

※ 코드 실행을 위해서 OpenCV를 설치하고 모든 image, cpp파일을 프로젝트 경로에 넣어야 함.

0. 프로그램 소개

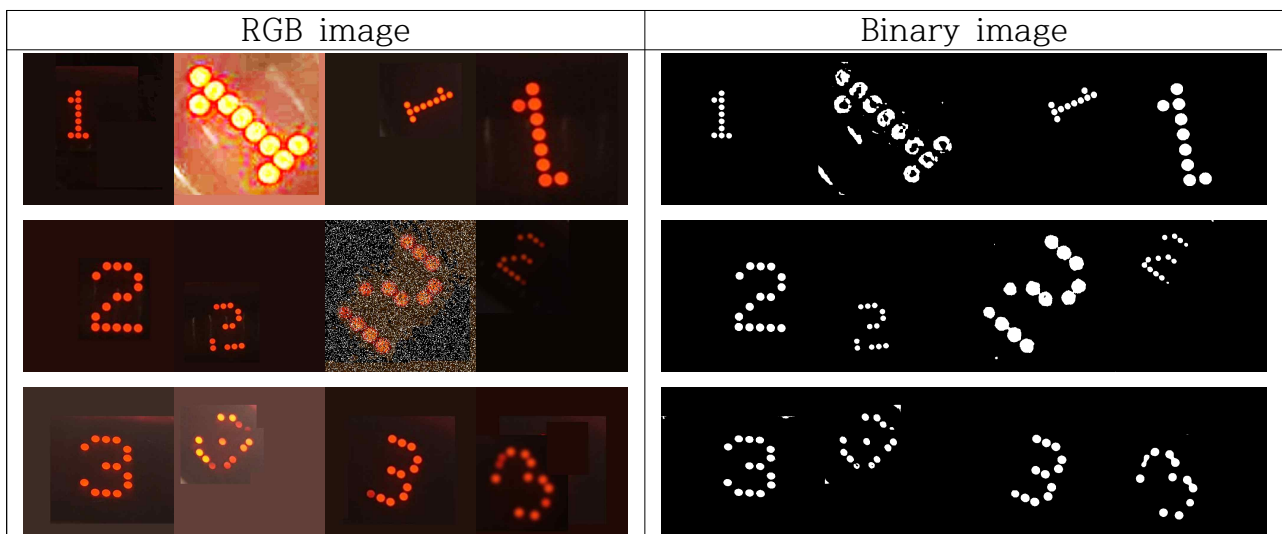
저는 엘리베이터 숫자 인식하는 프로그램을 만들기 위해 가장 직관적인 Template Matching 방법을 사용하였습니다. MDC는 특징을 단정하기 어려운 주제라고 생각했고, Fourier Descriptor는 구현하기가 복잡하다고 생각했습니다.

작업 순서는 Binarization -> Template Matching으로 진행하였습니다. Binarization을 통해 이미지의 intensity를 단순화하고, 1, 2, 3 패턴의 템플릿을 대조해가며 유사성을 찾는 방식입니다. Binarization은 Dev-c++에서 코딩하였고, Template Matching은 Visual Studio 2019에서 OpenCV 라이브러리를 사용하였습니다.

1. Binarization (binarization.cpp)

우선 이미지를 단순화시키기 위해서 Binarization을 진행하였습니다. 기존의 이미지를 ImageJ를 통해 512x512 24bit bmp이미지로 변환하였습니다. 256x256 대신 512x512를 사용한 이유는 더 정밀한 매칭을 위해서, 그리고 이전 과제 코드의 재활용을 위해서입니다.

