[설명서]

1. 대상 선정 개요

이 프로그램은 육각형(hexagon)/직사각형(rectangle)/원(circle) 영상 이미지를 학습시키고, 입력 영상에 대해 위 3개의 도형 중 어떤 도형인지 판단하는 프로그램이다. 육각형, 직사각형, 원을 학습 대상 및 인식 대상으로 선정한 이유는 비교적 인식이 간단한 형태이기 때문이다. 다른 3차원 형태의 모양과 달리 육각형, 직사각형, 원과 같은 2차원 형태의 도형 영상은 명암 영상으로 만든 뒤, 이진화시킨 후 morphological 연산을 수행한 뒤, 경계 화소를 표시하여 특징을 추출하는 과정이 3차원 형태의 물체 영상보다 더 계산량이 적고 간결한 특징이 있다.

또한, 육각형, 직사각형, 원과 같은 도형은 contour-tracing을 한 후 표시되는 경계 화소 영상이 뚜렷한 모양과 명확한 특징을 지니고 있어, 이를 통해 특징 추출을 하고 매칭을 할 때 인식률이 올라갈 수 있는 특징이 있어서 육각형, 직사각형, 원 영상을 인식 및 학습 대상으로 선정하였다.

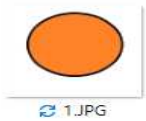
2. 학습과 인식 자료 구분

각 도형 별로 폴더를 생성하여 그 안에 11개의 영상을 넣는다. 이 중에서 1개의 영상은 학습 자료로 선정, 나머지 10개의 영상은 인식 자료로 선정한다.

-육각형 학습 자료: [hex/1.jpg]

-육각형 인식 자료: [hex/2.jpg, 3.jpg, 4.jpg, 5.jpg, 6.jpg, 7.jpg, 8.jpg, 9.jpg, 10.jpg, 11.jpg]

-원 학습 자료: [circle/1.jpg]



-원 인식 자료: [circle/2.jpg, 3.jpg, 4.jpg, 5.jpg, 6.jpg, 7.jpg, 8.jpg, 9.jpg, 10.jpg, 11.jpg]

-직사각형 학습 자료: [rect/1.jpg]

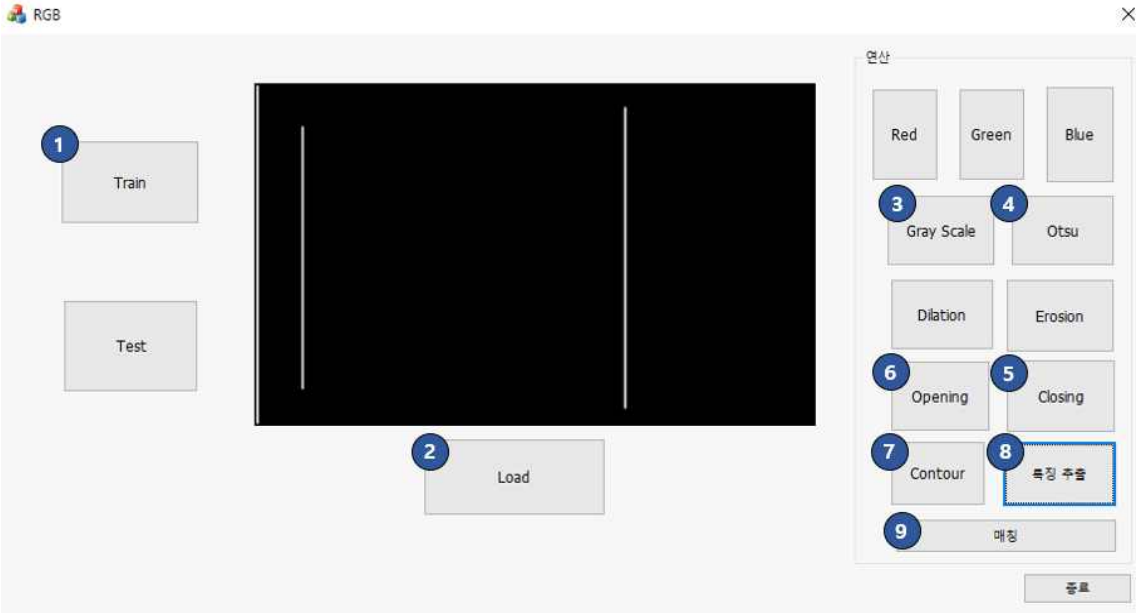
-직사각형 인식 자료: [rect/2.jpg, 3.jpg, 4.jpg, 5.jpg, 6.jpg, 7.jpg, 8.jpg, 9.jpg, 10.jpg, 11.jpg]

