Term Project

: 시각 장애인 경로 안내를 위한 점자블록 패턴 인식

지능형영상처리

Team 11

201910906 이학민

201910913 형성빈

상명대학교 SANGMYUNG UNIVERSITY

융합전자공학전공 / 지능IoT융합전공

[Contents]

Ⅰ. Introduction

1) 프로젝트 주제 선정

그림 1. 방배역 시각 장애인 선로 추락 사고를 보도한 SBS 영상 캡쳐

길거리에 보이는 노란색 점자블록은 시각 장애인들이 안전하게 이동할 수 있도록 도와주는 시설로, 시각 장애인 유도 블록 또는 안전 유도 블록이라고 한다. 시각 장애인은 흰지팡이로 점자블록을 쓸면서 앞으로 가야 할 경로를 스스로 판단할 수 있다. 하지만 시각 장애인의 눈이 되어주는 점자블록이 있음에도 불구하고, 지하철역에 있는 점자블록을 따라가다가 추락하는 사고가 종종 생긴다는 것을 접하였다.

뉴스에서는 허술한 안전조치와 노후된 시설을 사고의 원인으로 지적하였지만, 시각 장애인의 신체조건에 초점을 맞추어 상황을 재구성해보니 점자블록에만 의지하여 길을 걸어가기엔 팔 길이의 한계로 인해 바로 앞의 경로만 알 수 있을 뿐 넓은 범위의 경로를 한꺼번에 얻을 수 없다는 점이 큰 문제였다. 따라서 뉴스에서 지적한 환경적인 사항이 개선된다고 해도 돌발상황이 발생하였을 때 대처할 시간이 부족하여 위와 같은 추락 사고는 끊이지 않을 것이라고 판단하였다. 다음과 같은 문제를 해결하기 위해 시각 장애인이 점자블록 근처에 도달했을 때 앞으로 예정된 경로 정보를 음성으로 미리 제공해주는 방법을 떠올렸다.

본 프로젝트에서는 점자블록을 통해 시각 장애인에게 경로를 안내하기 위한 첫 발걸음인 점형 점자블록과 선형 점자블록의 자동 인식 구현을 목표로 한다. 시각 장애인들은 보통 선글라스를 쓰고 다니는데 선글라스에 점자블록을 인식할 수 있는 카메라가 달려있다고 가정하고 테스트 이미지를 눈높이 기준으로 촬영하였다.

2) 프로젝트 진행 일정

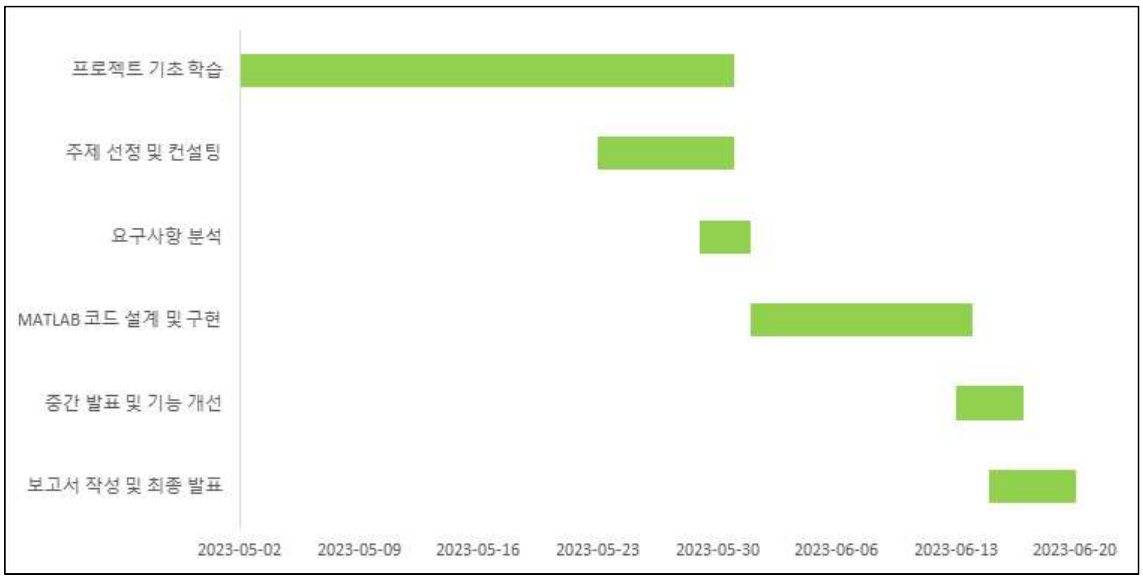


그림 2. Gantt Chart를 이용한 프로젝트 일정 도식화

프로젝트를 진행하기에 앞서 영상처리 이론에 대해 학습하고 약 1달간 Point Operation, Image Composition, Pattern Recognition, Image Segmentation 등의 이미지 처리 기법에 대한 실습을 진행하였다.

3) 프로젝트 개요

해당 교과목은 '지능형영상처리'로 이미지의 처리 기법에 대해 배우는 과목이다. 궁극적으로 구현하고자 하는 바와 같이 실제 선글라스에 카메라를 달아 촬영된 이미지를 실시간으로 처리하여 음성 안내까지 수행하는 것은 큰 제한 사항이 따른다. 또한 교과목의 목표인 '영상 데이터 분석 기술 습득'을 넘어선다. 따라서 본 프로젝트에서는 수업 시간에 배운 이론과 실습의 수준에 충실하여 점형 점자블록과 선형 점자블록을 인식하고 구분해내는 것까지 구현하는 것으로 정했다. 추후 실제로 제품을 개발하고자 한다면, 이 프로젝트에서 구현한 점자블록 인식 및 구분 기술을 응용하고 발전시키는 과정을 거쳐 최종 목표에 도달할 수 있을 것이다. 점자블록을 인식하고 종류별로 구분하는데 요구되는 사항은 다음과 같이 정리할 수 있다.

① 기울어진 이미지 수평화
② 노란색 블록 추출
③ 이미지 노이즈 제거
④ 패턴 유사도 검사를 통한 점형 점자블록 검출
⑤ 세로 방향 에지 검출을 통한 선형 점자블록 검출

Ⅱ. Body

1) Fundamentals

여러 개의 점자블록을 인식하기 위한 기초 학습으로 단일 블록 인식 테스트를 진행하였다. 이미지의 특정 패턴을 강조하고, 비교 가능한 범위로 조정하기 위해 Convolution 연산을 수행하고 에지 검출을 거쳐 해당 블록의 반복되는 패턴을 인식하였다.

① 점형 점자블록

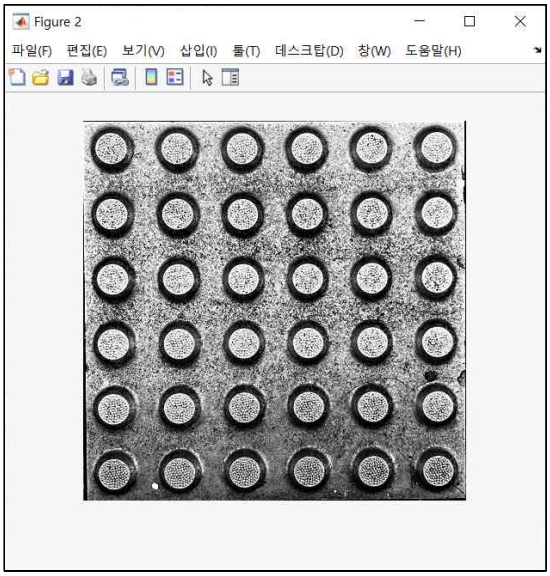
[MATLAB 코드]

```
clear; clc;

fname = "obj_image.png";
img = imread(fname);

figure(1); imshow(img);

imgY = rgb2gray(img);
imgY = histeq(imgY);
figure(2); imshow(imgY);
figure(3); imhist(imgY);

imgY = double(imgY);
obj = imgY(75:75+65,80:80+65);
figure(4); imshow(obj);

patt = flipud(fliplr(obj));
patt = patt/sum(patt(:));
patt = patt - mean(patt(:));

imgR = conv2(imgY(:,:,1), patt, 'same'); % imgY의 1번째 채널을 사용하여 conv2 수행
imgR = imgR/max(imgR(:));
figure(5); imshow(imgR);

% Edge detection using Canny & Canny filter
edge_img = edge(imgR, 'sobel');
edge_img = edge(edge_img, 'canny');
edge_img = edge(edge_img, 'canny');
edge_img = edge(edge_img, 'canny');
edge_img = edge(edge_img, 'canny');
figure(6); imshow(edge_img);
```

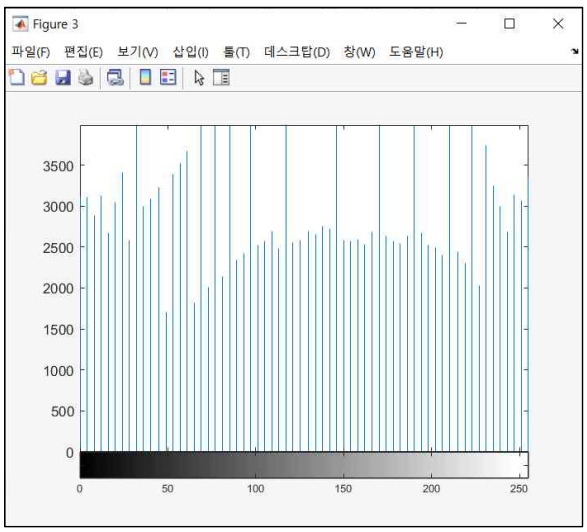
```
se = strel('square', 3);
edge_img = imdilate(edge_img, se);

% Fill the edge area with white
filled_img = imfill(edge_img, 'holes');

% Display the filled image
figure(7); imshow(filled_img);

% Binary filtering
imgB = filled_img;
se = strel('disk', 8);
imgB = imerode(imgB, se);
imgB = imdilate(imgB, se);
figure(8);
imshow(imgB);

stats = regionprops(imgB, {'Area', 'Centroid'});
tab = struct2table(stats);

% Sorting
ordered = sortrows(tab, 1, "descend");

% Show results
figure(9);
imshow(img);
hold on;
Num = 36;
title([' Detected Dot Blockes : ', num2str(Num)]);

for n=1:Num
 r = ordered.Centroid(n,1);
 c = ordered.Centroid(n,2);
 % text(r,c,num2str(n));
 text(r,c,'+','Color','red');
end

hold off;
```

[코드 실행 결과]

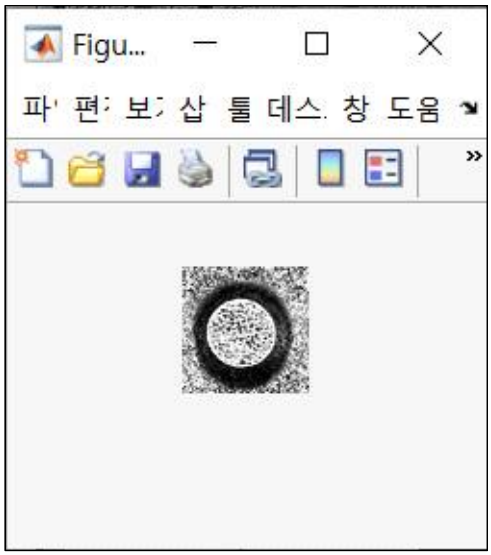
(a)