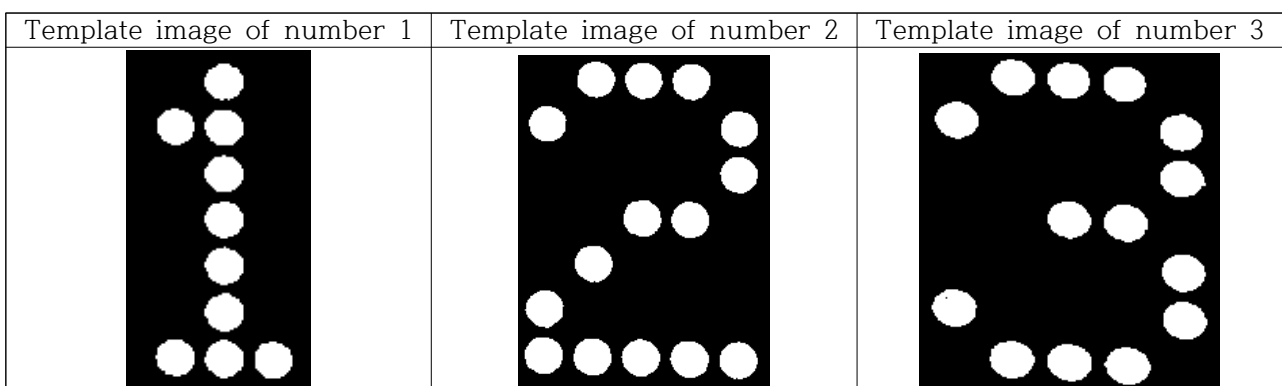
이미지는 RGB -> Grayscale -> Binary로 한 코드 내에서 변환한 뒤, 별도의 binary 이미지 파일을 생성하였습니다. 2-c image의 경우 salt-pepper noise가 있어서 median 필터를 적용시킨 뒤 binarization 하였습니다.



=> Binarization 과정이 이번 과제의 Main topic은 아니므로, 결과 이미지만 첨부하도록 하겠습니다.

2. Template Matching (main.cpp)

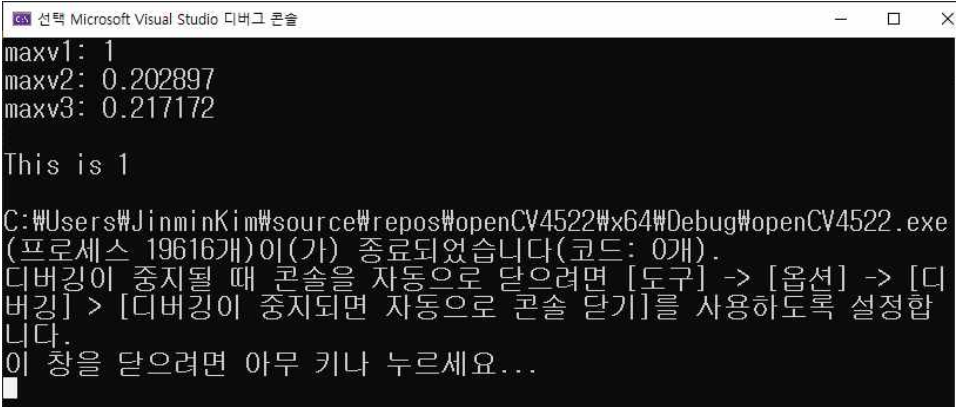
템플릿 매칭을 위해서 템플릿 이미지가 필요했습니다. 템플릿 이미지는 그 숫자를 대표할 수 있어야 하므로, 빈 요소가 없고 왜곡이 없는 이미지인 1-a, 2-a, 3-a를 각각 숫자 1,2,3의 템플릿 이미지로 사용하였습니다. 숫자 모양이 있는 명확한 템플릿 이미지를 위해 ImageJ에서 cutting 하였습니다.

