Kyungpook National University

Department of Electronics Engineering

ID: 2023000853

#Homework 2 **Name: 윤시원**

Prof. Hyeonbeom Lee

[problem 1.]

Consider the problem from the well known children's book "Where's Wally" or "Where'sWaldo" – the fun is trying to find Wally's face in a crowd. In this exercise, you will implement atemplate matching algorithm. Using ZNCC algorithm, find where is wally.

In this exercise, you will implement a simple harris-corner detector, using only theknowledge you have so far obtained in the class. You will achieve this with the following steps:First, you will evaluate the Harris score for each pixel of the input image. Then, you will selectkeypoints based on the Harris scores. In a next step, you will evaluate simple image patchdescriptors at the selected keypoint locations. Finally, you will match these descriptors usingthe SSD norm in order to find feature correspondences between frames.

For your homework, you have to implement your code in 'harris.m' and 'selectKeypoints.m'

[Theoretical analysis & Simulation results]

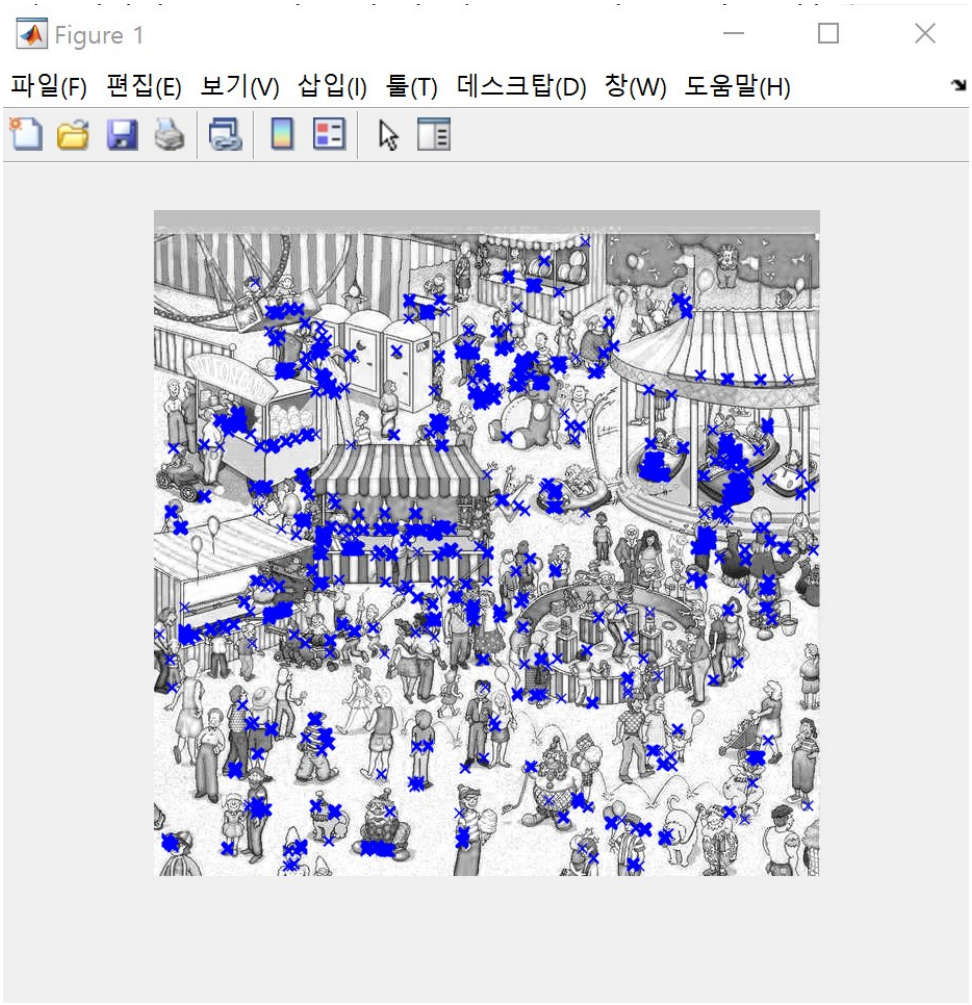
For your homework, you have to implement your code in 'harris.m' and 'selectKeypoints.m' 라고 하였으나, 함수 파일이 따로 제공된 것은 없는 듯하여, 임의로 harris corner detection 과 select keypoint 기능을 모두 수행하는 함수 m 파일 하나를 만들었다. (harris.m)

harris.m 파일을 보면 function [x, y, scores, Ix, Iy] = harris(image, value) 와 같이 함수를 선언하였다. Input 값 중 첫번째 인자인 image 는 harris corner detection 을 수행하고자 하는 이미지, input 값 중 두 번째 인자인 value 는 keypoints 를 select 하기 위한 threshold 를 조절하는 임의의 값이다. code 상에서 value 가 사용된 부분을 살펴 보면 다음과 같다.

```
% (step 5) Find points with large corner response (R > threshold)

% (step 6) Take the points of local maxima of R

avg_r = mean(mean(H));

threshold = abs(value * avg_r);        % abs : absolute value

[row, col] = find(H > threshold);
```

