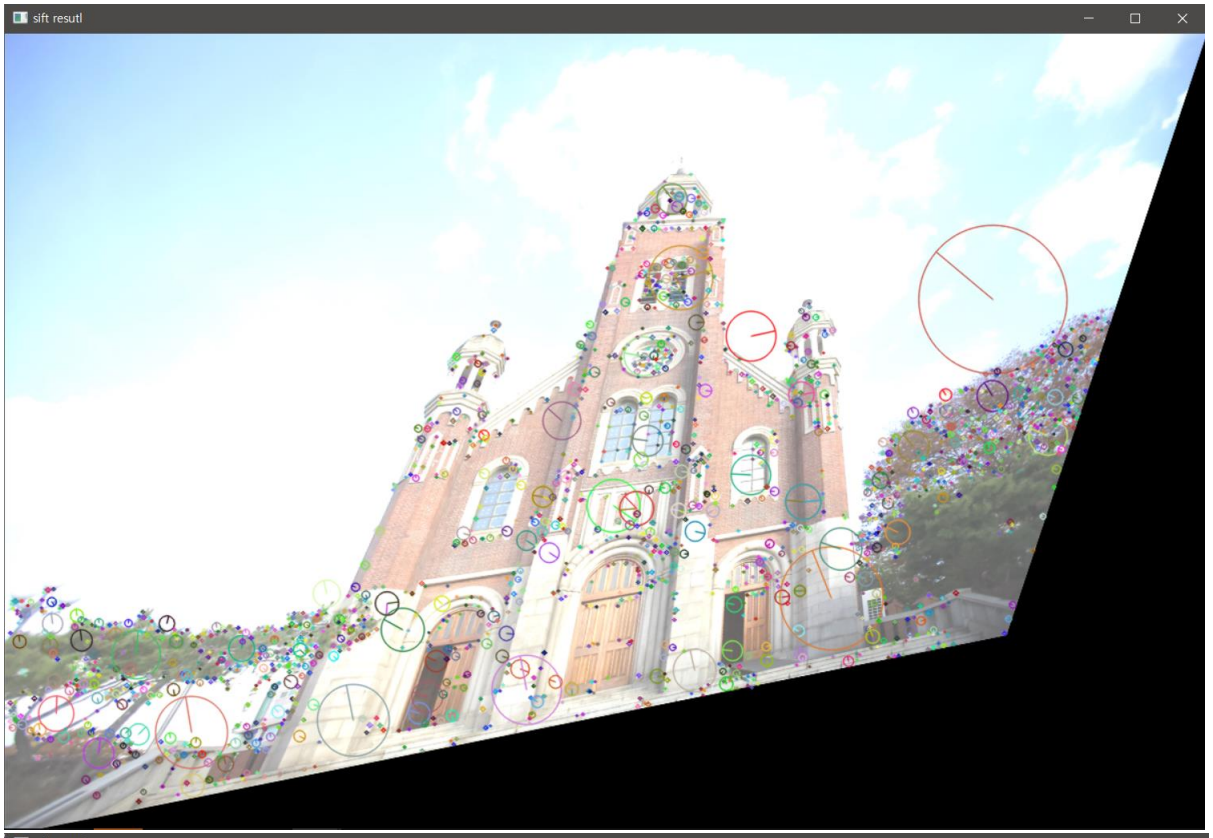
church.jpg 에 투시변환(perspective change)과 밝기 변화를 같이 수행한 후

SIFT 특징점을 구했을 때 원본 영상의 SIFT 특징점이 보존되는지 확인해

볼 것(warpPerspective()함수 사용)