3. 사용법

사용법은 총 2가지 방법이 있다. 첫 번째 방법은, Train 버튼을 눌러 학습 영상을 이용해 학습한 뒤 Load 버튼을 눌러 불러올 인식 영상을 불러온다. 그 후 Test 버튼을 눌러 로딩된 인식 영상이 어느 도형에 속하는지 확인 한다.
이때 Test 버튼 안에서는 로딩된 인식 영상을 Gray Scale 영상(명암 영상) 으로 변환한 뒤, Otsu로 이진화를 수행한 뒤, morphological 연산(closing과 opening 연산)을 수행한 뒤, contour-tracing algorithm을 이용해 경계 화소를 출력한 뒤, 이를 바탕으로 특징 추출을 한 뒤 매칭을 하는 과정이 다 포함되어 있다.

두 번째 방법은 Train 버튼을 눌러 학습 영상을 이용해 학습한 뒤 Load 버튼을 눌러 불러올 인식 영상을 불러온 후 우측의 “연산” 코너의 버튼을 순차적으로 눌러 연산을

수행한 뒤 매칭하는 방식이다. “연산” 코너의 버튼은 다음과 같은 순서로 누른다. Red, Green, Blue 버튼(선택 사항)->Gray Scale 버튼->Otsu 버튼->Closing 버튼->Opening 버튼(morphological 연산 버튼을 누르는 순서는 상관 없음)->Contour 버튼->특징 추출 버튼->매칭 버튼 과 같은 순서로 누르면 해당 입력 영상이 어떤 도형에 속하는지 메시지 박스로 출력한다.



4. 기능 설명

(1) 학습 과정

-void CRGBDlg::OnBnClickedTraining1(): train 버튼을 누르면 OnBnClickedTraining1 함수가 실행된다. 이 함수에서 train 함수를 3번 호출해 각각 육각형, 직사각형, 원 학습 영상에 대한 LCS를 얻는다.

-void train(int fig_num, CArray<CPoint, CPoint>& c, CArray<double, double&>& lcs): fig_num이 1이면 육각형 학습 영상을, 2이면 직사각형 학습 영상을, 3이면 원 학습 영상을 폴더에서 읽어온다. 읽어온 영상을 grayscale 함수를 호출하여 명암 영상으로 바꾼 후, segmentation 함수를 호출하여 명암 영상을 이진화 하고, morp 함수를 호출하여 이진화한 영상에 closing, opening 연산을 수행하고, contourTracing 함수를 호출하여 morphological 연산을 수행한 영상의 경계 화소의 위치를 저장한다. 그 후, lcs_extract 함수를 호출하여 해당 학습 영상의 LCS를 추출한다.

-Mat grayscale(Mat img): 입력 영상을 명암 영상으로 바꾸는 기능을 한다. GrayScale 영상을 만들기 위해 1개 채널로 구성된 single-channel matrix를 생성한다. 새로 생성한 gray_img는 img와 크기가 유사하며 CV_8UC1로 1개 채널로 구성되고 픽셀 하나당 0부터 255까지의 값을 갖는 영상이다. img의 모든 픽셀에 대하여, 각 RGB 채널에 대한 픽셀값을 픽셀 접근 함수 img.at<Vec3b>(y,x)를

이용해 얻고, 이렇게 얻은 3개의 픽셀 값을 모두 더해 3으로 나누어 gray_img의 동일 픽셀 위치에 그 픽셀 값을 저장해 준다. 그 후, 해당 gray_img 영상에 clone 함수를 호출하여 리턴한다.