(b)

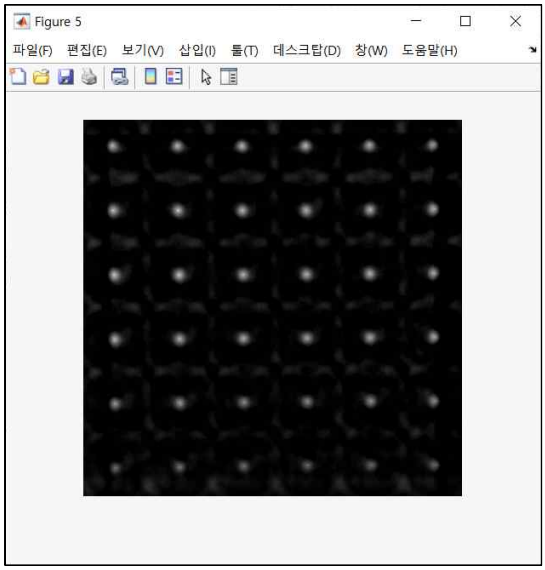


(c)

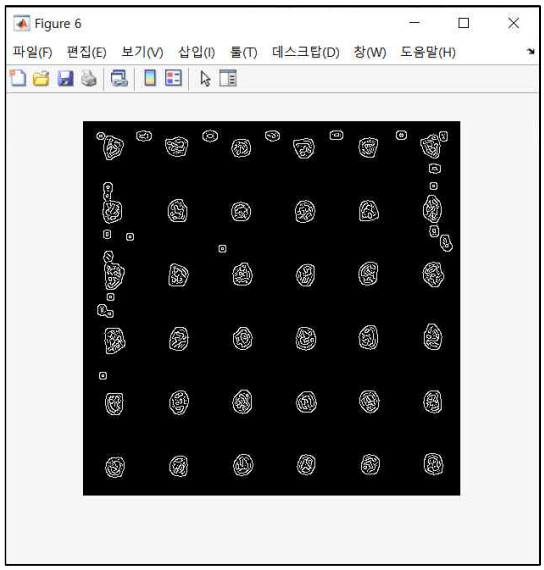


(d)

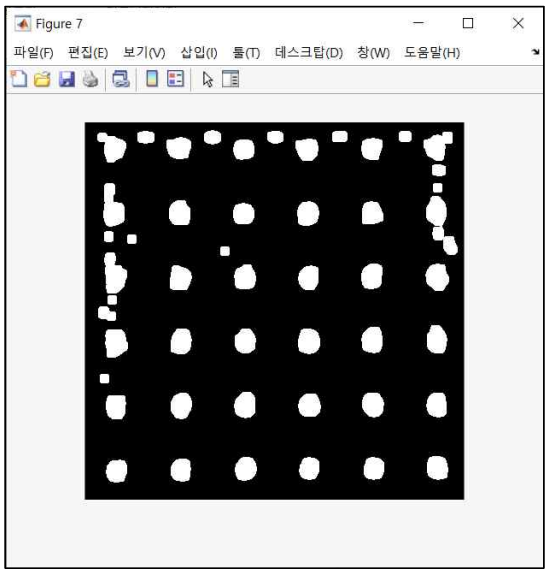
그림 3.
(a) 단일의 점형 점자블록 원본 이미지
(b) 원본 이미지를 gray 변환 후 히스토그램 균등화한 이미지
(c) figure(2) 이미지의 히스토그램
(d) 패턴 인식을 위해 점자블록의 점 하나를 object로 설정한 이미지



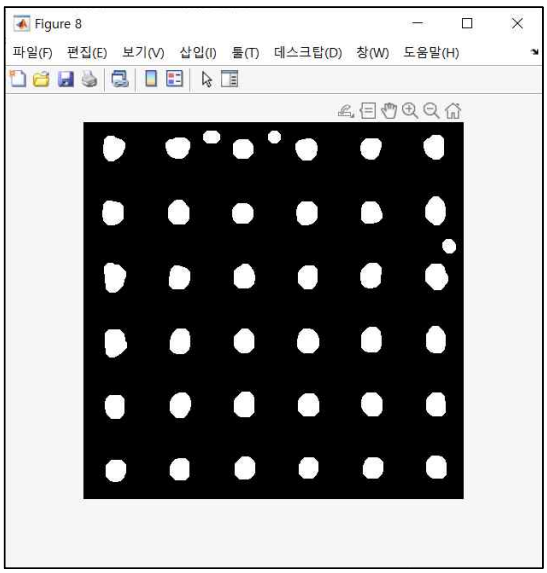
(a)



(b)



(c)



(d)

그림 4.
(a) object 이미지를 flipud, fliplr하고 patt와 figure(2) 이미지를 Convolution한 이미지
(b) High Pass Filter의 일종인 Sobel 및 Canny 필터를 통과시켜 에지를 검출한 이미지
(c) figure(6)에서 imdilate, imfill을 수행하여 에지를 증폭시킨 이미지
(d) figure(7)에서 imerode로 잡음을 제거한 후 imdilate로 다시 팽창시킨 이미지

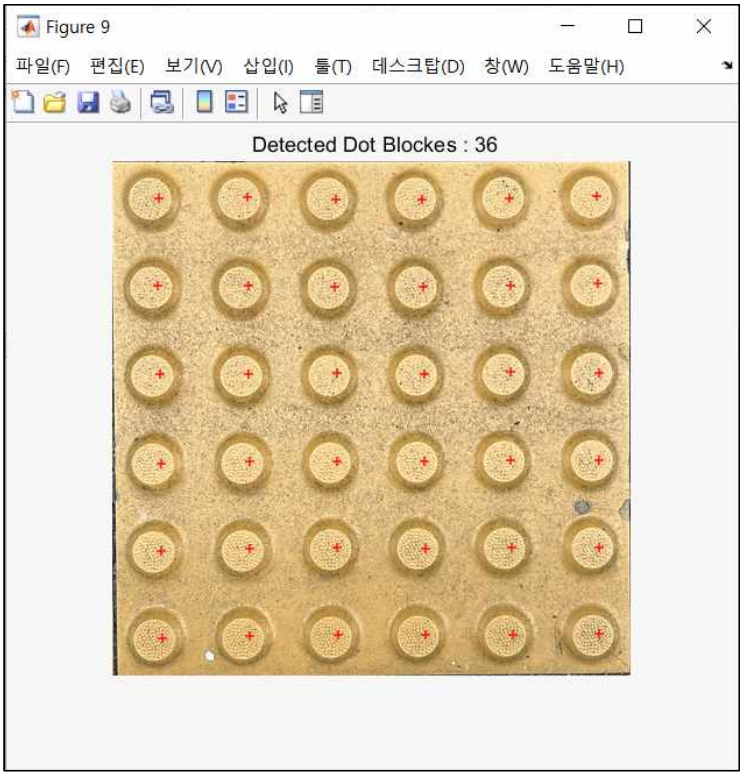


그림 5

그림 5. figure(8)의 에지 영역 area 크기를 기준으로 내림차순 정렬하여 탐지된 36개의 에지에 대해 중심 좌표에 빨간색 십자가를 표시한 이미지

점형 점자블록에서 볼록하게 튀어나온 점이 얼마나 많은지 세보는 과정을 통해 낱개의 블록이 점형 점자블록인지 아닌지 판단할 수 있다. 점자블록의 점 하나를 객체로 설정하고 전체 점자블록과 Convolution 연산하는 과정을 통해 인식하고자 하는 패턴의 성분을 강화하였다. 이후 Convolution 연산을 거친 이미지의 에지를 검출하고, 잡음을 필터링하는 과정을 거치면 점자블록의 점 하나와 비슷한 패턴인 성분만이 남게 된다.

위의 이미지에서 알 수 있듯이 점형 점자블록의 점 개수는 총 36개로, 36개의 점만 탐지하였을 때 점자블록의 모든 점이 감지되는 결과를 얻었다. 실제 임의의 각도에서 촬영된 이미지에 적용할 때는 threshold를 적절히 결정하여 감지된 점의 개수가 threshold보다 크면 점형 점자블록으로 인식하는 방식을 택하면 된다.

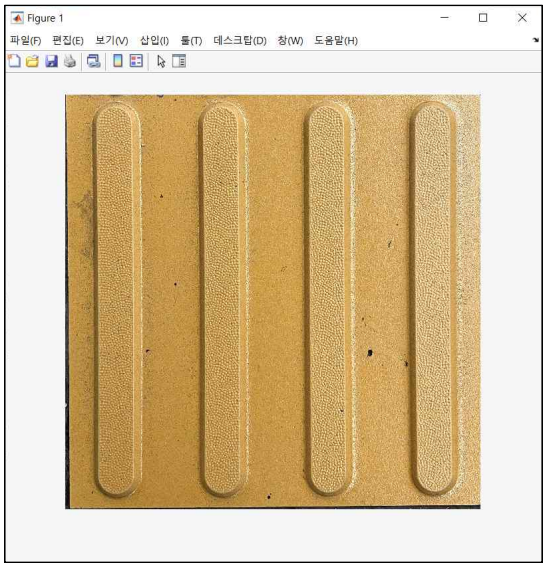
② 선형 점자블록

[MATLAB 코드]

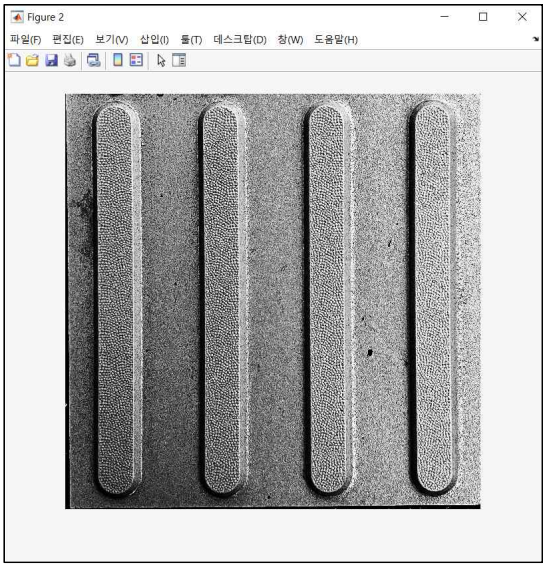
```
clear; clc;
fname = "obj_image_.jpg";
img = imread(fname);

figure(1); imshow(img);

imgY = rgb2gray(img);
imgY = histeq(imgY);
figure(2); imshow(imgY);
figure(3); imhist(imgY);

imgY = double(imgY);
obj = imgY(40:1800+40,80:300+65);
figure(4); imshow(obj);

patt = flipud(fliplr(obj));
patt = patt/sum(patt(:));
patt = patt - mean(patt(:));

imgR = conv2(imgY(:,:,1), patt, 'same'); % imgY의 1번째 채널을 사용하여 conv2 수행
imgR = imgR/max(imgR(:));
figure(5); imshow(imgR);

% Edge detection using Canny & Canny filter
edge_img = edge(imgR, 'sobel');
edge_img = edge(edge_img, 'canny');
edge_img = edge(edge_img, 'canny');
edge_img = edge(edge_img, 'canny');
edge_img = edge(edge_img, 'canny');
figure(6); imshow(edge_img);

% Fill the edge area with white
filled_img = imfill(edge_img, 'holes');
figure(7); imshow(filled_img);
```

```
% Binary filtering
imgB = filled_img;
se = strel('line', 100, 90);
imgB = imdilate(imgB, se);
figure(8);
imshow(imgB);

stats = regionprops(imgB, {'Area', 'Centroid'});
tab = struct2table(stats);

% Sorting
ordered = sortrows(tab, 1, "descend");

% Show results
figure(9);
imshow(img);
hold on;
Num = 4;
title([' Detected Circles : ', num2str(Num)]);

for n=1:Num
 r = ordered.Centroid(n,1);
 c = ordered.Centroid(n,2);
 % text(r,c,num2str(n));
 text(r,c,'+','Color','red');
end

hold off;
```