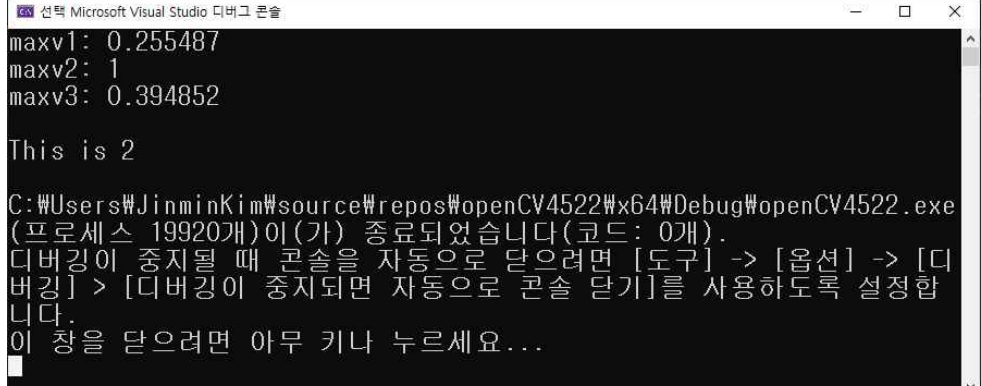
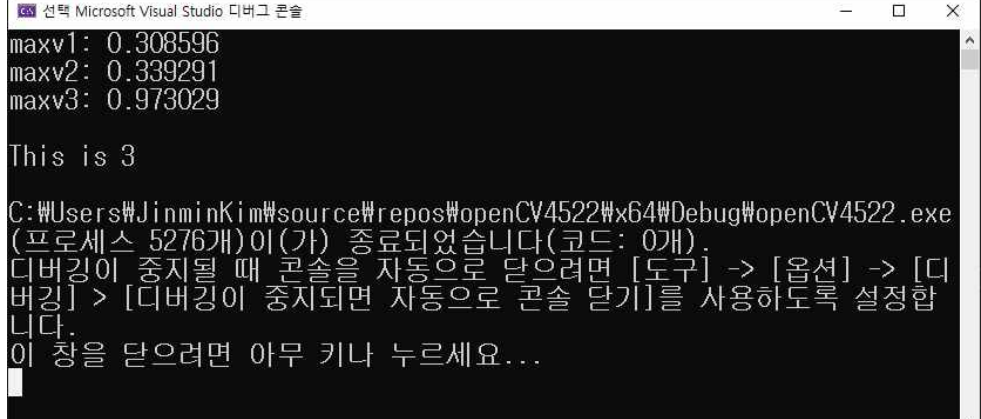


- 코드 분석

코드	해석
<pre>Mat img = imread("3-a_binary.bmp", IMREAD_COLOR); Mat templ1 = imread("1-a_binary-1.bmp", IMREAD_COLOR); Mat templ2 = imread("2-a_binary-1.bmp", IMREAD_COLOR); Mat templ3 = imread("3-a_binary-1.bmp", IMREAD_COLOR);</pre>	<p>img : 분석대상 이미지 templ1 ~ templ3 : 템플릿 이미지</p>
<pre>if (img.empty() templ1.empty() templ2.empty() templ3.empty()) { cerr << "Image load failed!" << endl; return; }</pre>	<p>위 4개의 이미지 중 하나라도 없으면 에러 출력</p>
<pre>// img가 templ1과 얼마나 겹치는지 확인 matchTemplate(img, templ1, res, TM_CCOEFF_NORMED); normalize(res, res_norm, 0, 255, NORM_MINMAX, CV_8U); double maxv1; Point maxloc; minMaxLoc(res, 0, &maxv1, 0, &maxloc); cout << "maxv1: " << maxv1 << endl;</pre>	<p>img가 숫자1과 얼마나 겹치는지 => maxv1 : 1과 겹치는 정도 (0~1값)</p>
<pre>// img가 templ2와 얼마나 겹치는지 확인 matchTemplate(img, templ2, res, TM_CCOEFF_NORMED); normalize(res, res_norm, 0, 255, NORM_MINMAX, CV_8U); double maxv2; minMaxLoc(res, 0, &maxv2, 0, &maxloc); cout << "maxv2: " << maxv2 << endl;</pre>	<p>img가 숫자2와 얼마나 겹치는지 => maxv2 : 2와 겹치는 정도 (0~1값)</p>
<pre>// img가 templ3과 얼마나 겹치는지 확인 matchTemplate(img, templ3, res, TM_CCOEFF_NORMED); normalize(res, res_norm, 0, 255, NORM_MINMAX, CV_8U); double maxv3; minMaxLoc(res, 0, &maxv3, 0, &maxloc); cout << "maxv3: " << maxv3 << endl;</pre>	<p>img가 숫자3과 얼마나 겹치는지 => maxv3 : 3과 겹치는 정도 (0~1값)</p>
<pre>if (maxv1 >= 0.9) cout << endl << "This is 1" << endl; if (maxv2 >= 0.9) cout << endl << "This is 2" << endl; if (maxv3 >= 0.9) cout << endl << "This is 3" << endl;</pre>	<p>img를 templ1~templ3과 비교 후 maxv가 0.9보다 크면 (90%이상 일치하면) img가 그 숫자라는 것을 알려주도록 함.</p>