여기서 H 는 input image 와 동일한 크기의 영행렬로 초기 선언되어, input image 의 각 픽셀에서 harris corner 를 측정한 R 값들을 동일한 위치에 저장한 matrix 이다.

harris function 을 통해 얻은 output 값 중 x, y 는 keypoint 로 선택된 harris corner 들의 픽셀 좌표를 의미한다. y 가 row 좌표이고, x 가 column 좌표를 의미한다. 해당 좌표값을 이용해 최종적인 결과를 도출하는 lec04_wally.m 파일에서는 이미지 패치를 만들어 준다.



위는 lec04_wally.m 파일에서 harris.m 에서 선언된 harris 함수를 통해 추출한 harris corner detection keypoints 를 그림 위에 표시한 후 figure 1 이미지로 imshow 해준 것이다.

im_sub 가 추출한 이미지 패치이다. 비교를 위해 첫번째 x,y 좌표로 추출한 이미지 패치는 반복문 밖에 선언해 주었다. 그리고 범위를 지정하여 추출한 이미지 패치와 wally 를 매칭해 주었다.

```
max = 0;

im_sub = crowd(y1(1):y1(1)+24,x1(1):x1(1)+20);

% figure(2); imshow(im_sub);

in = WallySimilarity(im_sub,wally);

for i = 2 : length

    if and(and((y1(i)>350),(y1(i)<400)),and((x1(i)>200),(x1(i)<500)))
```

```
            im_sub = crowd(y1(i)-5:y1(i)+19,x1(i):x1(i)+20);
            m1 = WallySimilarity(im_sub,wally)
            if m1 > in
                in = m1;
                max = i;
            end
        end
    end
```

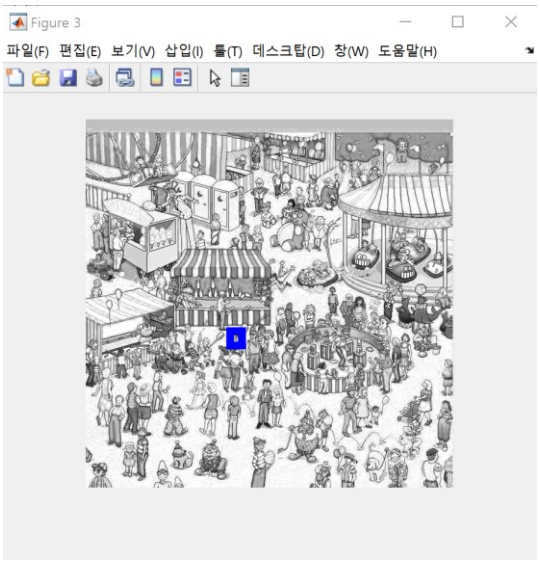
코너 포인트로 추출된 좌표보다 y 좌표는 5 더 작은 지점부터 25만큼의 길이를 갖도록 이미지 패치를 추출하였는데, 이는 추출한 좌표로 미리 직사각형을 그려 보며 임의로 지정한 값이다. (정확도를 높이기 위함)

WallySimilarity 함수는 WallySimilarity.m 파일에서 선언된 함수이며, zncc 와 동일한 기능을 수행하는 함수이다. 위 코드를 보면 해당 함수를 사용하여 반복문을 통해 zncc 결과 값이 가장 큰 좌표의 인덱스를 추출하는 것이 간단히 구현되어 있다. max 가 해당 인덱스이다.

그 결과는 다음과 같다.

위는 바로 아래 코드를 통해 imshow 한 것이다. x, y 좌표를 통해 추출한 이미지 패치 중 zncc 값이 가장 큰 것을 보여준 것이다.

```
figure(2); imshow(crowd(y1(max):y1(max)+24,x1(max):x1(max)+20))
```