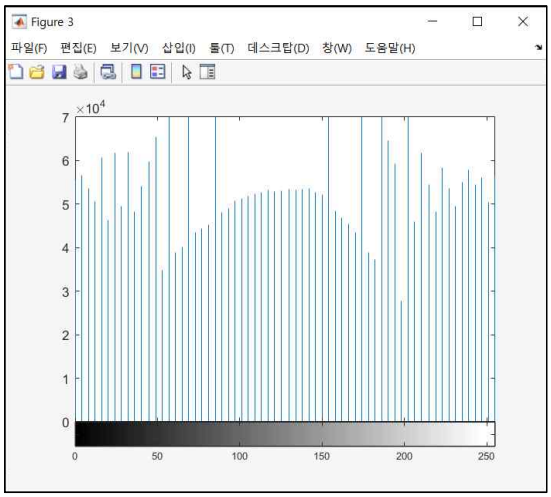
[코드 실행 결과]



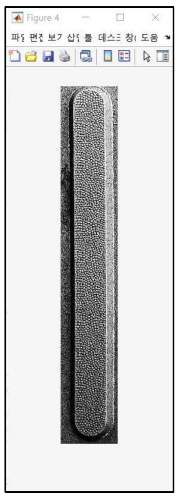
(a)



(b)

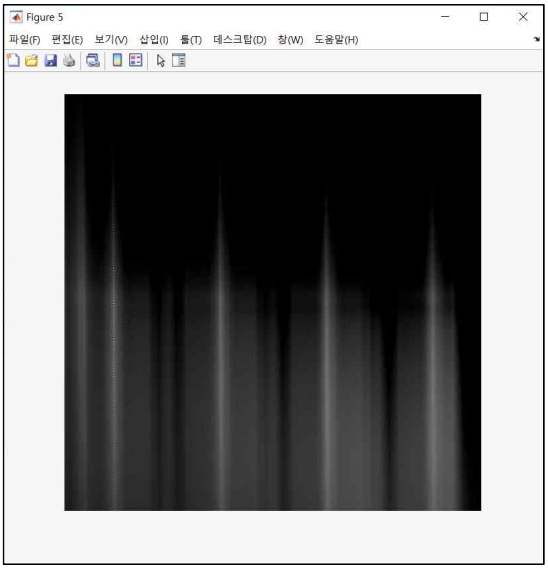


(c)

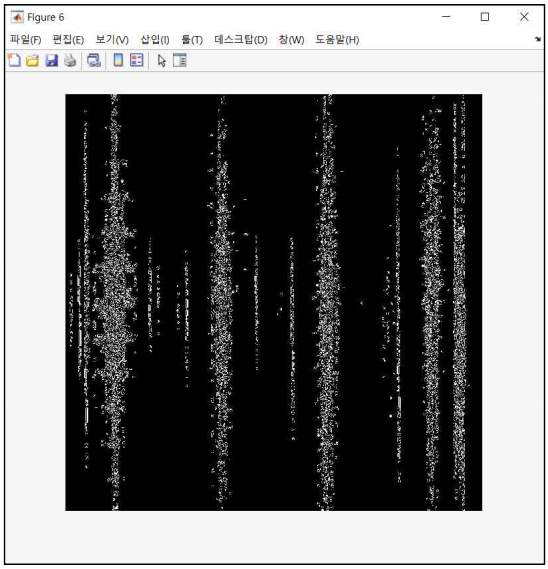


(d)

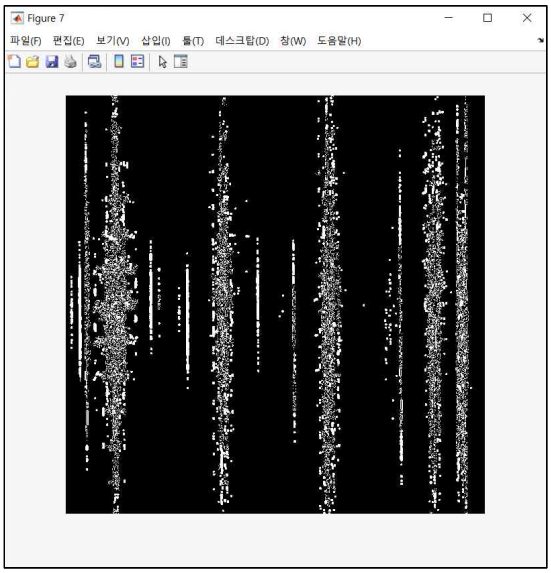
그림 6.
(a) : 단일의 선형 점자블록 원본 이미지
(b) : 원본 이미지를 gray 변환 후 히스토그램 균등화한 이미지
(c) figure(2) 이미지의 히스토그램
(d) 패턴 인식을 위해 점자블록의 긴 막대 하나를 object로 설정한 이미지



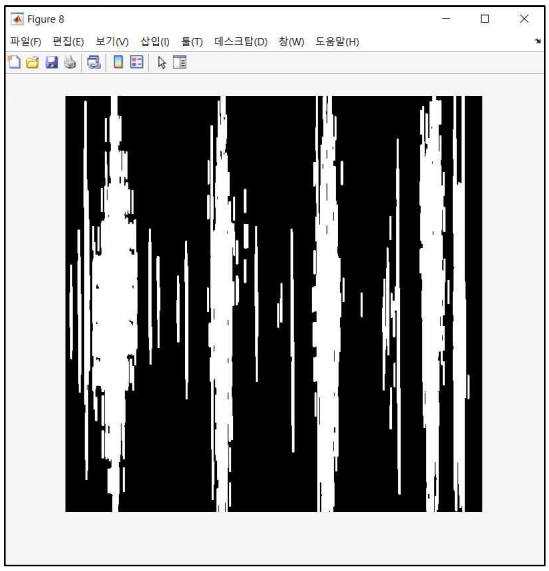
(a)



(b)



(c)



(d)

그림 7.
(a) object 이미지를 flipud, fliplr하고 patt와 figure(2) 이미지를 Convolution한 이미지
(b) High Pass Filter의 일종인 Sobel 및 Canny 필터를 통과시켜 에지를 검출한 이미지
(c) figure(6)에서 imdilate, imfill을 수행하여 에지를 증폭시킨 이미지
(d) figure(7)에서 imerode로 잡음을 제거한 후 imdilate로 다시 팽창시킨 이미지

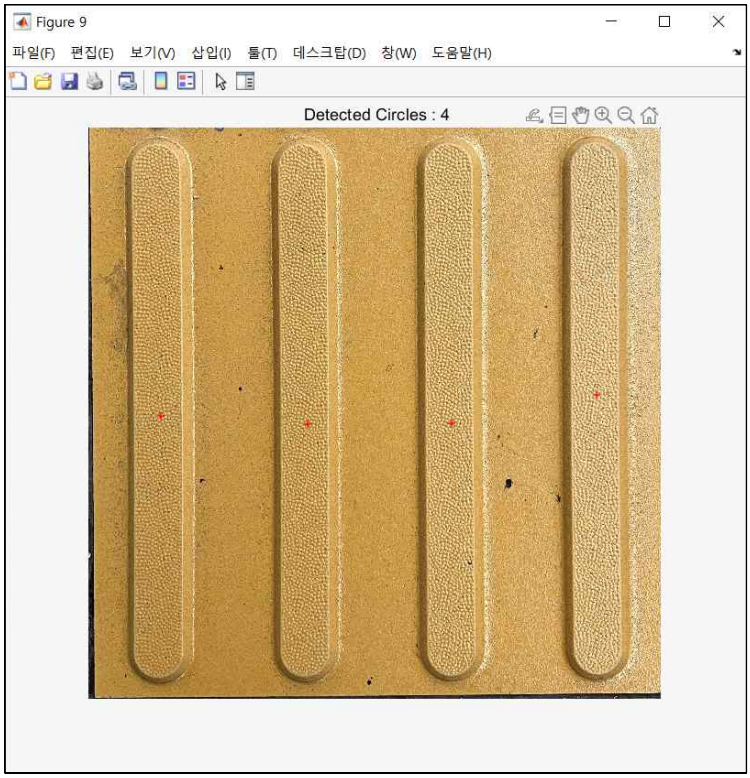


그림 8

그림 8. figure(8)의 에지 영역 area 크기를 기준으로 내림차순 정렬하여 탐지된 4개의 에지에 대해 중심 좌표에 빨간색 십자가를 표시한 이미지

앞서 점형 점자블록을 인식하는 것과 같은 방식을 선형 점자블록에 적용한 예시이다. 점형 점자블록과 달리 선형 점자블록은 긴 막대 형태의 패턴이 4개 존재한다. 점자블록의 막대 하나를 객체로 설정하고 전체 점자블록과 Convolution 연산하는 과정을 통해 인식하고자 하는 세로로 길게 늘어진 패턴의 성분을 강화하였다. 이후 Convolution 연산을 거친 이미지의 에지를 검출하고, 잡음을 필터링하는 과정을 거치면 점자블록의 막대를 나타내는 세로 에지 성분만이 남게 된다.

그림 7. (d)에서 4개의 선명한 세로 방향 에지는 선형 점자블록의 막대 패턴 4개를 의미한다. 다만 잡음 필터링 이후에도 area 크기가 일정 이하인 세로 에지 성분은 완전히 사라지지 않은 잡음으로 판단하여 선형 점자블록의 막대라고 인식하지 않는다. 이와 같은 Convolution 연산을 통한 패턴 인식 개선 기법을 Ⅱ-4)의 성능개선에 적용할 것이다.

