[F21] Sensing, Perception and Actuation

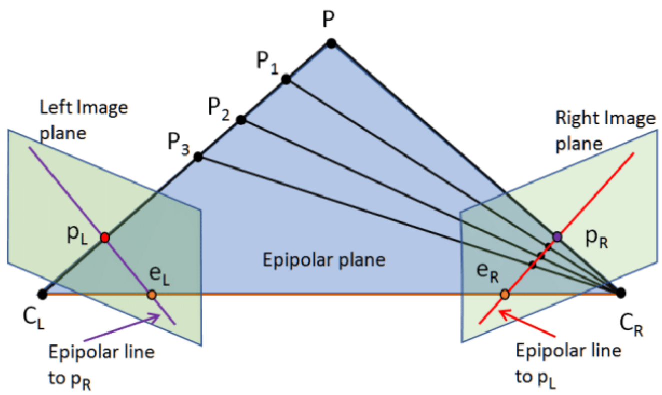
Assignment 3

Walid Shaker

w.shaker@innopolis.university

Task1: Stereo Vision

The idea of the simple stereo vision system is using 2 cameras to take images of an object, then the size and depth of this object can be estimated. The technique used to solve this problem is called epipolar geometry.



Epipolar geometry scheme

The implementation of this task goes as follows:

1.1. Read the images

```
%% 1- read images and convert them into gray
I1 = rgb2gray(imread('left15.png'));
I2 = rgb2gray(imread('right15.png'));
```