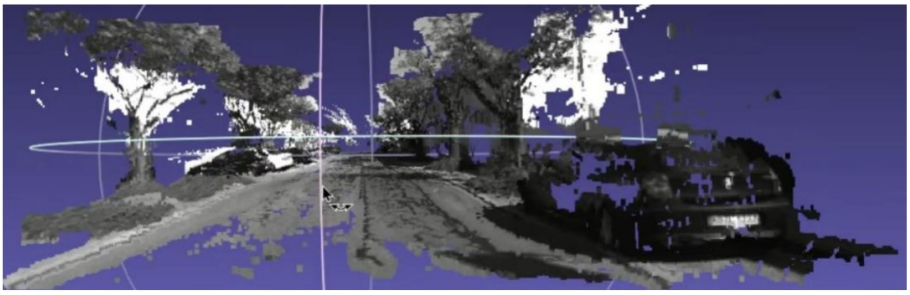


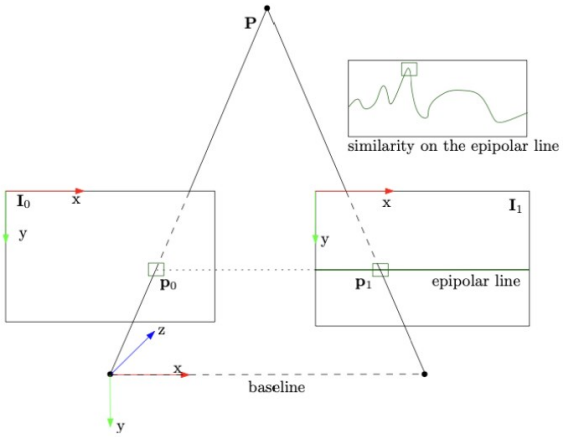
위는 다음 코드를 통해 crowd 이미지 위에 zncc 를 통해 찾아낸 wally 를 표시해준 것이다. 왼쪽이 원본 figure 3 이고, 오른쪽이 figure 3 위에서 마우스 휠을 굴려 확대한 것이다. wally 를 정확하게 잘 찾았음을 알 수 있다.

```
figure(3); imshow(crowd); hold on
rectangle('Position',[x1(max) y1(max)-5 21 25], 'EdgeColor','b', 'LineWidth',5);
hold off
```

[problem 2.]

In this exercise, you will reconstruct a 3 Pointcloud using Dense Stereo. First, try toimplement the 'getDisparity' function in the source code. Second, you have to obtain 3Dpointclouds using the disparity and triangulate the corresponding 3D point.





For your homework, we provide you with 'disparityToPointCloud' Matlab code and main file(main.m). Your job will be to implement 'getDisparity' function and analyze the process in'disparityToPointCloud' function. Finally, attached the 3D pointclod!

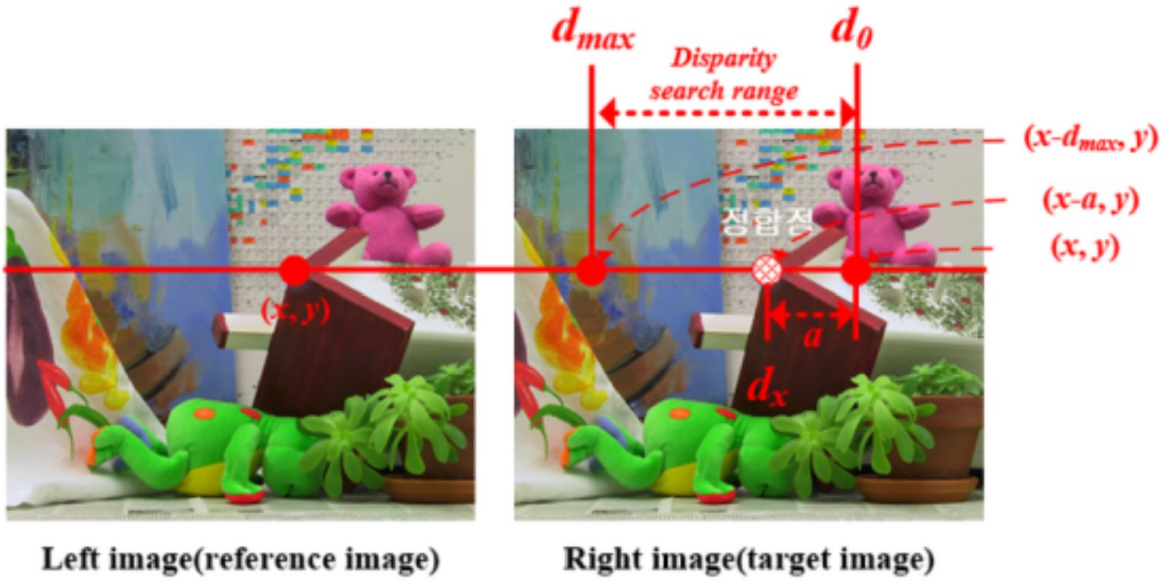
[Theoretical analysis & Simulation results]

Your job will be to implement 'getDisparity' function and analyze the process in'disparityToPointCloud' function. 라고 하였다. 따라서 제공받은 getDisparity.m code 에서 빈 부분을 수정하였다. 해당 문제에서 주어진 위 그림과 주어진 코드의 채워진 부분을 참고하면(전체 코드에서 아래쪽 부분을 살피면 ssd 방식을 사용한다고 나와 있다), 해당 코드는 stereo matching 중 local matching 인 ssd 방식을 사용해야함을 알 수 있다.

```
left_patch = single(left_img((row-r):(row+r), (col-r):(col+r)));

right_strip = single(right_img((row-r):(row+r), (col-r-max_disp):(col+r-min_disp)));
```

위 코드는 비어 있던 것을 알맞게 채운 것이다. r 은 patch radius 로 patch size = 2r+1 이다. 즉, 위 코드와 같이 (row-r):(row+r) 로 지정해 주면 patch size 범위와 같아진다. 중심점은 row 좌표인 row 이다. 즉 여기에서 기준 영상은 left 로 한 것이다. 따라서 right image 는 목표 영상이 된다. 우측은 범위를 patch size 로 지정하지 않는다. disparity