2) 투시 변환(Perspective Transform)과 Homograpy

통상 물체가 촬영될 때 카메라 렌즈와 피사체는 정확히 수평을 이루지 않는다. 비스듬히 촬영된 이미지를 보면, 원근에 의해 먼 물체는 작아 보이고 가까운 물체는 커 보이므로 왜곡이 발생해 정확한 물체의 형태를 판가름하기 어렵다는 것을 알 수 있다. 따라서 임의로 촬영된 기울어진 이미지에서 점자블록을 제대로 인식하기 위해서는 수평화 작업이 요구된다. 프로젝트 컨설팅에서 Perspective Transform의 개념에 대해 소개 받았고, MATLAB 코드 설계에 적용하기 전 학습한 내용을 정리하였다.

우선, 이미지의 변환에 대한 수식을 이해하기 위해서 동차좌표(Homogenous Coordinate)의 개념이 필요하다. 동차좌표는 2차원 상에서 점의 위치를 2차원 벡터가 아닌 3차원 벡터로 표현하는 방법이다. 아래의 수식 1과 같이 마지막 원소에 1을 추가하여 동차좌표를 만들 수 있다. 마지막 원소가 1이 아닌 경우에는 아래와 같이 1로 스케일링한다.

$$x = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad \begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} x/w \\ y/w \\ 1 \end{bmatrix}$$

수식 1

동차좌표를 사용하면 변환 행렬을 곱하여 선형 변환을 하고 평행하게 이동하는 과정을 수식 2처럼 한 번의 행렬 곱으로 나타낼 수 있기 때문에 수식과 연산이 간단해진다.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = A \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} = A \begin{bmatrix} x \\ y \end{bmatrix} + t$$

수식 2

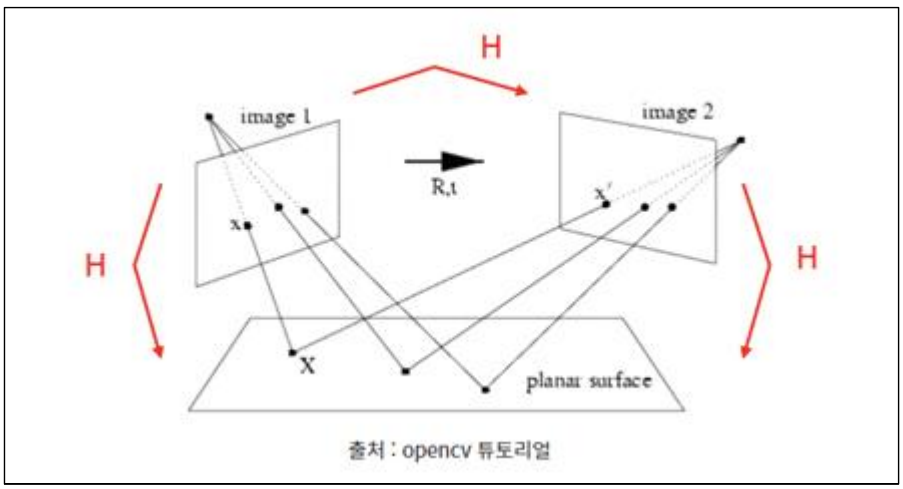


그림 9

Homograpy란 두 평면 사이의 투시 변환(Perspective Transform)을 의미한다. 그림 9에서

왼쪽 상단의 평면에서 바닥 사진을 카메라로 촬영한 것을 image1이라고 가정한다. image1은 바닥을 기준으로 정확히 90도 방향에서 촬영한 것이 아니고 왼쪽 상단에서 비스듬하게 촬영했기 때문에 사진이 기울어지는 것을 알 수 있다. 여기서 원래 바닥 사진과 image1의 관계를 Homograpy H1이라고 표현한다. 마찬가지로 오른쪽 상단 평면에서 바닥 사진을 촬영한 것을 image2라고 하면 원래 바닥 사진과 image2의 관계를 Homograpy H2로 표현 가능하다.

이때 비스듬히 촬영한 바닥 사진인 image1과 image2를 90도 각도로 수평하게 찍은 것처럼 변환하기 위해서는 두 평면 사이의 투시 변환 즉, Homograpy를 활용한다. 투시 변환은 기하학적 변환 중 하나로 3차원 공간에서의 객체를 2차원 이미지로 투영하는 과정을 나타낸다. 투시 변환은 원근 효과를 시뮬레이션하고 카메라의 위치와 방향에 따라 객체의 모습을 변형시키는 역할을 한다. 이를 통해 이미지를 수평화할 수 있다. 투시 변환은 투시 변환 행렬을 사용하는데 투시 변환 행렬은 원래 공간과 투영된 공간 사이의 변환 관계를 나타내는 3x3 크기의 행렬이다. 투시 변환 행렬은 상수 1개와 8개의 미지수로 구성되는데, 원본 이미지의 좌표 4쌍과 이에 대응되는 변환 후 좌표 4쌍을 안다면 미지수를 구해 얻어낼 수 있다. Homograpy는 이 투시 변환 행렬로, 수식 3처럼 나타낸다. 수식 3을 보면 동차 좌표를 활용하여 투시 변환 행렬과 동차 좌표의 행렬 곱을 통해 Homograpy를 나타낸 것을 알 수 있다. 아래 투시 변환에서 h_{33}은 상수이고 그 외의 8개 변수는 원본 이미지의 좌표와 변환된 이미지의 좌표이다.

$$s\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = H\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} h_{12} h_{13} \\ h_{21} h_{22} h_{23} \\ h_{31} h_{32} h_{33} \end{bmatrix}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

수식 3

Homograpy를 이용하면 기울어진 이미지를 마치 수평하게 촬영한 듯한 이미지로 변환할 수 있다. 그림 10은 건물 기준으로 약 45도 방향에서 기울어지게 찍은 이미지가 Homograpy를 통해 수평화 처리한 사진이다.

그림 10

앞서 학습한 Perspective Transform과 Homograpy를 MATLAB 코드에 적용하는 테스트를 진행하였다. 상명대학교 서울캠퍼스 제1공학관 G524호에 있는 실습용 모니터를 비스듬하게 촬영하여 모니터를 수평화하였다.

[MATLAB 코드]

```
% 이미지 로드
img = imread('test_monitor.jpg');

% 원본 이미지 출력
figure;
imshow(img);
title('Original Image');
hold on;

% 4개의 점 선택
pts = ginput(4); % 사용자가 4개의 점을 찍도록 합니다.

% 주어진 4개의 점 좌표
pts1 = pts; % 원본 좌표
pts2 = [1, 1; 4032, 1; 4032, 3024; 1, 3024]; % 변환 좌표

% 원근 변환 행렬 계산
H_perspective = fitgeotrans(pts1, pts2, 'projective');

% 이미지 변환
outputImageSize = [3024, 4032];
img2 = imwarp(img, H_perspective, 'OutputView', imref2d(outputImageSize));
```

```
% 시각화
figure;
subplot(1, 2, 1);
imshow(img);
title('Original');
hold on;
scatter(pts1(:, 1), pts1(:, 2), 'r', 'filled');
hold off;

subplot(1, 2, 2);
imshow(img2);
title('Perspective Transformed');
hold on;
scatter(pts2(:, 1), pts2(:, 2), 'w');
hold off;
```

[코드 실행 결과]

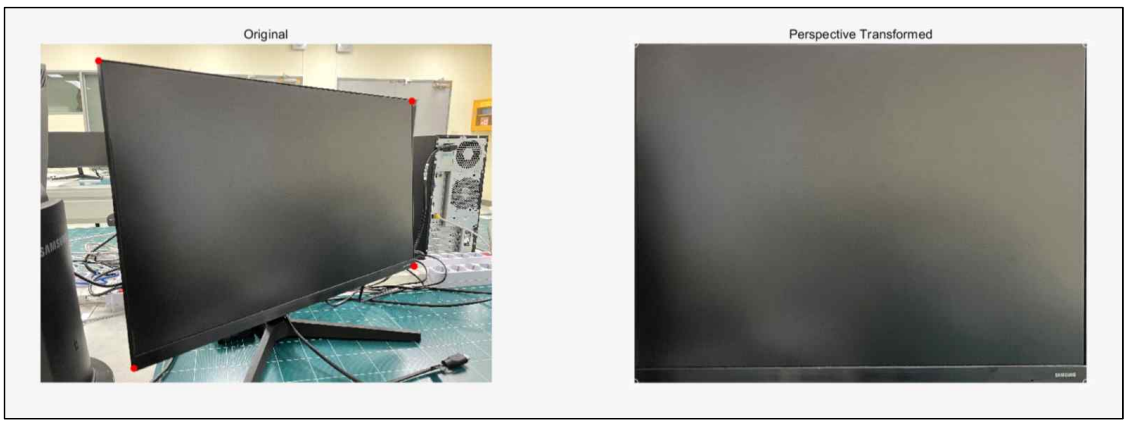


그림 11. 기울어져 촬영된 모니터를 수평화 처리한 결과 이미지

3) MATLAB 코드 설계 및 결과 분석

앞서 학습한 내용을 바탕으로 시각 장애인 경로 안내를 위한 기초 작업인 점자블록 패턴 인식을 수행한 과정 및 결과이다.

[MATLAB 코드]

```
% 블록의 정보를 저장할 배열 생성
arrayBlock = zeros(1,8); % 크기가 8인 배열을 0으로 초기화

% 몇 번째 블록인지 나타내는 변수 생성
numblk = 1;

% 종류별 블록의 개수를 저장할 변수 생성
num_goblock = 0;
num_stopblock = 0;

% 원본 이미지 로드
originalImage = imread('original_image.jpg');
originalImage = imrotate(originalImage, -90);
figure(1); imshow(originalImage);

pts1 = [1020, 1570; 1840, 1600; 2130, 3170; 750, 3120]; % 원본 좌표
pts2 = [1, 1; 3024, 1; 3024, 4032; 1, 4032]; % 변환 좌표

% 이미지에 좌표 표시
figure(1);
hold on;
plot(pts1(:, 1), pts1(:, 2), 'r+', 'MarkerSize', 10);
hold off;

% 원근 변환 행렬 계산
H_perspective = fitgeotrans(pts1, pts2, 'projective');

% 이미지 변환
PerspectiveSize = [4032, 3024];
PerspectiveImage = imwarp(originalImage, H_perspective, 'OutputView',
imref2d(PerspectiveSize));
```