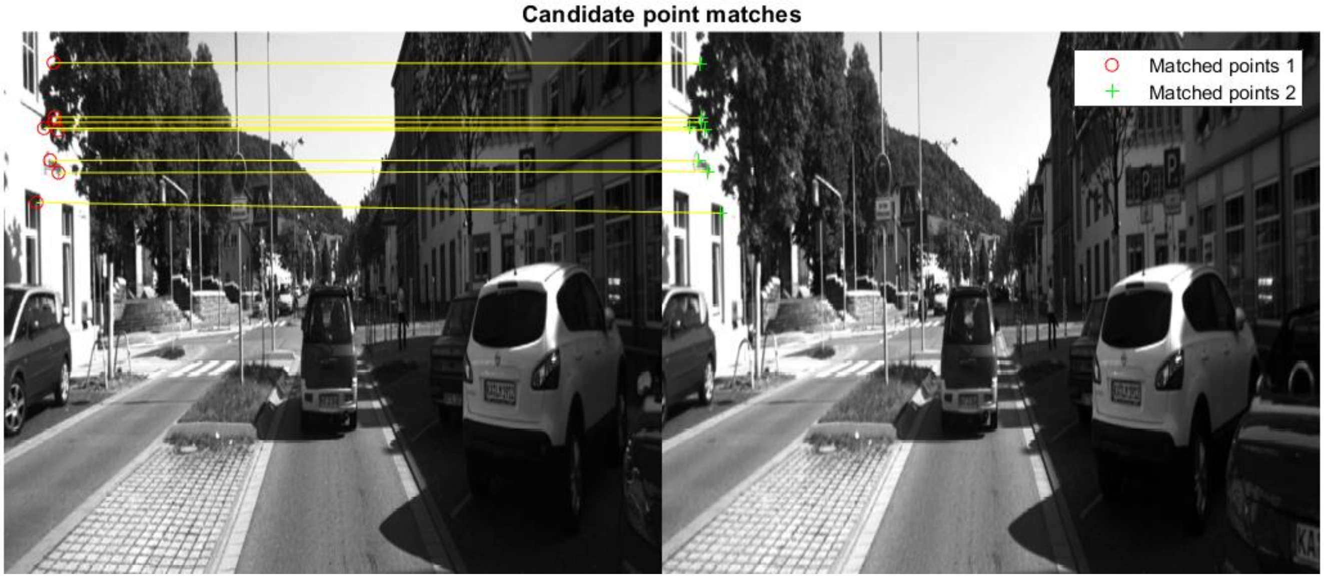
1.2. Points detection

Points are detected from the pair of images using Harris feature detection.

```
%% 2- detect points using Harris features
points1 = detectHarrisFeatures(I1);
points2 = detectHarrisFeatures(I2);
[f1, vpts1] = extractFeatures(I1, points1);
[f2, vpts2] = extractFeatures(I2, points2);
indexPairs = matchFeatures(f1, f2) ;
matchedPoints1 = vpts1(indexPairs(1:8, 1));
matchedPoints2 = vpts2(indexPairs(1:8, 2));
```

8 matched points are assigned because we need to implement 8-point algorithm. Following, the plot of the matched points.

1.3. Show matched points



1.4. Compute A matrix using DLT

First, we need to convert matched points to 3-dimensional homogonous vector by adding a column of ones, as shown below where x_l and $x_r$refers to left and right images respectively.

```
xl = [matchedPoints1.Location, ones(8,1)];
xr = [matchedPoints2.Location, ones(8,1)];
```

```
Matched points from left image:      Matched points from right image:
   43.2190    38.4712    1.0000         25.0969    37.8741    1.0000
   43.6968   142.3208    1.0000         25.5404   142.3439    1.0000
   45.1607   133.6207    1.0000         26.9462   133.1084    1.0000
   47.3584   117.9342    1.0000         29.7369   117.9219    1.0000
   50.7072   134.2417    1.0000         32.8637   133.3833    1.0000
   58.6971   143.0397    1.0000         40.4374   143.6969    1.0000
   62.2133   136.2202    1.0000         44.5108   135.5486    1.0000
   71.2019    65.3532    1.0000         49.4485    65.3767    1.0000
```

For each point, we have the coplanarity constrain:

$$x_r * F * {x_l}^T = 0$$

This leads to

$$[x_r \quad y_r \quad 1] \begin{bmatrix} f11 & f12 & f13 \\ f21 & f22 & f23 \\ f31 & f32 & f33 \end{bmatrix} \begin{bmatrix} x_l \\ y_l \\ 1 \end{bmatrix} = 0$$

The goal is to transform the previous formula into $A * F = 0$ in order to apply singular value decomposition (SVD).

Using kron function from MATLAB, matrix A is calculated as follows:

```
for n=1:8
    A(n,:) = kron(xr(n,:),xl(n,:));
end
fprintf("A matrix:\n");
disp(A)
```

Now the system is reformulated as $A * F = 0$

```
A matrix:
   1.0e+04 *
```

$$\begin{array}{ccccccccc} 0.1085 & 0.0966 & 0.0025 & 0.1637 & 0.1457 & 0.0038 & 0.0043 & 0.0038 & 0.0001 \\ 0.1116 & 0.3635 & 0.0026 & 0.6220 & 2.0259 & 0.0142 & 0.0044 & 0.0142 & 0.0001 \\ 0.1217 & 0.3601 & 0.0027 & 0.6011 & 1.7786 & 0.0133 & 0.0045 & 0.0134 & 0.0001 \\ 0.1408 & 0.3507 & 0.0030 & 0.5585 & 1.3907 & 0.0118 & 0.0047 & 0.0118 & 0.0001 \\ 0.1666 & 0.4412 & 0.0033 & 0.6763 & 1.7906 & 0.0133 & 0.0051 & 0.0134 & 0.0001 \\ 0.2374 & 0.5784 & 0.0040 & 0.8435 & 2.0554 & 0.0144 & 0.0059 & 0.0143 & 0.0001 \\ 0.2769 & 0.6063 & 0.0045 & 0.8433 & 1.8464 & 0.0136 & 0.0062 & 0.0136 & 0.0001 \\ 0.3521 & 0.3232 & 0.0049 & 0.4655 & 0.4273 & 0.0065 & 0.0071 & 0.0065 & 0.0001 \end{array} \begin{bmatrix} f11 \\ f21 \\ f31 \\ f12 \\ f22 \\ f32 \\ f13 \\ f23 \\ f33 \end{bmatrix} = 0$$