-void segmentation(Mat img): 입력된 명암 영상을 이진화하는 기능을 수행한다. Otsu 알고리즘은 강의 교재의 p.71 알고리즘2-4를 참고하였다. 히스토그램을 나타내는 배열 H, 정규화 히스토그램을 나타내는 배열 NH를 선언한다. w0, m0, m1, v_between 배열을 선언한다. 명암 영상의 모든 픽셀에 접근하여 명암값에 해당하는 히스토그램(H)의 인덱스에 있는 값을 1만큼 증가시킨다. 정규화 히스토그램(NH)은 H의 각 인덱스에 있는 값을 img의 크기만큼 나누어 NH의 해당 인덱스에 저장한다. 전체 평균을 나타내는 m값은 각 명암값과 NH 배열의 요소 값들을 곱한 값을 더해서 구한다.

w0의 0번째 값에 NH의 0번째 값을 대입하고 m0의 0번째 값에는 0을 대입한다. 이후 1부터 255까지 반복문을 실행한다. 반복문의 내용은 다음과 같다.

w0의 t번째 값에 w0의 t-1번째 값과 NH의 t번째 값을 더한다. 이후 w0의 t번째 값이 0이면, m0의 t번째 값에 0을 대입하고 그렇지 않으면 m0의 t번째 값에 w0의 t-1번째 값, m0의 t-1번째 값, NH의 t번째 값에 t를 곱한 값을 곱하고 이를 w0의 t번째 값으로 나눈다. m1의 t번째 값은 전체평균 m에서 w0의 t번째 값과 m0의 t번째 값을 곱한 것을 빼고, 이 값을 1-w0[t]로 나눈다. v_between[t] 값은 w0[t]와 (1-w0[t])를 곱하고 그 값에 (m0[t]-m1[t]) 값을 거듭제곱한 값을 곱한다. 이때 최대의 v_between 배열 요소의 값을 저장하는 max_value가 현재 v_between[t] 보다 작으면, t를 임계값 T로 취하고 max_value를 해당 v_between[t] 값으로 바꾸어 준다.

반복문의 과정을 거치고 앞의 루프에서 가장 큰 v_between[t] 값을 보인 t를 임계값 T로 설정한다. img의 모든 픽셀에 대해, 이 임계값 T보다 크거나 같은 값들은 모두 255(=1) 로 처리한다. T보다 작은 값들은 0으로 처리하여 영상을 이진화한다.

-void morp(Mat& img): 이진화된 영상에 closing 연산 및 opening 연산을 적용한다. 이진화된 영상에 closing 연산을 적용한 영상을 result 변수에 넣은 뒤, result 영상에 opening 연산을 적용한 영상을 다시 img 변수에 넣어 morphological 연산을 수행한다.

-Mat closing(Mat img): closing 연산은 dilation 후 erosion을 수행하는 연산이다. dilation 함수를 호출하고 호출이 끝난 후 리턴된 영상을 결과 영상을 저장하는 result 변수에 저장한다. 그 후 erosion 함수에 result를 매개변수로 주어 호출한 뒤 리턴된 값을 반환하면 closing 연산을 수행한 영상을 리턴하게 된다.

-Mat opening(Mat img): opening 연산은 erosion 후 dilation을 수행하는 연산이다. closing 과 같은 매커니즘으로 수행한다. 단 다른 점은 erosion 함수를 호출한 뒤에 dilation 함수를 호출한다는 점이다. 이 함수에서는 opening 연산을 수행한 영상을 리턴하게 된다.

dilation, erosion을 구현 시 필요한 structuring element에 대해 다음과 같이 구성하였다. structuring element를 5x5 크기이며 모두 255로 이루어진 배열로 설정하였다. 전역 변수로 structuring_element를 정의하고, structuring_element 의 요소와 영상의 픽셀을 비교하기 위해 좌표를 움직여야 하므로 y축, x축의 움직임을 나타내는 dy, dx 배열도 정의한다.

-Mat dilation(Mat img): dilate 된 영상을 저장하는 Mat 형의 result를 선언한다. result는 img와 크기가 같고 1개의 채널로 구성된 0부터 255까지의 픽셀값을 갖는다. img의 모든 화소에 대해서 다음과 같은 루프를 실행한다.

img의 한 화소와 그 주변 화소들과 structuring element를 비교하는 작업을 한다. Structuring_element의 크기가 5x5 이므로, 한 픽셀과 그 주변의 24개 픽셀을 structuring_element와 비교한다. 비교할 때 기준이 되는 픽셀의 위치에서 dy, dx 배열의 값들을 빼서 주변 픽셀의 위치를 가져와 비교를 수행하게 된다. 비교 시에는 화소 접근 함수 at을 사용해 structuring_element 와 비교한다. img의 한 픽셀이라도 structuring_element와 동일한 값, 즉 255라면 flag 변수를 1로 바꾸고, 비교하는 반복문을 탈출한다. 비교하는 반복문이 끝난 후 flag 가 1이면 structuring_element에서 255인 성분을 가지는 위치에 영상 화소가 한 곳이라도 255값을 가졌다는 의미이므로 결과 영상인 result의 해당 픽셀을 255로 설정한다. flag가 0인 경우는 비교했을 때 하나도 겹치는 것이 없었으므로 해당 픽셀을 0으로 설정한다. 모든 화소에 대해 위와 같은 루프를 수행한 뒤에, 결과 영상인 result를 리턴한다.