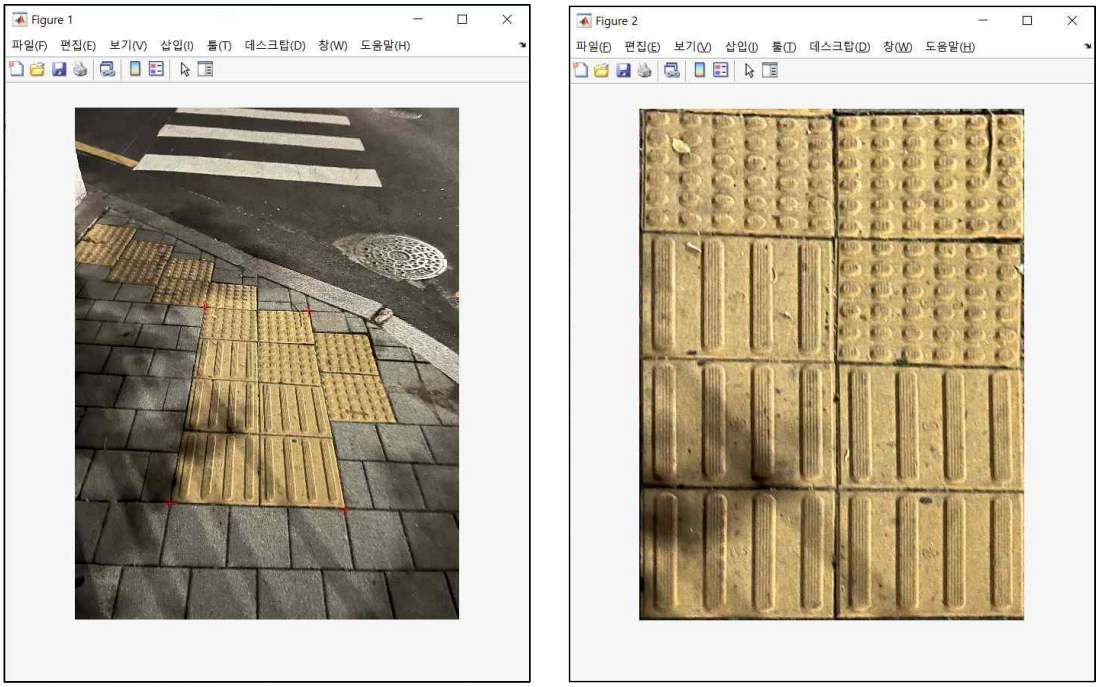
```
% 변환된 이미지 출력
figure(2); imshow(PerspectiveImage);

% objImage는 단일 점형 점자블럭 이미지임.
objImage = imread('obj_image1.png');
obj = imresize(objImage, [1000, 1500]);
figure(3); imshow(obj);

PerspectiveImage_gray = rgb2gray(PerspectiveImage); % edge검출을 위해 gray이미
지로 변환

% Edge detection using Sobel & Canny filter
EdgeImage = edge(PerspectiveImage_gray, 'sobel');
EdgeImage = edge(EdgeImage, 'canny');
figure(4); imshow(EdgeImage);

% Binary filtering
imgB = EdgeImage;
se1 = strel('square', 5);
imgB = imdilate(imgB, se1);
figure(5); imshow(imgB);
se2 = strel('line', 120, 90);
imgB = imerode(imgB, se2);
imgB = imdilate(imgB, se2);
figure(6); imshow(imgB);

% edge의 area 크기를 내림차순으로 정렬
stats = regionprops(imgB, {'Area', 'Centroid'});
tab = struct2table(stats);

% Sorting
ordered = sortrows(tab, 1, "descend");
```

```
% area 크기가 edge_threshold 이하인 엣지의 센터 좌표를 추출하여 +표시 후 제거
edge_threshold = 500;
smallEdges = tab(tab.Area <= edge_threshold, :);
centerCoords = round(smallEdges.Centroid);

figure(6);
hold on;
plot(centerCoords(:, 1), centerCoords(:, 2), 'r+', 'MarkerSize', 10);
hold off;

% 빨간색 "+"로 표시된 엣지를 0으로 만들기
imgR = bwareaopen(imgB, edge_threshold);

% 수정된 이미지 출력
figure(7);
imshow(imgR);

% imgR을 1000x1500 크기의 다수의 블록으로 나누고
% 블록별로 세로 에지 수가 일정 이상이면
% PerspectiveImage에 그 블록의 자리 위치에 맞게 +를 표시하는 코드

blockSize = [1000, 1500]; % 블록 크기
blockSizeY = blockSize(1);
blockSizeX = blockSize(2);

% 이미지의 크기 및 블록 수 계산
imageSize = size(imgR);
numBlocks = floor([imageSize(1) / blockSizeY, imageSize(2) / blockSizeX]);

% 블록 인식 및 개수 세기
for i = 1:numBlocks(1)
for j = 1:numBlocks(2)
 % 현재 블록 추출
 startIndexY = (i - 1) * blockSizeY + 1 + (8*(i-1));
 endIndexY = startIndexY + blockSizeY - 1;
 startIndexX = (j - 1) * blockSizeX + 1 + (12*(j-1));
 endIndexX = startIndexX + blockSizeX - 1;
 block1 = PerspectiveImage(startIndexY:endIndexY, startIndexX:endIndexX);
```

```
 % STOP 블록 검출
 % 템플릿 매칭 수행
 similarity = normxcorr2(obj(:,:,1), block1(:,:,1)); % 빨간 채널만 사용

 % 매칭 결과를 기반으로 블록 인식
 patt_threshold = 0.23; % 임계값 설정
 [maxValue, ~] = max(similarity(:));

if maxValue > patt_threshold
 arrayBlock(numblk) = 0;

 % 같은 패턴의 블록을 빨간색 "+"로 표시
 centerX = (startIndexX + endIndexX) / 2;
 centerY = (startIndexY + endIndexY) / 2;
 figure(2);
 hold on;
 plot(centerX, centerY, 'rx', 'MarkerSize', 20);
 hold off;

 block2 = imgR(startIndexY:endIndexY, startIndexX:endIndexX);
 % GO 블록 검출
 % 세로 엣지 검출
 verticalEdges = sum(block2, 1);

 % 세로 엣지 수가 32개 이상인 경우
elseif sum(verticalEdges >= 1) >= 32
 % 블록 중심 좌표 계산
 centerX = (startIndexX + endIndexX) / 2;
 centerY = (startIndexY + endIndexY) / 2;

 % 블록 중심에 + 표시
 figure(2);
 hold on;
 plot(centerX, centerY, 'r+', 'MarkerSize', 20);
 hold off;
```

```
 % Go 블록이면 arrayBlock에 1을 저장
 arrayBlock(numblk) = 1;
end
 numblk = numblk + 1;
end
end

numblk = numblk - 1;

for i = 1:numblk
if arrayBlock(i) == 1
 num_goblock = num_goblock + 1;
else
 num_stopblock = num_stopblock + 1;
end
end

disp(['Go 블록 개수: ', num2str(num_goblock)]);
disp(['Stop 블록 개수: ', num2str(num_stopblock)]);

for i = 1:numblk
if arrayBlock(i) == 1
 disp([num2str(i) ,'번째 블록은 Go 블록입니다.']);
else
 disp([num2str(i) ,'번째 블록은 Stop 블록입니다.']);
end
end
```

[코드 실행 결과]



(a) (b)

그림 12.
(a) 사람의 눈높이에서 촬영한 점자블록을 90도 회전시켜 figure(1)에 나타내고 투시 변환할 영역의 꼭짓점 좌표에 빨간색 십자가를 표시한 이미지
(b) 원본 이미지에서 선택한 영역을 4032x3024 크기의 이미지로 투시 변환하여 수평화한 이미지

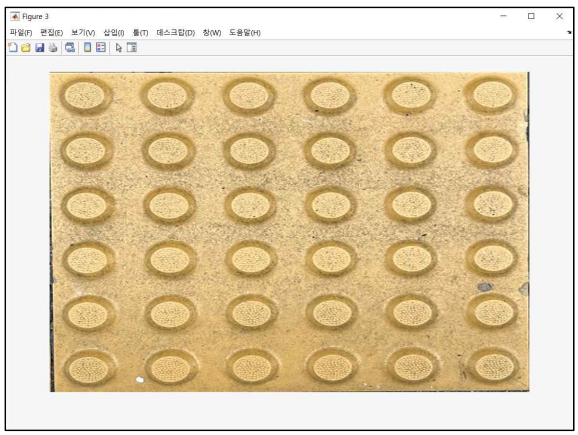
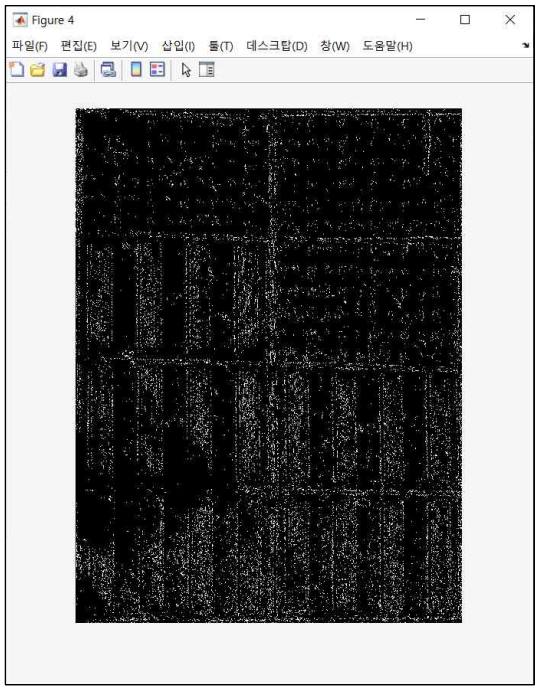
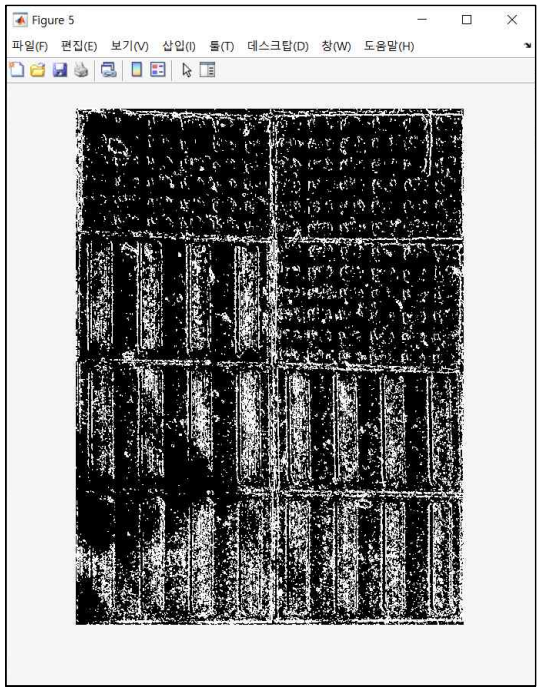


그림 13

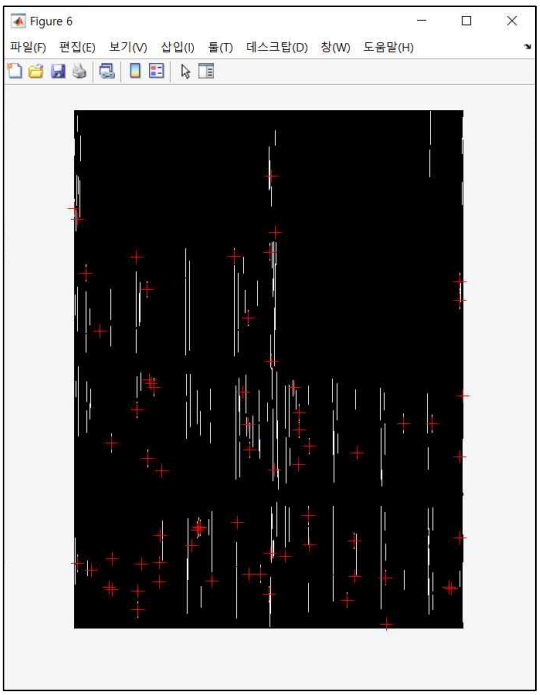
그림 13. 블록 유사도 비교를 위해 figure(2)와 템플릿 매칭할 기준 점형 점자블록 이미지를 1000x1500의 크기로 resize한 이미지