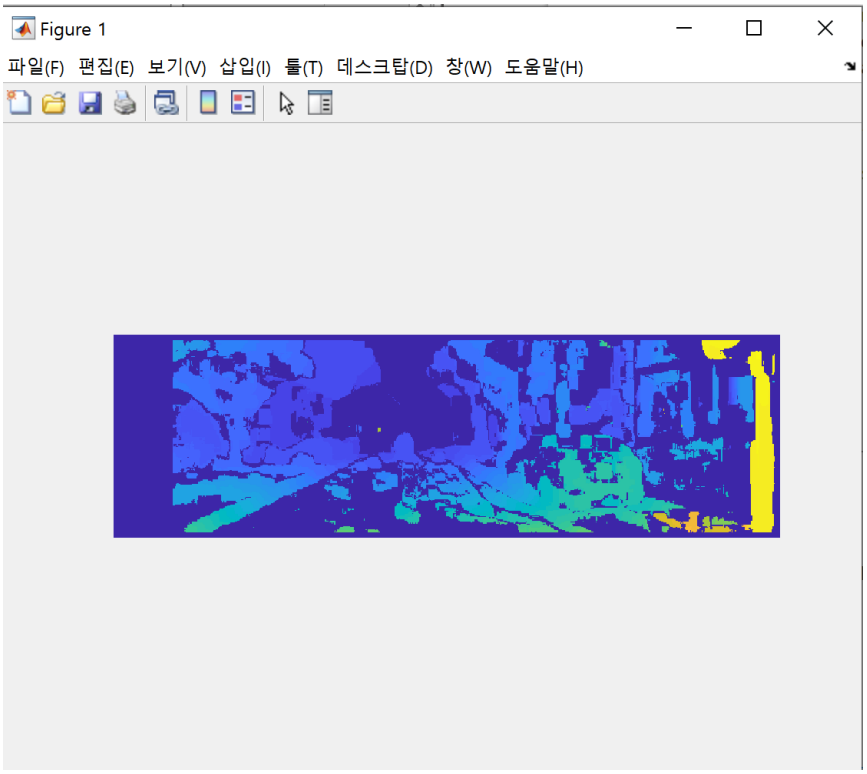
탐색 범위만큼 설정해주어야 한다. local matching 의 경우 (col-r-max_disp):(col+r-min_disp) 와 같이 설정하여 disparity search range(column 기준)를 정해 준다.



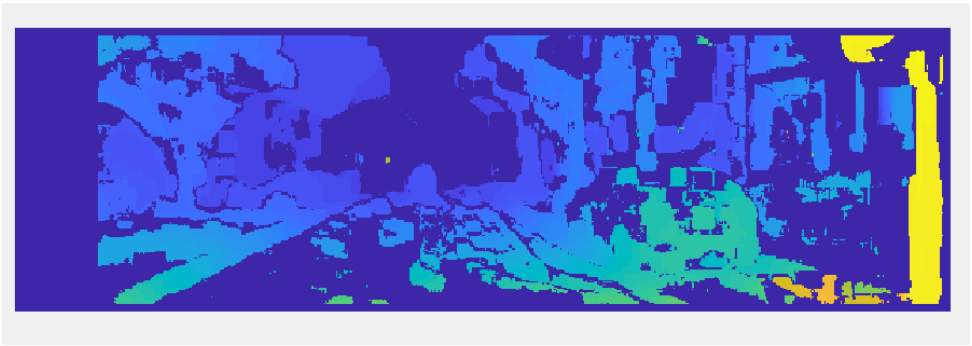
(위) disparity search 방법 (출처 : https://blog.naver.com/dldlsrb45/220879295400)

```
for i = 1:patch_size
        rsvecs(((i-1)*patch_size+1):(i*patch_size), :) = ...
            right_strip(:, i:(max_disp - min_disp + i));
end
```

이 부분도 앞에서 설명한 것과 동일한 이유로 위와 같이 채워 줄 수 있다. 이를 통해 동일한 지점에 대해 left image(기준)와 right image(목표)에 대해 disparity map 을 구성할 수 있다. 그 결과는 다음 페이지의 첨부 이미지와 같다.