-Mat erosion(Mat img): 구현 방법은 위 dillation 함수와 같다. 다른 점은 erosion 연산에서는 structuring_element에서 255 값을 가지는 위치는 입력 영상인 img에서도 모두 255여야 결과 영상의 화소값도 255가 되기 때문에, structuring_element와 기준 화소, 주변 화소를 비교하는 과정에서 한 픽셀이라도 다른 값이 발생하면 flag 변수를 1로 바꾸고 비교하는 반복문을 탈출한다는 것이다.

비교하는 반복문이 끝난 후에 flag 가 1이면 한 곳이라도 structuring_element에서 255값을 가지는 위치가 img에서는 255값을 갖지 못한 것을 의미하기 때문에 결과 영상 result의 해당 픽셀을 0으로 설정한다. flag가 0이면 structuring_element에서 255였던 위치가 img에서 기준, 주변 픽셀에 대해 모두 255를 가졌다는 것을 의미하므로 해당 픽셀을 255로 설정한다. 모든 화소에 대해 위와 같은 루프를 수행한 뒤, 결과 영상인 result를 리턴한다.

라벨링 및 경계 화소 추적을 위해 선언된 전역변수는 다음과 같다.

-Mat labImg: 입력 영상의 라벨링된 영상

-Mat labImg2: 라벨링된 영역의 경계 화소를 표시하는 영상

-int LUT_BLabeling8: 전 위치, 현재 위치, 다음 위치를 이용해

올라가는 방향인지 표시해주는 LookUp Table
-num_region[500]: 각 label을 가진 화소의 수를 저장하는 배열
-CArray<CPoint, CPoint> contour_array: 입력 영상의 경계 화소(contour pixels)의 위치 x,y 값을 저장하는 배열
-CArray<CPoint, CPoint> chex, crect, ccircle: 육각형, 직사각형, 원의 학습 영상의 경계 화소의 위치 x,y 값을 저장하는 배열

-void contourTracing(Mat &m, CArray<CPoint, CPoint>& contours): 입력 영상에 라벨링을 수행하고, 경계 화소 영상을 만들고, 경계 화소의 위치를 저장하는 배열을 저장하는 함수이다. 라벨 번호를 나타내는 labelnumber 변수는 1로 초기화한다. 라벨링된 이미지를 나타내는 labImg는 화소를 모두 0으로 초기화한다. 경계 화소만을 나타내는 이미지 labImg2도 화소를 모두 0으로 초기화한다. Mat class의 zeros 함수를 이용하여 크기를 입력 영상의 크기로 맞춰주고, 화소를 0으로 초기화한다.
그 다음, 1행부터 마지막 행의 바로 이전행, 1열부터 마지막 열의 바로 이전행 까지 2중 for문을 돌면서 contour tracing을 실행한다. cur_p 변수에 입력 영상의 (y,x) (이때, y는 행의 위치, x는 열의 위치를 나타냄) 위치의 화소를 저장한다.
cur_p가 255, 즉 object이며 labImg에서의 (y,x) 위치의 화소값이 0인 경우(즉 라벨링이 아직 이루어지지 않았을 경우) 에 다음 문단의 과정을 실행한다.
현재 (y,x) 의 위치에서 4번 위치방향인 (y,x-1) 위치의 labImg의 화소, 즉 (y,x-1) 위치의 라벨값을 ref_p1에 저장한다. 5번 위치 방향인 (y-1,x-1) 위치의 라벨값을 ref_p2에 저장한다. ref_p1이 1보다 크면, 즉 현재 위치에서 4번 위치방향의 위치가 이미 라벨링된 상태라면 전파조건이다. 따라서 라벨 번호 ref_p1 를 가지는 화소수를 나타내는 num_region[ref_p1]를 1 증가시켜주고, labImg의 (y,x) 픽셀을 ref_p1로 지정해줌으로써 ref_p1 번으로 (y,x) 위치를 라벨링한다. ref_p1이 0이고, ref_p2가 2보다 작거나 크다면, ref_p2가 라벨링되어있고 ref_p1은 라벨링되지 않은 상태이다. 이는 hole 조건이다. 마찬가지로 라벨 번호 ref_p2 를 가지는 화소수를 나타내는 num_region[ref_p2]를 1 증가시켜주고, labImg의 (y,x) 픽셀을 ref_p2로 지정해줌으로써 ref_p2 번으로 (y,x) 위치를 라벨링한다. 또한 경계 화소만 표시하는 이미지인 labImg2 영상의 (y,x) 위치의 화소에 255를 넣어 흰 색으로 경계를 나타낸다. 그리고 경계 화소의 위치를 저장하는 배열인 contours 배열에 해당 픽셀의 위치 x,y 값을 넣어준다. 그 다음 BTracing 함수를 호출하여 (y,x) 위치로부터 라벨 번호 ref_p2를 주면서 역방향(BACKWARD), 시계 방향으로 그 주변 화소를 탐색하여 조건에 맞으면 라벨링을 하게 된다. ref_p1, ref_p2가 모두 0이면 다 라벨링되지 않은 상태이므로 이는 region start 조건이다. labelnumber를 1 증가시켜주고, 그 labelnumber를 가지는 화소수를 나타내는 num_region[labelnumber]를 1 증가시켜주고, labImg의 (y,x) 픽셀을