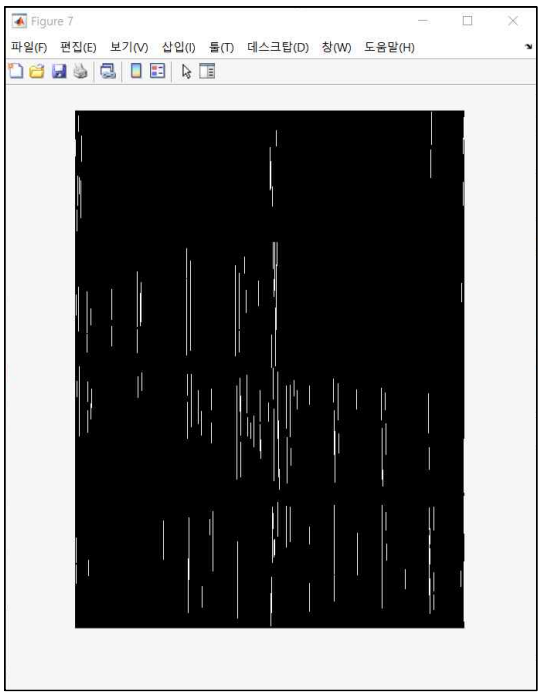
(a)



(b)



(c)



(d)

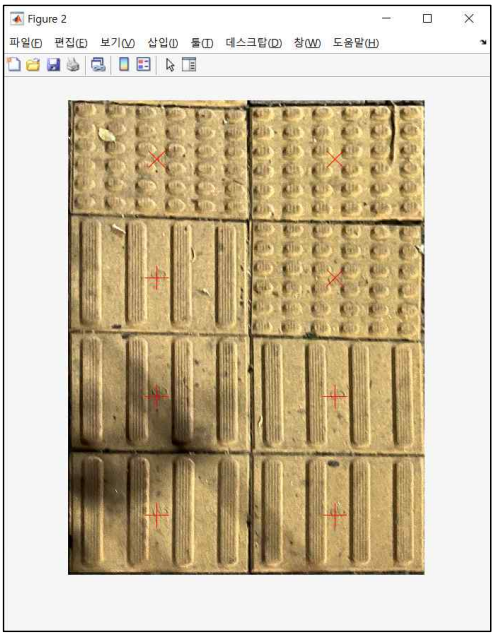
그림 14.

(a) figure(2)에서 Sobel 필터와 Canny 필터를 거쳐 에지를 얻은 결과 이미지

(b) figure(3)에서 얻은 에지를 팽창시켜 에지 성분을 증폭시킨 결과 이미지

(c) figure(4)에서 세로로 긴 에지 성분만 남기기 위해 침식 및 팽창을 거치고 에지 영역의 area가 일정 크기 이하이면 해당 에지의 중심부에 빨간색 십자가를 표시한 이미지

(d) area가 일정 크기 이상인 에지만 남도록 필터링한 최종 에지 이미지



(a)

(b)

그림 15.
(a) 점형 점자블록은 x, 선형 점자블록은 +로 표시하여 구분한 이미지
(b) 점형 점자블록과 선형 점자블록의 개수, 각 블록이 어떤 블록인지를 표시한 결과

사람의 눈높이에서 촬영된 점자블록을 인식하기 위해 Perspective Transform을 수행하였다. 원본 이미지의 좌표와 변환될 이미지의 좌표 각각 4쌍을 이용하여 fitgeotrans 함수로 투시 변환 행렬을 얻었다. 최종적으로, imwarp 함수에 투시 변환 행렬과 원본 이미지를 이용하여 이미지를 수평화하였다. 원본 이미지에서 추출한 8개의 점자블록을 수평화한 이미지는 크기를 4032x3024로 설정하였고 따라서 낱개 블록은 약 1000x1500의 크기임을 계산할 수 있었다. 1000x1500 크기로 나누어진 영역을 옮겨가면서 검사하여 낱개 블록이 각각 어떤 블록인지 구분하는 과정을 거쳤다.

점형 점자블록과 선형 점자블록을 구분하기 위하여 에지 분석을 사용하였다. 그림 14의 (d)는 에지의 area 크기가 일정 이하이면 잡음으로 취급하고 제거하여 크기가 큰 에지 성분만 남긴 이미지이다. MATLAB 코드에 담긴 에지 팽창과 침식 과정은 점형 점자블록의 에지 성분보다 선형 점자블록의 에지 성분이 생존하기 유리하도록 설계되었으므로 세로 방향의 에지 성분만이 남게 되고, 1000x1500 크기의 블록 별로 세로 방향의 에지가 일정 개수 이상 존재하면 해당 블록을 선형 점자블록으로 판단하였다.

앞서 언급한 MATLAB 코드상의 에지 팽창과 침식 과정으로는 최종 에지 검출 이미지에서 점형 점자블록의 에지가 모두 사라지기 때문에 점형 점자블록 여부를 에지를 통해 파악하는 것이 불가능하였다. 따라서 점형 점자블록은 MATLAB 내장 함수인 normxcorr2()를 이용하여 템플릿 매칭을 통해 유사도 검사를 진행하였다.

마지막으로 미리 생성하였던 점형 점자블록과 선형 점자블록의 개수를 저장하는 변수를 이용하여 조건문에 해당할 때 개수를 1씩 증가하였고 반복문을 통해 그 결과를 출력하였다.

4) 설계한 MATLAB 코드의 성능개선

앞서 개발한 점자블록 인식 코드에서 다음과 같은 문제점이 지적되었다.

① 점형 점자블록과 선형 점자블록에 해당하지 않는 일반 블록의 처리 과정 부재
② edge_threshold의 값에 따라 선형 점자블록이 점형 점자블록으로 인식되는 오류
③ 점형 점자블록을 필터링하기 위해 edge_threshold의 값을 올리다 선형 점자블록을 인식하지 못하게 되는 오류

첫 번째 개선사항인 일반 블록의 처리 과정은 점자블록을 구분하는 if 조건문에 else문을 추가하는 것으로 구현할 수 있다.
두 번째와 세 번째 문제점을 해결할 수 있도록 edge_threshold에 덜 민감하게 반응하기 위해서는 선형 점자블록이 가지고 있는 세로 방향의 에지 성분을 더 극대화하는 방법이 있다. Ⅱ-1)에서 소개한 Convolution 연산을 통한 패턴 인식 개선법을 활용하면 선형 점자블록의 세로 방향 에지 성분이 강화되고 성능개선 효과를 기대할 수 있다.

다음은 성능을 개선한 MATLAB 코드와 비교·분석 결과이다.

[MATLAB 코드]

```
% 블록의 정보를 저장할 배열 생성
arrayBlock = zeros(1,8); % 크기가 8인 배열을 0으로 초기화

% 몇 번째 블록인지 나타내는 변수 생성
numblk = 1;

% 종류별 블록의 개수를 저장할 변수 생성
num_goblock = 0;
num_stopblock = 0;
num_notblock = 0;

% 원본 이미지 로드
originalImage = imread('original_image.jpg');
originalImage = imrotate(originalImage, -90);
figure(1); imshow(originalImage);

pts1 = [1020, 1570; 1840, 1600; 2130, 3170; 750, 3120]; % 원본 좌표
pts2 = [1, 1; 3024, 1; 3024, 4032; 1, 4032]; % 변환 좌표
```

```
% 이미지에 좌표 표시
figure(1);
hold on;
plot(pts1(:, 1), pts1(:, 2), 'r+', 'MarkerSize', 10);
hold off;

% 원근 변환 행렬 계산
H_perspective = fitgeotrans(pts1, pts2, 'projective');

% 이미지 변환
PerspectiveSize = [4032, 3024];
PerspectiveImage = imwarp(originalImage, H_perspective, 'OutputView',
imref2d(PerspectiveSize));

% 변환된 이미지 출력
figure(2); imshow(PerspectiveImage);

objImage1 = imread('obj_image1.png');
obj1 = imresize(objImage1, [1000, 1500]);
figure(3); imshow(obj1);

PerspectiveImage_gray = rgb2gray(PerspectiveImage); % edge검출을 위해 gray이미
지로 변환
figure(4); imhist(PerspectiveImage_gray);

fname = "obj_image2.jpg";
objImage2 = imread(fname);

figure(5); imshow(objImage2);
objImage2 = rgb2gray(objImage2);
objImage2 = histeq(objImage2);
figure(6); imshow(objImage2);
figure(7); imhist(objImage2);

objImage2 = double(objImage2);
obj2 = objImage2(40:1800+40,80:300+65);
figure(8); imshow(obj2);
```

```
patt = flipud(fliplr(obj2));
patt = patt/sum(patt(:));
patt = patt - mean(patt(:));

ConvImage = conv2(PerspectiveImage_gray, patt, 'same');
ConvImage = ConvImage/max(ConvImage(:));
figure(9); imshow(ConvImage);

% Edge detection using Sobel & Canny filter
EdgeImage = edge(ConvImage, 'sobel');
EdgeImage = edge(EdgeImage, 'canny');
figure(10); imshow(EdgeImage);

% Binary filtering
imgB = EdgeImage;
se1 = strel('square', 5);
imgB = imdilate(imgB, se1);
figure(11); imshow(imgB);

se2 = strel('line', 120, 90);
imgB = imerode(imgB, se2);
imgB = imdilate(imgB, se2);
figure(12); imshow(imgB);

% edge의 area 크기를 내림차순으로 정렬
stats = regionprops(imgB, {'Area', 'Centroid'});
tab = struct2table(stats);

% Sorting
ordered = sortrows(tab, 1, "descend");
edge_threshold = 5000;

% area 크기가 edge_threshold 이하인 엣지의 센터 좌표를 추출하여 +표시 후 제거
smallEdges = tab(tab.Area <= edge_threshold, :);
centerCoords = round(smallEdges.Centroid);
```

```
figure(12);
hold on;
plot(centerCoords(:, 1), centerCoords(:, 2), 'r+', 'MarkerSize', 10);
hold off;

% 빨간색 "+"로 표시된 엣지를 0으로 만들기
imgR = bwareaopen(imgB, edge_threshold);

% 수정된 이미지 출력
figure(13);
imshow(imgR);

% imgR을 1000x1500 크기의 다수의 블록으로 나누고
% 블록별로 세로 에지 수가 일정 이상이면
% PerspectiveImage에 그 블록의 자리에 +를 표시하는 코드

blockSize = [1000, 1500]; % 블록 크기
blockSizeY = blockSize(1);
blockSizeX = blockSize(2);

% 이미지의 크기 및 블록 수 계산
imageSize = size(imgR);
numBlocks = floor([imageSize(1) / blockSizeY, imageSize(2) / blockSizeX]);

% 블록 인식 및 개수 세기
for i = 1:numBlocks(1)
for j = 1:numBlocks(2)

 % 현재 블록 추출
 startIndexY = (i - 1) * blockSizeY + 1 + (8*(i-1));
 endIndexY = startIndexY + blockSizeY - 1;
 startIndexX = (j - 1) * blockSizeX + 1 + (12*(j-1));
 endIndexX = startIndexX + blockSizeX - 1;
 block1 = PerspectiveImage(startIndexY:endIndexY, startIndexX:endIndexX);
```

```
 % STOP 블록 검출
 % 템플릿 매칭 수행
 similarity = normxcorr2(obj1(:,:,1), block1(:,:,1)); % 빨간 채널만 사용

 % 매칭 결과를 기반으로 블록 인식
 patt_threshold = 0.23; % 임계값 설정
 [maxValue, ~] = max(similarity(:));

if maxValue > patt_threshold
 arrayBlock(numblk) = 0;

 % 같은 패턴의 블록을 빨간색 "+"로 표시
 centerX = (startIndexX + endIndexX) / 2;
 centerY = (startIndexY + endIndexY) / 2;
 figure(2);
 hold on;
 plot(centerX, centerY, 'rx', 'MarkerSize', 30);
 hold off;

 block2 = imgR(startIndexY:endIndexY, startIndexX:endIndexX);
 % GO 블록 검출
 % 세로 엣지 검출
 verticalEdges = sum(block2, 1);

 % 세로 엣지 수가 42개 이상인 경우
elseif sum(verticalEdges >= 1) >= 42
 % 블록 중심 좌표 계산
 centerX = (startIndexX + endIndexX) / 2;
 centerY = (startIndexY + endIndexY) / 2;

 % 블록 중심에 + 표시
 figure(2);
 hold on;
 plot(centerX, centerY, 'r+', 'MarkerSize', 20);
 hold off;
```

```
% Go 블록이면 arrayBlock에 1을 저장
 arrayBlock(numblk) = 1;
else
 arrayBlock(numblk) = -1;
end
 numblk = numblk + 1;
end
end

numblk = numblk - 1;

for i = 1:numblk
if arrayBlock(i) == 1
 num_goblock = num_goblock + 1;

elseif arrayBlock(i) == 0
 num_stopblock = num_stopblock + 1;

else
 num_notblock = num_notblock + 1;
end
end

disp(['Go 블록 개수: ', num2str(num_goblock)]);
disp(['Stop 블록 개수: ', num2str(num_stopblock)]);
disp(['점자블록이 아닌 개수: ', num2str(num_notblock)]);

for i = 1:numblk
if arrayBlock(i) == 1
 disp([num2str(i) ,'번째 블록은 Go 블록입니다.']);
elseif arrayBlock(i) == 0
 disp([num2str(i) ,'번째 블록은 Stop 블록입니다.']);
else
 disp([num2str(i) ,'번째 블록은 점자블록이 아닙니다.']);
end
end
```