(위) getDisparity function 을 통해 추출한 disparity map



(위) figure1 을 확대하여 캡쳐 후 결과 부분만 자른 것 : disparity map of left image

(위) 원본 영상 (left image(기준영상))

Your job will be to implement 'getDisparity' function and analyze the process in'disparityToPointCloud' function. 이라고 하였으므로 다음으로 disparityToPointCloud.m 을 분석하고자 한다. 주석을 참고하여 분석하면 다음과 같다.

```
function [points, intensities] = disparityToPointCloud(...
    disp_img, K, baseline, left_img)
% points should be 3xN and intensities 1xN, where N is the amount of pixels
% which have a valid disparity. I.e., only return points and intensities
% for pixels of left_img which have a valid disparity estimate! The i-th
% intensity should correspond to the i-th point.
```

N 이 유효한 disparity 값을 가지는 픽셀의 개수를 의미할 때 points 는 3N 개여야 하고, intensity 는 N 개여야 한다. 다시 말해, 유효한 disparity 추정값을 가지는 left image(기준 영상) 에 대한 반환 points 들과 intensities 만 고려한다는 것이다. i 번째 intensity 는 i 번째 point 와 대응해야 한다. 따라서 function 의 output 은 points 와 intensities 가 되는 것이다. 유효한 값을 가지는 (기준 영상의) 픽셀에 대해 points 와 intensities 를 도출해야만 하기 때문이다.

```
% Convenient way to build per-pixel coordinates.
[X,Y] = meshgrid(1:size(disp_img,2),1:size(disp_img,1));
```

위는 meshgrid 함수를 이용해 정사각행렬 x y 를 각각 생성해주는 코드이다. x 는 row 와 column size 가 getDisparity function output 인 disp_ image 의 cloumn size 와 같고, y 는 row 와 column size 가 disp_image 의 row size 와 같다.

```
% From (row, col) to (row, col, 1)
px_left = [Y(:) X(:) ones(numel(disp_img), 1)]';
```

위의 결과를 확인하면 [row col 1] 의 transpose 형태인 열 벡터들로 구성된 matrix 를 확인할 수 있다. (row 와 col 은 순서가 대응됨)

```
% Corresponding pixels in right image = pixel coords in left img minus
% disparity.
px_right = px_left;
px_right(2, :) = px_right(2, :) - disp_img(:)';
```

위 코드를 통해 px_right 는 px_left 와 동일하다고 선언한 후 Y 에 해당하는 인자들, 즉 row 인자들에서 left image 의 구성 요소를 빼 준다.

```
% Filter out pixels that do not have a known disparity.
px_left = px_left(:, disp_img(:)' > 0);
px_right = px_right(:, disp_img(:)' > 0);
```

위 코드를 통해 px_left 와 px_right matrix 중에서 disparity 가 유효한 값들만 남도록 filtering 해 준다.

```
% Switch from (row, col, 1) to (u, v, 1)
px_left(1:2, :) = flipud(px_left(1:2, :));
px_right(1:2, :) = flipud(px_right(1:2, :));
```

위 코드를 통해 matlab 의 filpud function 으로 행렬의 위아래를 뒤집어 준다. 즉, 행렬을 구성하는 행 벡터들의 순서를 역순으로 재배치하는 것과 같다.

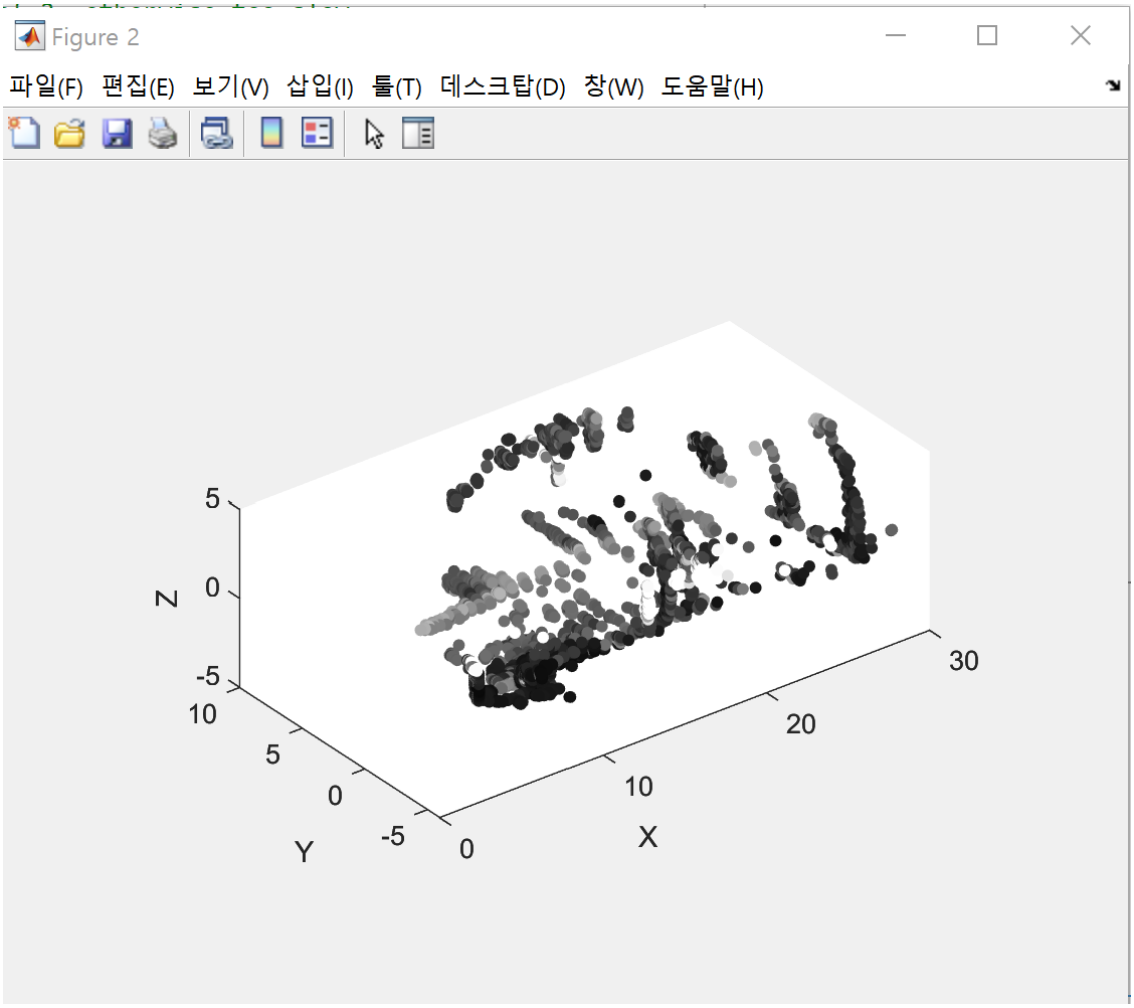
```
% Reproject pixels: Get bearing vectors of rays in camera frame.
bv_left = K^-1 * px_left;
bv_right = K^-1 * px_right;
```