labelnumber로 지정해줌으로써 labelnumber로 (y,x) 위치를 라벨링한다. 이 역시 경계 화소이므로, labImg2 영상의 (y,x) 위치의 화소에 255를 넣어 경계를 나타내도록 한다. contours 배열에 해당 픽셀 위치 x,y를 넣어준다. 그 다음 BTracing 함수를 호출하여 순방향(FOREWARD), 시계방향으로 주변 화소를 탐색하여 라벨링 하도록 한다. 이 과정을 2중 for문이 끝날 때까지 반복하고 contour tracing이 끝나면 labImg2에 경계 화소만이 표시된 영상이 저장된다.

-void BTracing(int y, int x, int label, int tag, CArray<CPoint, CPoint> &contours, Mat &m): 파라미터로 위치값 (y,x), label number, tag, contours(경계 화소의 위치값을 저장하는 배열), m(입력 영상)를 받는다. tag가 FOREWARD 일 때는 즉 hole이 아니며 1번째로 라벨을 할당받는 경우이다. 이때는 순방향으로 탐색을 진행하며, 탐색 시작 방향과 전 탐색 시작 방향을 나타내는 변수 cur_orient와 pre_orient를 0으로 지정한다.

tag가 BACKWARD 일때는 hole인 경우이므로 역방향으로 움직인다. cur_orient와 pre_orient를 6으로 지정한다.

현재 위치값 (y,x)를 y,x 변수에 저장하고 함수가 호출될 때의 처음 위치로 돌아오는 경우가 loop의 종료조건이 되므로, end_y 와 end_x 변수에 y,x 변수의 값을 저장한다. 이전의 y,x 위치를 저장하는 변수 pre_y, pre_x에 y,x 값을 저장하고 loop를 시작한다.

처음 위치로 돌아올 때까지 loop가 수행된다. read_neighbor8 함수를 호출하여 (y,x) 로부터의 8개의 주변 화소를 읽어 neighbor8 배열에 저장한다.

(8+cur_orient-2)%8 값을 start_o 변수에 저장한다. 현재 방향(cur_orient)-2를 해야 90도 전의 위치가 되기 때문이다. for loop를 돌면서 start_o 방향으로부터 시계방향으로 이동하여 해당 방향의 화소를 neighbor8 배열로부터 읽어들인다. 이때 읽은 화소가 0이 아니면, 즉 object 인 화소일 때 for loop을 탈출하고, 해당 방향이 add_o 변수에 저장된다.