- 결과

코드	해석
<pre>Mat img = imread("1-a_binary.bmp", IMREAD_COLOR); Mat templ1 = imread("1-a_binary-1.bmp", IMREAD_COLOR); Mat templ2 = imread("2-a_binary-1.bmp", IMREAD_COLOR); Mat templ3 = imread("3-a_binary-1.bmp", IMREAD_COLOR);</pre>	<p>1-a를 넣었을 때</p>
 <pre>선택 Microsoft Visual Studio 디버그 콘솔 maxv1: 1 maxv2: 0.202897 maxv3: 0.217172 This is 1 C:\Users\JinminKim\source\repos\openCV4522\wx64\Debug\openCV4522.exe (프로세스 19616개)이(가) 종료되었습니다(코드: 0개). 디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다. 이 창을 닫으려면 아무 키나 누르세요...</pre>	<p>maxv1 > 0.9이므로 "This is 1" 출력 (숫자 1로 인식됨)</p>

<pre>Mat img = imread("2-a_binary.bmp", IMREAD_COLOR); Mat templ1 = imread("1-a_binary-1.bmp", IMREAD_COLOR); Mat templ2 = imread("2-a_binary-1.bmp", IMREAD_COLOR); Mat templ3 = imread("3-a_binary-1.bmp", IMREAD_COLOR);</pre>	2-a를 넣었을 때
	maxv2 > 0.9이므로 “This is 2” 출력 (숫자 2로 인식됨)
<pre>Mat img = imread("3-a_binary.bmp", IMREAD_COLOR); Mat templ1 = imread("1-a_binary-1.bmp", IMREAD_COLOR); Mat templ2 = imread("2-a_binary-1.bmp", IMREAD_COLOR); Mat templ3 = imread("3-a_binary-1.bmp", IMREAD_COLOR);</pre>	2-a를 넣었을 때
	maxv3 > 0.9이므로 “This is 3” 출력 (숫자 3으로 인식됨)

- 고찰 및 한계점

=> 1-a, 2-a, 3-a image는 왜곡과 노이즈가 없는 깔끔한 이미지이므로 잘 구별되었습니다. 하지만 다른 악조건을 가진 이미지로 테스트를 해본다면 성능이 나오지 않을 것입니다.

사실 아래와 같은 아이디어로 악조건 속에서도 숫자 검출을 구현하기 위한 코드 공간을 main.cpp에 만들어두었습니다.

코드	해석
<pre>/* templ을 scaling & rotate시키는 코드 double scale = 0.5; while (scale < 1.5) { // templ을 scaling 시키는 코드 // templ을 rotate시키는 코드 for (int angle = 0; angle < 360; angle += 10) { } scale += 0.1 } */</pre>	<p>scale : 0.5 -> 1.5로 0.1씩 커지는 동안 매번 angle : 0 -> 360로 10씩 커지도록 template을 조정하여, maxv를 조건마다 전부 계산하는 방식</p>

그러나 OpenCV에 대한 코딩 능력과 시간이 부족하여 구현에는 실패하였습니다. 앞으로 OpenCV를 더 공부하여 구현해보도록 하겠습니다.