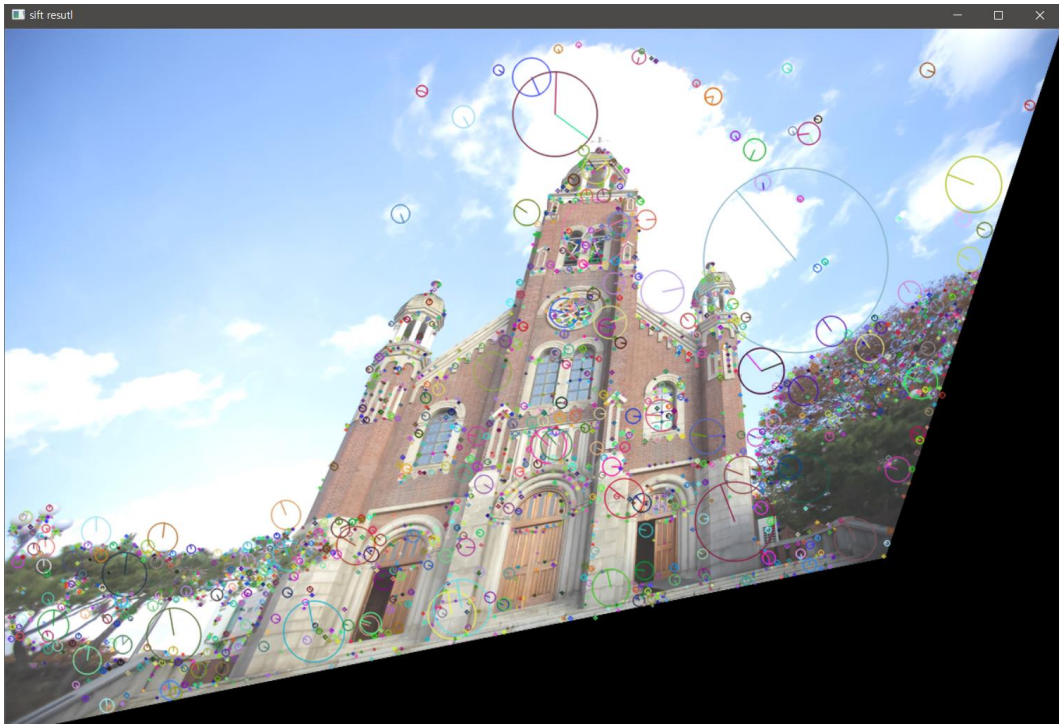
```
    //3
    Mat img3 = imread("9/church.jpg", IMREAD_COLOR);
    Mat dst;
    img3.convertTo(dst, -1, 1, 100);

    imshow("briteness", dst);
    Mat result = warpPers(dst);
    imshow("warp", result);
    cvFeatureSIFT(result);
    cvFeatureSIFT(img3);
}

Mat warpPers(Mat img) {
    Mat dst;
    Point2f src_p[4], dst_p[4];

    src_p[0] = Point2f(0, 0);
    src_p[1] = Point2f(1200, 0);
    src_p[2] = Point2f(0, 800);
    src_p[3] = Point2f(1200, 800);

    dst_p[0] = Point2f(0, 0);
    dst_p[1] = Point2f(1200, 0);
    dst_p[2] = Point2f(0, 800);
    dst_p[3] = Point2f(1000, 600);

    Mat pers_mat = getPerspectiveTransform(src_p, dst_p);
    warpPerspective(img, dst, pers_mat, Size(1200, 800));
    return dst;
}
void cvFeatureSIFT(Mat img) {
    Mat gray;
    cvtColor(img, gray, COLOR_BGR2GRAY);
    Ptr<SiftFeatureDetector>detector = SiftFeatureDetector::create();
    vector<KeyPoint> keypoints;
    detector->detect(gray, keypoints);

    Mat result;
    drawKeypoints(img, keypoints, result, Scalar::all(-1), DrawMatchesFlags::DRAW_RICH_KEYPOINTS);

    imwrite("sift_result.jpg", result);
    imshow("sift resutl", result);
    waitKey(0);
    destroyAllWindows();
}
```

우선 밝기조절을 위하여 converTo 를 작성하여 100 정도로 밝기를 올려주었다. 이후에 이 이미지를 warping 한 뒤 sift 를 진행시켰다. 둘을 동시에 진행한 결과와 원본의 결과를 비교하자면 아래와 같다.



우선 밝기변화를 진행한 이후에 구름 쪽에서의 특징점이 거의 사라지게되었다. 그 이유는 밝기가 올라감에 의하여 구름과 하늘의 색이 거의 동일해여 보존이 안되었기 때문이다.

밝기 변화를 50 정도로 진행한 결과는 아래와 같다.



이처럼 어느정도의 밝기 변화까지에는 sift 알고리즘이 보존이되나, 변화가 극단적으로 이루어질때에는 특징점이 검출되지 않는다고 결과를 도출해 낼 수 있었다.

또한 투시변환을 진행한 이후의 결과에서 일부분의 특징점은 보존이 되었으나, 위치가 약간 변형이 되거나 없었던 특징점들이 생겨남을 확인할 수 있었다.

결론은 밝기나 투시변환이 극단적, 강하게 일어날수록 원본이미지에서의 보존확률이 줄어든다.