[비교 · 분석 결과]

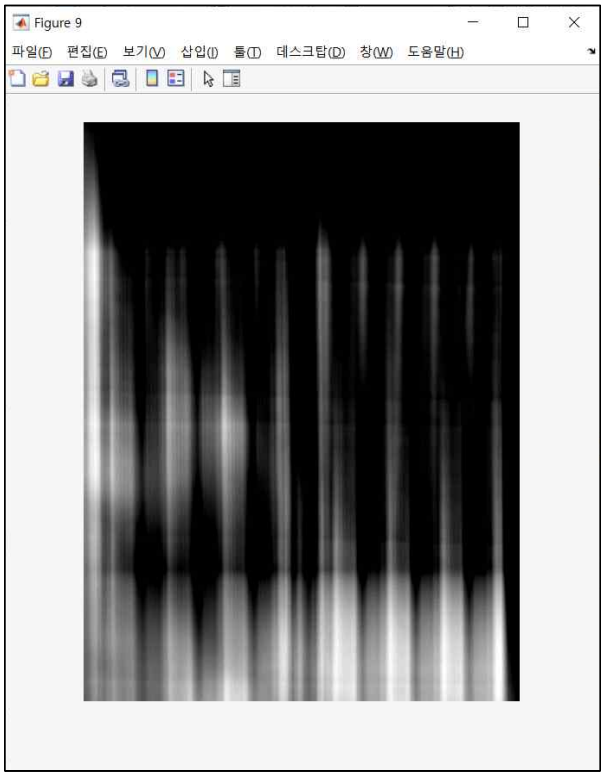
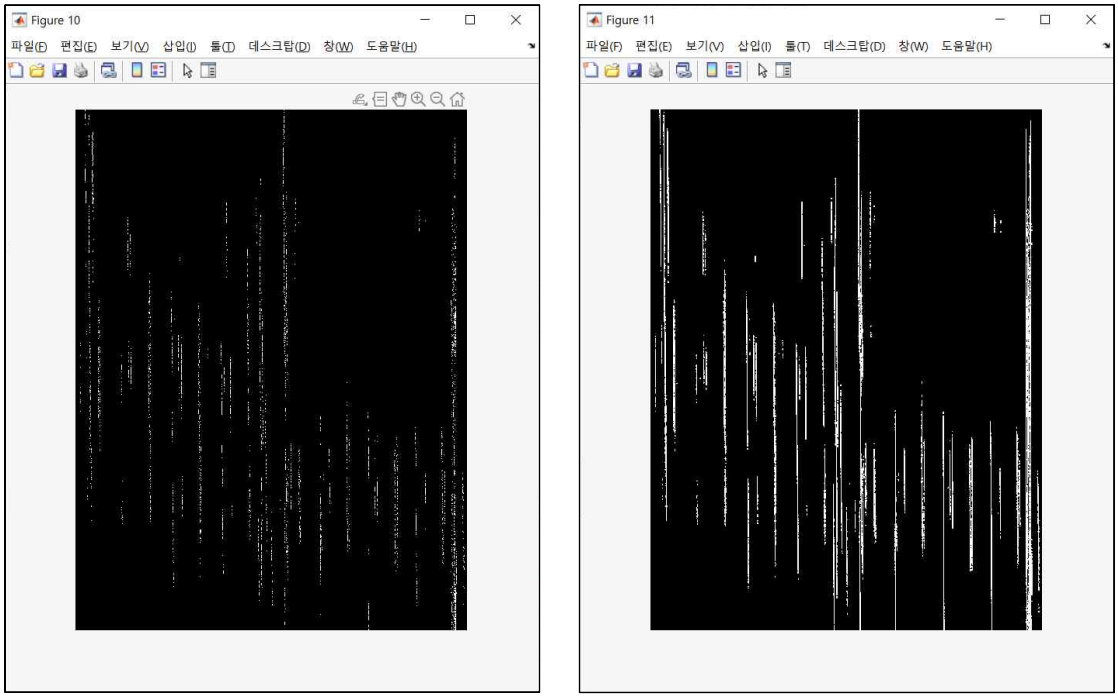


그림 16

그림 16. 선형 점자블록의 막대 1개를 object로 설정하고 object와 수평화한 이미지를 Convolution 연산한 결과 이미지

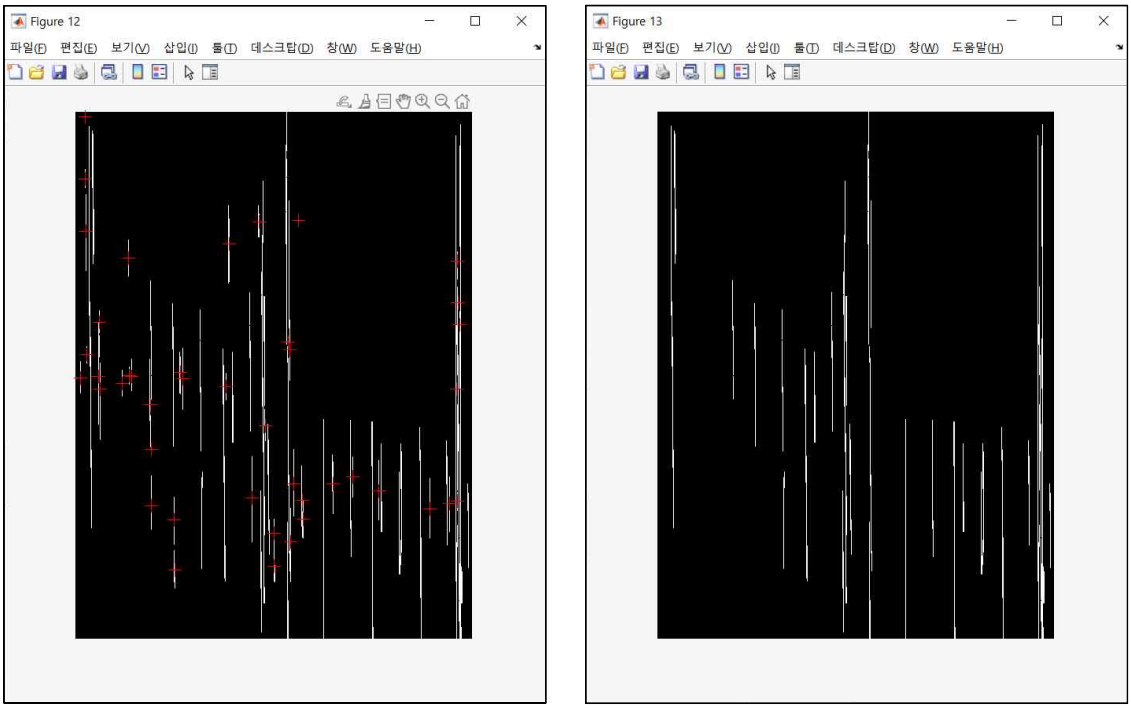
Ⅱ-1)에서 수행한 과정을 수평화 이미지에 적용하여 코드를 개선하였다. 결과적으로 선형 점자블록의 막대 성분이 도드라지는 그림 16과 같은 이미지를 얻었다. 따라서 그림 16의 에지를 분석하면 앞서 설계한 코드보다 더 명확한 결과를 얻을 수 있을 것이라 기대하였다. 아래에서 볼 수 있는 그림 17은 그림 16의 에지 성분을 추출한 결과 이미지로 세로 방향의 에지 성분이 그림 14의 (a)보다 훨씬 뚜렷하게 나타남을 보여준다.



(a) (b)

그림 17.
(a) 그림 16에서 에지를 추출한 결과 이미지
(b) 에지 성분을 강화한 이미지

그림 14와 비교해보았을 때 그림 17에서는 세로 방향 에지만 선명하게 나타남을 알 수 있다. 이전 코드의 결과인 그림 14의 (a)에서는 점형 점자블록의 점으로 인한 에지까지 같이 나타났다. 따라서 세로 방향으로 에지를 과도하게 팽창시키면 점형 점자블록의 에지가 서로 붙어버려 세로로 긴 막대 모양의 에지로 바뀌기 때문에 선형 점자블록과 구분하기 어려워진다는 단점이 있었다. 하지만 개선된 코드에서는 세로 방향의 에지만 남아 있다. 그러므로 처음 설계한 코드보다 더 과감하게 세로 방향으로 에지를 강화할 수 있다.