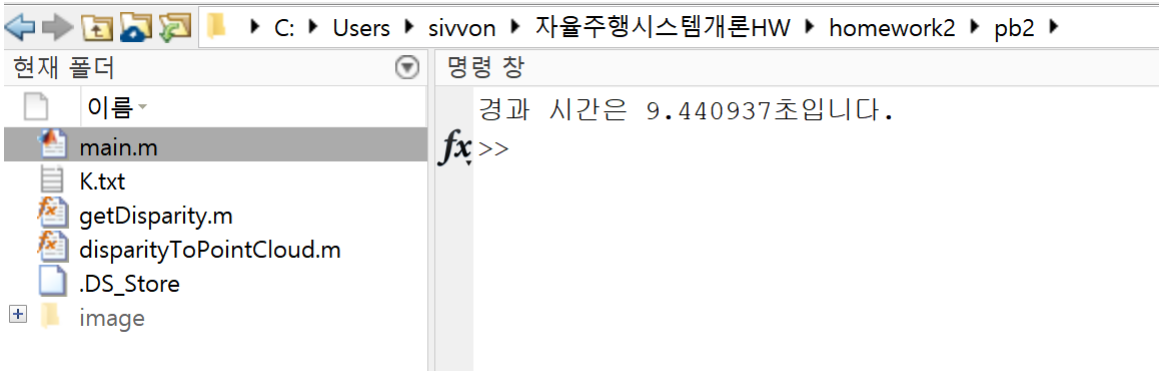
위 코드를 통해 카메라 프레임의 pixel 들을 world frame 으로 reproject 해 준다. K 는 world frame 의 points 를 camera frame 으로 transform 할 때 사용하는 intrinsic parameter 며, 여기서는 그 반대이므로 K^-1 을 사용했다. K 는 미리 지정된 값을 불러와서 사용한다.

```
% Intersect rays according to formula in problem statement.
points = zeros(size(px_left));
b = [baseline; 0; 0];
for i = 1:size(px_left, 2)
    A = [bv_left(:, i) -bv_right(:, i)];
    x = (A' * A) \ (A' * b);
    points(:, i) = bv_left(:, i) * x(1);
end

intensities = left_img(disp_img(:)' > 0);

end
```

위 코드를 통해 points 와 intensities 를 얻는다. baseline 은 스테레오 카메라 사이의 간격을 의미한다. x 는 스테레오 카메라의 eigen value 를 얻은 것이다. 그리고 이를 통해

points 와 intensities 의 값들을 얻을 수 있었고 이를 main.m 파일의 코드를 통해 3 차원에 scatter 해줄 수 있다. 이를 통해 다음 figure 2 의 결과를 얻을 수 있다.