위의 for loop에서 읽은 화소가 모두 0이었을 경우, y,x 변수와 cur_orient 값을 바꾸지 않는다. for loop에서 읽은 화소 중 하나라도 0이 아니었다면, calCoord 함수를 호출하여 해당 방향에 맞게 y,x 변수의 값을 바꿔준다. 즉 y,x의 위치를 이동시킨다. 또한 탐색 시작 방향인 cur_orient 변수의 값을 add_o 변수의 값으로 바꿔준다. 이후, 전역으로 선언된 LUT_BLabeling8 테이블에서 전 위치-현재 위치의 방향인 pre_orient와 현재 위치-다음 위치의 방향인 cur_orient를 참조하여 올라가는 방향인지 확인한다. 올라가는 방향, 즉 LUT_BLabeling8 테이블의 값이 1이면 해당 label을 가진 화소의 수를 저장하는 배열 값인 num_region[label] 값을 1 증가시켜주고, 라벨링된 이미지를 나타내는 labImg 영상의 (pre_y, pre_x) 위치의 화소에 해당 label을 넣어 라벨링되었음을 표시한다. 또한 경계 화소만 표시하는 이미지인 labImg2 영상의 (pre_y, pre_x) 위치의 화소에 255를 넣어 경계를 흰색으로 표시하도록 한다. contours 배열에 경계 화소의 위치 x,y를 저장한다. pre_y,

pre_x 값에 현재 y,x 값을 넣는다. pre_orient 값에 현재 cur_orient 값을 넣고 위의 loop 과정을 y,x 가 end_y, end_x 에 도달할 때까지 반복한다.
-void calCoord(int i, int* y, int* x): 이미지의 (y,x) 의 위치를 지정한 방향 코드에 맞게 바꿔주는 함수이다. 방향 코드는 0,1,2,3,4,5,6,7 8가지이며 각각 현재의 위치에서 좌, 우, 상, 하, 대각선으로 이동한다. 파라미터로 주소값을 전달해 (y,x) 의 위치가 바뀔 수 있도록 한다.
-void read_neighbor8(int y, int x, int *neighbor8, Mat &m):이미지의 (y,x) 위치 화소의 주변 화소 8개를 배열 neighbor8 에 저장한다. 배열 dx, dy 값을 두어 이동할 방향을 기록할 수 있도록 한다. for loop를 돌며 현재 위치 (y,x)에서 배열 dy[i], dx[i] 값을 더해 주변 화소의 위치로 이동한 뒤 주변 화소의 값을 배열에 저장한다. 주변 화소의 위치가 이미지의 영역 밖을 벗어나는 경우 배열에 0을 저장한다.

LCS를 추출하기 위해 선언된 전역 변수는 다음과 같다.
-CArray <double, double&> hex_lcs, rect_lcs, circle_lcs: 육각형, 직사각형, 원의 학습 영상의 LCS를 저장하는 배열
-CArray <double, double&> lcs_array: 입력 영상의 LCS를 저장하는 배열

-void lcs_extract(CArray<CPoint, CPoint>& c, CArray<double, double&>& lcs): 영상의 특징을 추출하기 위한 함수로, 특징 추출에는 LCS(Localized Contour Sequence)가 사용된다. LCS는 경계 화소의 위치를 저장한 배열을 이용해 추출한다. LCS는 앞에서 n번째 화소, 뒤에서 n번째 화소의 직선의 방정식과 현재 화소가 직교하는 거리를 의미한다. 경계 화소의 수만큼 반복문을 수행한다. 이 반복문 안에서 이루어지는 과정은 다음과 같다. 현재 경계 화소의 x,y 위치는 변수 x, y 에 저장한다. windows의 크기는 13으로 설정하였다. 현재 경계 화소 배열의 인덱스의 앞에서 6번째, 뒤에서 6번째인 경계 화소의 위치를 추출한다. 이때 '(배열 전체 크기+인덱스 ±6)%배열 전체 크기' 로 앞뒤 6번째 인덱스 값을 알아낸다. 배열의 그 인덱스 위치에 저장된 경계 화소의 x,y 위치를 알아내어 앞뒤 두 점간 직선의 방정식을 구성하는 a,b,z 값(ax+by+z=0)을 알아낸다. 그 후 현재 화소와 직선의 방정식 간의 거리를 구하기 위해 perpendicular distance 공식((| a*x + b*y + z |) / (sqrt(a*a + b*b)) 을 이용한다. 이후 구한 거리 값을 lcs 배열에 저장한다. 반복문을 다 수행한 후, lcs 배열에는 영상의 LCS가 저장된다.