(a) (b)

그림 18.
(a) figure(11)에서 세로로 긴 에지 성분만 남기기 위해 침식 및 팽창을 거치고 에지 영역의 area가 일정 크기 이하이면 해당 에지의 중심부에 빨간색 십자가를 표시한 이미지
(b) area가 일정 크기 이상인 에지만 남도록 필터링한 최종 에지 이미지

(a) (b)

그림 19. (a), (b) 새롭게 설계한 일반 블록 탐지 케이스를 테스트하기 위해 블록 인지 기준값을 과도하게 높게 설정하여 코드를 실행한 결과 이미지

5) 길 안내 메시지 출력

앞서 인식한 블록 8개의 정보를 바탕으로 시각 장애인이 몇 블록이나 앞으로 걸어가다 멈춰야 하는지 알려주는 코드를 추가하였다. 비록 좌회전, 우회전 등의 세세한 정보는 제공할 수 없지만, 얼마나 앞으로 가다 멈춰야 하는지에 대한 정보만으로도 낙상사고는 충분히 예방할 수 있을 것이라는 점에서 큰 의의가 있다.

[추가 MATLAB 코드]

```
cnt1 = 0;
cnt2 = 0;

for i = numblk:-2:1
if arrayBlock(i) == 0
break;
else
 cnt1 = cnt1 + 1;
end
end

for i = (numblk-1):-2:1
if arrayBlock(i) == 0
break;
else
 cnt2 = cnt2 + 1;
end
end

if cnt1 == cnt2
 disp(['앞으로', num2str(cnt1) ,'블록 만큼 걸어가다 멈추세요.']);
else
 disp(['앞으로 ', num2str(cnt1),'에서 ' num2str(cnt2) ,'블록 만큼 걸어가다 멈추세요.']);
end
```

위의 코드를 변형, 응용하면 복잡한 경로의 안내 메시지를 얻을 수 있고 탑재된 스피커로 메시지를 음성 출력하면 음성 안내 구현이 가능할 것이다.

그림 20. 메시지 출력 결과 이미지

Ⅲ. Conclusion

1) 프로젝트의 한계 및 극복방안

① 프로젝트의 한계

본 프로젝트에서는 원본 이미지 파일에서 노란색 블록 8개만 부분적으로 설정하여 점자블록 인식을 수행하였다. 하지만 실제 제품 개발을 위해서는 카메라 렌즈에 담긴 모든 영역 중 점자블록을 구분하는 과정이 구현되어야 한다. 아쉬운 대로 이미지 전체를 수평화 작업한 결과를 바탕으로 노란색 영역 추출과 노이즈 제거를 수행한 후 에지를 추출해보았다.

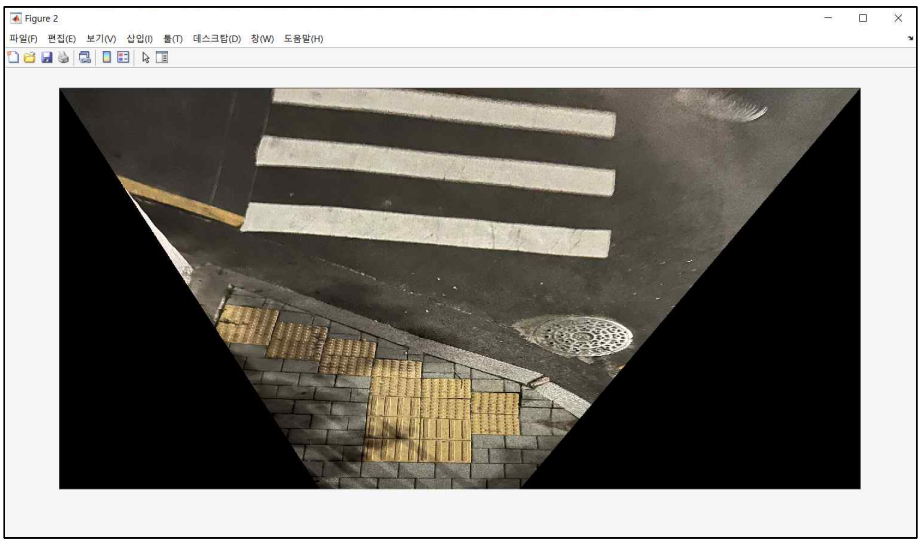


그림 21. 이미지 전체를 수평화한 결과 이미지

그림 22. 수평화한 이미지에서 노란색 영역만 추출한 결과 이미지

그림 23. 에지 검출 결과 이미지

그림 23을 보면 노란색 점자블록의 크기가 작고 스마트폰 카메라 렌즈 특성상 화질이 좋지 않아 에지 검출 성능이 떨어짐을 알 수 있다. 또한 점자블록의 에지 분석을 통한 구분을 위해서 점자블록을 더 크게 확대하는 과정이 필요하고, 어느 좌표를 기준으로 이미지를 잘라내어 확대해야 할지 깊이 생각해봐야 한다.

[MATLAB 코드]

```
% 수평화 + 노랑영역만 추출 + 잡음제거

% 원본 이미지 로드
originalImage = imread('original_image.jpg');
originalImage = imrotate(originalImage, -90);
figure(1); imshow(originalImage);

% 변환 전 피사체의 네 꼭지점 좌표 설정
originalPoints = [1, 1; 3024, 1; 3024, 4032; 1, 4032];

% 변환 후 피사체의 네 꼭지점 좌표 설정 (수평한 이미지로 변환하기 위한 좌표)
transformedPoints = [1000, 3000; 3000, 3000; 2150, 4000; 1650, 4000];

% 호모그래피 계산
H = fitgeotrans(originalPoints, transformedPoints, 'projective');
```

```
% 이미지 변환
outputImage = imwarp(originalImage, H);

% 변환된 이미지 출력
figure(2); imshow(outputImage);

% 변환 전 피사체의 네 꼭지점 좌표를 원본 이미지에 표시
figure(1); hold on;
plot(originalPoints(:, 1), originalPoints(:, 2), '+r', 'MarkerSize', 10);
hold off;

% RGB 색 공간을 HSV 색 공간으로 변환
hsvImg = rgb2hsv(outputImage);

% 노란색 범위 지정
yellowHueMin = 0.05;
yellowHueMax = 0.21;
yellowSaturationMin = 0.25;
yellowValueMin = 0.25;

% 노란색 마스크 생성
yellowMask = (hsvImg(:,:,1) >= yellowHueMin & hsvImg(:,:,1) <= yellowHueMax &
hsvImg(:,:,2) >= yellowSaturationMin & hsvImg(:,:,3) >= yellowValueMin);

% 마스크를 사용하여 원본 이미지에서 노란색 부분 추출
yellowOnlyImg = outputImage .* uint8(yellowMask);

% 결과 확인
figure(3); imshow(yellowOnlyImg);

% 3차원 이미지를 2D로 변환
yellowOnlyImg_gray = rgb2gray(yellowOnlyImg);

% 잡음 제거를 위한 미디언 필터 적용
filtered_img = medfilt2(yellowOnlyImg_gray, [4, 4]); % 필터 크기는 적절하게 조정
가능
```

```
% 영역 분할 수행
cc = bwconncomp(filtered_img);
areas = regionprops(cc, 'Area');
areas = [areas.Area];

% 영역의 크기에 따라 내림차순으로 정렬
[sortedAreas, sortedIndices] = sort(areas, 'descend');

% 특정 크기보다 작은 영역 제거
minAreaThreshold = 1200; % 적절한 값으로 설정
for i = 1:numel(sortedIndices)
if sortedAreas(i) < minAreaThreshold
 filtered_img(cc.PixelIdxList{sortedIndices(i)}) = 0;
end
end

figure(4); imshow(filtered_img);

% Edge detection using Sobel filter
edge_img = edge(filtered_img, 'sobel');
figure(5); imshow(edge_img);
```

② 극복방안

이번 프로젝트 내용과 간단하게 선보인 이미지 전체 수평화, 노란색 영역만 추출, 노이즈 제거 후 에지 검출한 내용을 바탕으로 영상처리에 대한 추가적인 지식을 습득한다면 전체 이미지 중 점자블록을 구분해내는 코드를 설계할 수 있을 것이다. 원본 이미지가 뭉개지는 이슈는 소프트웨어 개선의 방법이 있고 고성능의 하드웨어(고화질의 카메라)를 사용한다면 더욱 손쉽게 해결할 수 있을 것이다.

2) 프로젝트 수행 소감 및 기대효과

본 프로젝트의 주제는 시각 장애인의 경로 안내를 위한 점자블록 패턴 인식이다. 비록 기초적인 점자블록 인식과 구분만을 구현한 것이지만 추후 목표로 하는 실제 제품 개발을 위해 가장 뼈대가 되는 부분을 수행했다고 할 수 있다. 프로젝트를 수행하면서 시각 장애인들의 편의성과 안전성 도모에 조금이나마 일조할 수 있다는 점에 큰 가치를 느꼈다.

실제 제품이 개발된다면 시각 장애인들이 먼 거리에 있는 점자블록에 대한 정보도 미리 알게 되어 지하철 추락 사고나 교통사고 등의 불미스러운 사고를 줄일 수 있을 것이다. 나아가 많은 것들이 자동화·무인화 되어 가는 시대에 맞추어 우리가 제안하는 내용이 상용화되고 안정성을 갖추는 단계에 들어간다면 시각 장애인들이 별도의 안내견이나 보조인 없이도 안전하게 보행할 수 있는 효과를 얻을 것이라 기대하며 보고서를 마친다.

3) 참고 자료

[1] Dekel, T., Oron, S., Rubinstein, M., Avidan, S. & Freeman, W. T. in Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition 2021-2029 (IEEE, 2015).

[2] Yingping Huang / 2018 / Lane Detection Based on Inverse Perspective Transformation and Kalman Filter / KSII Transactions on Internet and Information Systems 12 (2)

[3] Transform Image in Matlab (Rotate , Scale , Affine, Homography/Perspective)
https://www.youtube.com/watch?v=Dg05elN5InM

[4] [파이썬 OpenCV] 호모그래피와 영상 매칭 - cv2.findHomgraphy
https://deep-learning-study.tistory.com/262

[5] perspective transformation와 homography 차이
https://ballentain.tistory.com/40

[6] 이미지 변환 - 동차좌표와 어파인 변환
https://datascienceschool.net/03%20machine%20learning/03.02.04%20%EC%9D%B4%EB%AF%B8%EC%A7%80%20%EB%B3%80%ED%99%98.html

[7] 이미지 템플릿 매칭
https://stackoverflow.com/questions/58158129/understanding-and-evaluating-template-matching-methods