Code of problem 1 -001. harris.m

```
% Input --> Image : 3 channel RBG image matrix
%{
output
x : n x 1 vector,

각 step : 자율주행시스템 개론 강의자료를 참고함.
%}
function [ x, y, scores, Ix, Iy] = harris( image, value )
% (step 1) compute derivatices in x and y directions (Ix,Iy) e.g. with Sobel filter

sobel_x = [ 1 0 -1; 2 0 -2; 1 0 -1];
sobel_y = [ 1 2 1; 0 0 0 ; -1 -2 -1];

filtered_x = imfilter(image, sobel_x);
filtered_y = imfilter(image, sobel_y);

Ix = filtered_x;
Iy = filtered_y;

%{
(step 2) compute Ix^2, Iy^2, IxIy
(step 3) convolve Ix^2, Iy^2, IxIy with a box filter to get sigma(Ix^2),
sigma(Iy^2), sigma(IxIy), which are the entries of the matrix M
(optionally use a Gaussian filter instead of a box filter to avoid
aliasing and give more "weight" to the central pixels)
(step 4) compute harris corner measure R (according to Shi-Tomasi or Harris)
%}
f = fspecial("gaussian");
Ix2 = imfilter(Ix.^2, f);
Iy2 = imfilter(Iy.^2, f);
IxIy = imfilter(Ix.*Iy, f);
```

```
k = 0.04;
num_rows = size(image, 1);
num_cols = size(image, 2);

H = zeros(num_rows, num_cols);

% get matrix M for each pixel
for y = 6:num_rows-6                % avoid edges
    for x = 6:num_cols-6            % avoid edges
        % calculate means -->    mean is sum/num pixels
        % Ix2 mean
        Ix2_matrix = Ix2(y-2:y+2,x-2:x+2);
        Ix2_mean = sum(Ix2_matrix(:));

        % Iy2 mean
        Iy2_matrix = Iy2(y-2:y+2,x-2:x+2);
        Iy2_mean = sum(Iy2_matrix(:));

        % IxIy mean
        IxIy_matrix = IxIy(y-2:y+2,x-2:x+2);
        IxIy_mean = sum(IxIy_matrix(:));

        % compute R, using te matrix we just created
        Matrix = [Ix2_mean, IxIy_mean;
                  IxIy_mean, Iy2_mean];
        R1 = det(Matrix) - (k * trace(Matrix)^2);

        % store the R values in our Harris Matrix
        H(y,x) = R1;

    end
```

```
end


% (step 5) Find points with large corner response (R > threshold)
% (step 6) Take the points of local maxima of R
avg_r = mean(mean(H));
threshold = abs(value * avg_r);        % abs : absolute value
[row, col] = find(H > threshold);
scores = [];

% get all the values
for index = 1:size(row,1)
        %see what the values are
        r = row(index);
        c = col(index);
        scores = cat(2, scores,H(r,c));
end


y = row;
x = col;


end
```

Code of problem 1 -002. WallySimilarity.m

```
%{
1. Consider the problem from the well known children's book "Where's Wally" or "Where's
Waldo" – the fun is trying to find Wally's face in a crowd.


In this exercise, you will implement a template matching algorithm.
Using ZNCC algorithm, find where is wally.


In this exercise, you will implement a simple harris-corner detector,
using only the knowledge you have so far obtained in the class.
```

You will achieve this with the following steps:

First, you will evaluate the Harris score for each pixel of the input image.

Then, you will select keypoints based on the Harris scores.

In a next step, you will evaluate simple image patch descriptors at the selected keypoint locations.

Finally, you will match these descriptors using

the SSD norm in order to find feature correspondences between frames.

For your homework, you have to implement your code in 'harris.m' and

'selectKeypoints.m'

%}

```
% TODO add all the other similarity metrics, including rank and census

% your code here

function m = WallySimilarity(w1, w2)
    w1 = w1 - mean(w1(:));
        w2 = w2 - mean(w2(:));

        denom = sqrt( sum(sum(w1.^2))*sum(sum(w2.^2)) );

        if denom < 1e-10
                m = 0;
        else
                m = sum(sum((w1.*w2))) / denom;
    end
```

Code of problem 1 -003. lec04_wally.m

```
close all; clear; clc


%%


crowd = iread('wheres-wally.png', 'double');
wally = iread('wally.png', 'double');


[ x1, y1, scores1, Ix1, Iy1 ] = harris( crowd, 10 );


figure(1); imshow(crowd)
hold on


for i = 1:size(scores1,2)
        plot(x1(i), y1(i), 'bx', 'MarkerSize',5);
end


hold off
length = size(y1,1);


max = 0;
im_sub = crowd(y1(1):y1(1)+24,x1(1):x1(1)+20);
% figure(2); imshow(im_sub);
in = WallySimilarity(im_sub,wally);
for i = 2 : length
        if and(and((y1(i)>350),(y1(i)<400)),and((x1(i)>200),(x1(i)<500)))
                im_sub = crowd(y1(i)-5:y1(i)+19,x1(i):x1(i)+20);
                m1 = WallySimilarity(im_sub,wally)
                if m1 > in
                        in = m1;
                        max = i;
                end
```

```
    end
end


figure(2); imshow(crowd(y1(max):y1(max)+24,x1(max):x1(max)+20))


max
figure(3); imshow(crowd); hold on
rectangle('Position',[x1(max) y1(max)-5 21 25], 'EdgeColor','b', 'LineWidth',5);
hold off
```