(2) 인식 과정
void CRGBDlg::OnBnClickedTesting(): Test 버튼을 누르면 수행되는 함수이다. 로딩한 입력 영상에 대해 grayscale, segmentation, morp, contourTracing, lcs_extract 함수들을 순차적으로 수행하여 명암영상으로 바꾸고, 이진화하며, morphological 연산을 적용하고, 경계화소를 추적하며, LCS를 추출한다. 이후 학습

영상 3종류 각각의 LCS에 대해 DTW(Dynamic Time Warping)를 수행하여 가장 작은 dissimilarity를 가진 도형에 대해 입력 영상에 그 도형을 매칭시킨다. result 배열에 각각 3종류 도형에 대한 DTW 함수의 리턴 값(dissimilarity 값)을 저장한 후, 배열 요소의 최소값과 최소값을 가지는 도형의 번호(육각형=1, 직사각형=2, 원=3)를 fig 변수에 저장한다. MessageBox를 이용하여 인식된 이 영상의 도형의 이름을 출력한다.

-double DTW(CArray<double, double&> &test_lcs, CArray<double, double&> &ref_lcs): 입력 LCS(test_lcs)와 학습 LCS(ref_lcs)의 dissimilarity를 계산하는 함수이다. 배열 D는 정보를 저장하는 캐시 공간, 배열 G는 방향 코드를 저장하는 공간이다. 각 배열의 크기는 세로로 입력 LCS의 길이(=n)만큼이며, 가로로 학습 LCS의 길이(=m) 만큼이다. 배열 D의 1번째 행, 1번째 위치에는 입력 LCS와 학습 LCS의 1번째 요소의 거리를 대입한다. 배열 G[1][1]에는 방향코드 0을 대입한다. 배열 D의 1번째 행의 모든 m개의 요소에 대해, 입력 LCS의 1번째 요소와 학습 LCS의 2번째 요소부터 m번째 요소까지의 거리를 대입한다. 배열 G의 1번째 행의 모든 m개의 요소에 대해서는 모두 방향코드 2(현재 위치에서 왼쪽으로 한칸 이동)를 대입한다. 배열 D의 1번째 열에 있는 모든 요소에 대해서는 무한대 값을 대입한다. 이후, 2번째 행부터 n번째 행(i)까지, 2번째 열부터 m번째 열(j)까지 2중 for문을 돌며 배열 D와 배열 G를 채운다. for문에서 D[i][j]에 입력 LCS의 i번째 요소와 학습 LCS의 j번째 요소 간 거리를 대입한다. 이후, 현재 위치에서 3가지 방향(방향코드 1: 현재 위치에서 왼쪽 대각선에 있는 방향, 방향코드 2: 현재 위치에서 왼쪽으로 한칸 이동한 방향, 방향코드 3: 현재 위치에서 위쪽으로 한칸 이동한 방향)에 있는 배열 D의 요소 값에 대해 가장 최소의 값을 가지는 방향의 코드를 G[i][j]에 넣어준다. 또한 최소값을 D[i][j] 에 더해준다. 이와 같은 반복을 마치면, 배열 G의 i번째 행, j번째 열부터 시작해 backtracking 하여 dissimilarity를 계산한다. G에 저장된 요소가 방향코드 1이라면, 위쪽으로 한칸 이동하고 k의 값을 증가시킨다. 방향코드 2라면, 왼쪽으로 한칸 이동하고 k의 값을 증가시킨다. 방향코드 3이라면, 왼쪽 대각선으로 한칸 이동하고 k값을 증가시킨다. 이렇게 배열의 맨 처음(1번째 행, 1번째 열)에 도착할 때까지 이 과정을 계속 반복한다. 반복이 끝나면 배열 D의 가장 마지막 요소인 D[n][m] 값과 k 값을 나누어 dissimilarity를 구한 뒤 이것을 리턴한다.

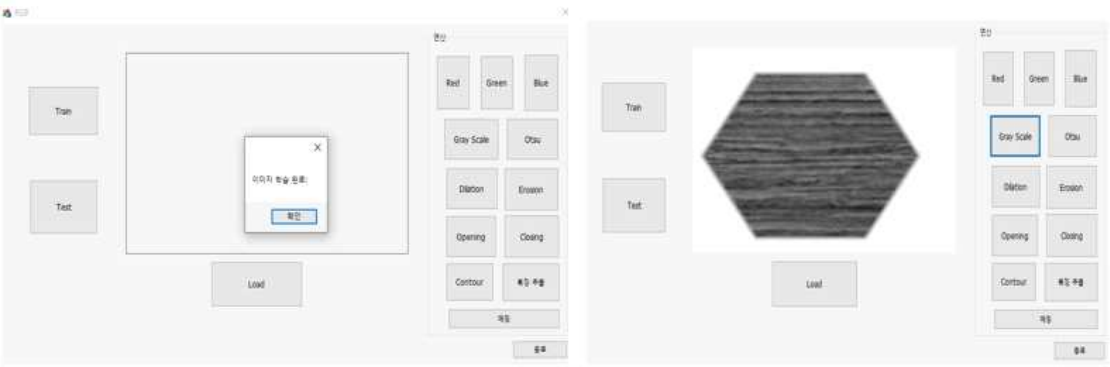
5. 인식률

	인식률(%)	오인식률(%)
육각형(hexagon)	80%(8/10)	20%(2/10)
직사각형(rectangle)	70%(7/10)	30%(3/10)
원(circle)	80%(8/10)	20%(2/10)

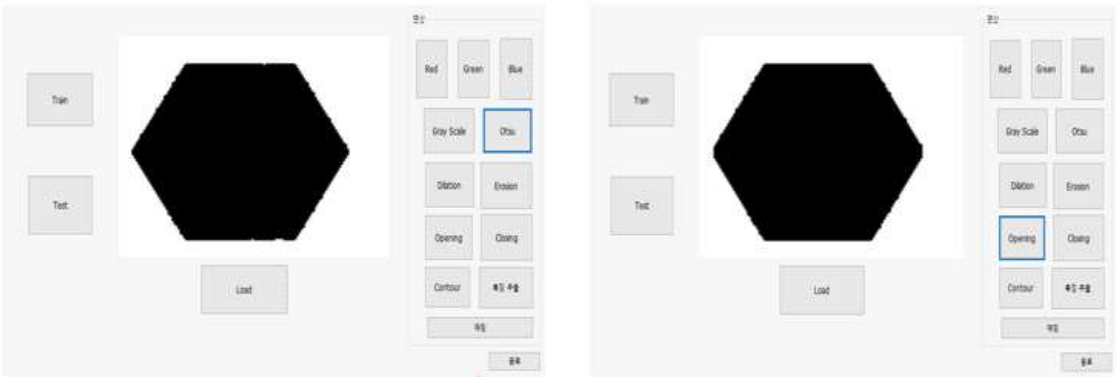
전체 인식률: 77%(23/30=0.76666)

전체 오인식률: 23%(7/30=0.23333)

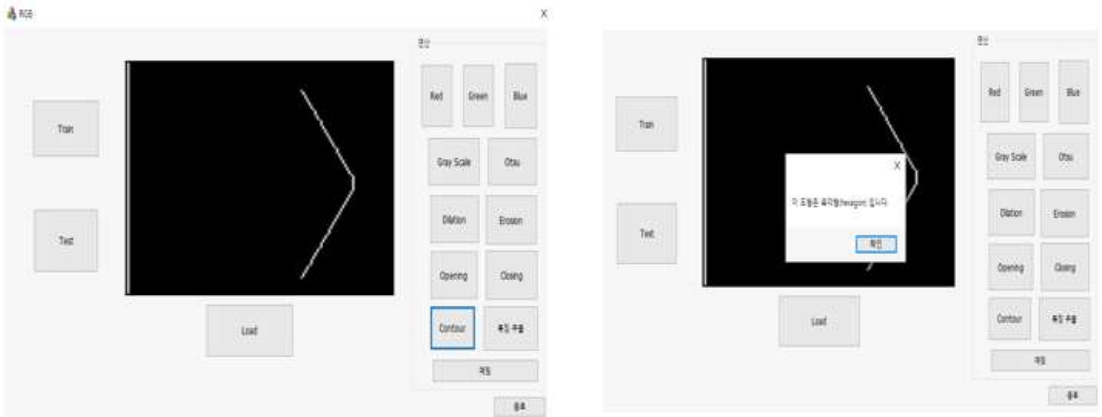
6. 실행 예시



왼: Train 버튼 눌러 이미지 학습을 완료한 모습, 오: 로딩한 이미지를 명암 영상으로 바꾼 모습



왼: Otsu 버튼 눌러 명암 이미지를 이진화한 모습, 오: Closing, Opening 버튼을 눌러 이미지에 morphological 연산을 적용한 모습



왼: Contour 버튼 눌러 경계 화소 영상을 출력한 모습, 오: 특징 추출 및 매칭 버튼을 눌러 입력 영상의 도형을 인식한 모습