Code of problem 2 -001. getDisparity.m

```
function disp_img = getDisparity(...
    left_img, right_img, patch_radius, min_disp, max_disp)
% left_img and right_img are both H x W and you should return a H x W
% matrix containing the disparity d for each pixel of left_img. Set
% disp_img to 0 for pixels where the SSD and/or d is not defined, and for d
% estimates rejected in Part 2. patch_radius specifies the SSD patch and
% each valid d should satisfy min_disp <= d <= max_disp.


r = patch_radius;
patch_size = 2 * patch_radius + 1;


disp_img = zeros(size(left_img));


rows = size(left_img, 1);
cols = size(left_img, 2);


for row = (1 + patch_radius):(rows-patch_radius)
    for col = (1 + max_disp + patch_radius):(cols - patch_radius)


        left_patch = single(left_img((row-r):(row+r), (col-r):(col+r)));
```

```
            right_strip = single(right_img((row-r):(row+r), (col-r-max_disp):(col+r-min_disp)));

            % Transforming the patches into vectors so we can run them through
            % pdist2.
            lpvec = single(left_patch(:));
            rsvecs = single(zeros(patch_size^2, max_disp - min_disp + 1));
            for i = 1:patch_size
                rsvecs(((i-1)*patch_size+1):(i*patch_size), :) = ...
                    right_strip(:, i:(max_disp - min_disp + i));
            end

            ssds = pdist2(lpvec', rsvecs', 'squaredeuclidean');

            % The way the patches are set up, the argmin of ssds will not
            % directly be the disparity, but rather (max_disparity -
            % disparity). We call this "neg_disp".
            [min_ssd, neg_disp] = min(ssds);

            if (nnz(ssds <= 1.5 * min_ssd) < 3 && neg_disp ~= 1 && ...
                    neg_disp ~= length(ssds))

            disp_img(row, col) = max_disp - neg_disp;

            else

            disp_img(row, col) =0;

            end
        end
    end

end
```

Code of problem 2-002. disparityToPointCloud.m

```
function [points, intensities] = disparityToPointCloud(...
        disp_img, K, baseline, left_img)
% points should be 3xN and intensities 1xN, where N is the amount of pixels
% which have a valid disparity. I.e., only return points and intensities
% for pixels of left_img which have a valid disparity estimate! The i-th
% intensity should correspond to the i-th point.

% Convenient way to build per-pixel coordinates.
[X,Y] = meshgrid(1:size(disp_img,2),1:size(disp_img,1));

% From (row, col) to (row, col, 1)
px_left = [Y(:) X(:) ones(numel(disp_img), 1)]';

% Corresponding pixels in right image = pixel coords in left img minus
% disparity.
px_right = px_left;
px_right(2, :) = px_right(2, :) - disp_img(:)';

% Filter out pixels that do not have a known disparity.
px_left = px_left(:, disp_img(:)' > 0);
px_right = px_right(:, disp_img(:)' > 0);

% Switch from (row, col, 1) to (u, v, 1)
px_left(1:2, :) = flipud(px_left(1:2, :));
px_right(1:2, :) = flipud(px_right(1:2, :));

% Reproject pixels: Get bearing vectors of rays in camera frame.
bv_left = K^-1 * px_left;
bv_right = K^-1 * px_right;
```

```
% Intersect rays according to formula in problem statement.
points = zeros(size(px_left));
b = [baseline; 0; 0];
for i = 1:size(px_left, 2)
    A = [bv_left(:, i) -bv_right(:, i)];
    x = (A' * A) \ (A' * b);
    points(:, i) = bv_left(:, i) * x(1);
end

intensities = left_img(disp_img(:)' > 0);

end
```

Code of problem 2-003. main.m

```
clear ; close all; clc;

% Scaling down by a factor of 2, otherwise too slow.
left_img = imresize(imread('image/left.png'), 0.5);
right_img = imresize(imread('image/right.png'), 0.5);
K = load('K.txt');
K(1:2, :) = K(1:2, :) / 2;

% Given by the KITTI dataset:
baseline = 0.54;

% Carefully tuned! Do not modify!
patch_radius = 5;
min_disp = 5;
max_disp = 50;
xlims = [7 20];
ylims = [-6 10];
zlims = [-5 5];
```

```
%% Parts 1


tic;
disp_img = getDisparity(...
        left_img, right_img, patch_radius, min_disp, max_disp);
toc
figure(1);
imagesc(disp_img);
axis equal;
axis off;



%% Part 3: Create point cloud for first pair


[p_C_points, intensities] = disparityToPointCloud(...
        disp_img, K, baseline, left_img);
% From camera frame to world frame:
p_F_points = [0 -1 0; 0 0 -1; 1 0 0]^-1 * p_C_points(:, 1:10:end);


figure(2);
scatter3(p_F_points(1, :), p_F_points(2, :), p_F_points(3, :), ...
        20 * ones(1, length(p_F_points)), ...
        repmat(single(intensities(1:10:end))'/255, [1 3]), 'filled');
axis equal;
axis([0 30 ylims zlims]);
axis vis3d;
grid off;
xlabel('X');
ylabel('Y');
zlabel('Z');
```