1.5. Compute F matrix

In order to find F, we have to find the null space vector of A by applying SVD on matrix A and the solution is the last column of V matrix.

```
%% 4- Apply SVD to get fundamental matrix
[U, D, V] = svd(A);
Fa = reshape(V(:,9),3,3);
Fa_rank = rank(Fa);
```

The obtained F matrix is

```
Fundamental matrix:
    0.0000    0.0005   -0.0658
   -0.0006    0.0000    0.0208
    0.0735   -0.0292    0.9945

Rank of fundamental matrix: 3
```

As shown above the rank of this matrix is 3, but the F matrix has a constrain that it should be a homogenous matrix of rank 2.

To solve this problem, we have to set the smallest singular value to zero. So, another SVD is applied on the obtained F matrix and rank is enforced to 2 as follows:

```
%% 5- Apply a second SVD for the obtained F to enforce the rank to 2
[Ua, Da, Va] = svd(Fa);
F = Ua * diag([Da(1,1), Da(2,2), 0]) * transpose(Va);
F_rank = rank(F);
```

The output of this second SVD is

```
Fundamental matrix after second SVD:
    0.0000    0.0005   -0.0658
   -0.0006    0.0000    0.0208
    0.0735   -0.0292    0.9945

Rank of fundamental matrix after second SVD: 2
```

To validate our estimation F matrix is reshaped into (9×1) size and the multiplied by A matrix, the result should be a (8×1) vector of zeros.

```
f = reshape(F,9,1);
x = A*f;
```

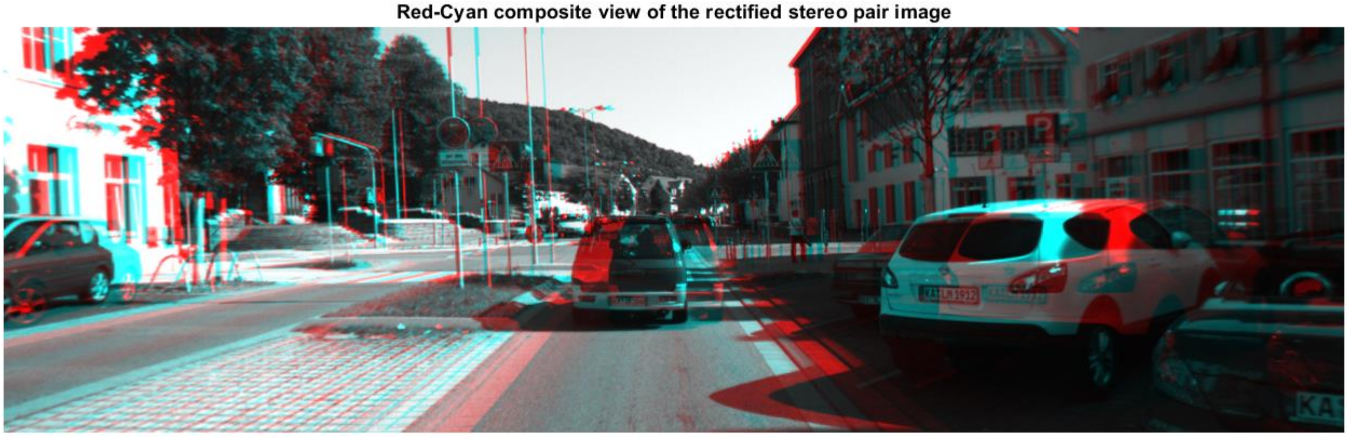
```
A*vect(F):
   -0.0005
   -0.0046
   -0.0041
   -0.0033
   -0.0042
   -0.0049
   -0.0046
   -0.0014
```

1.6. Disparity Map

A composite view of the rectified stereo pair of images is generated below:

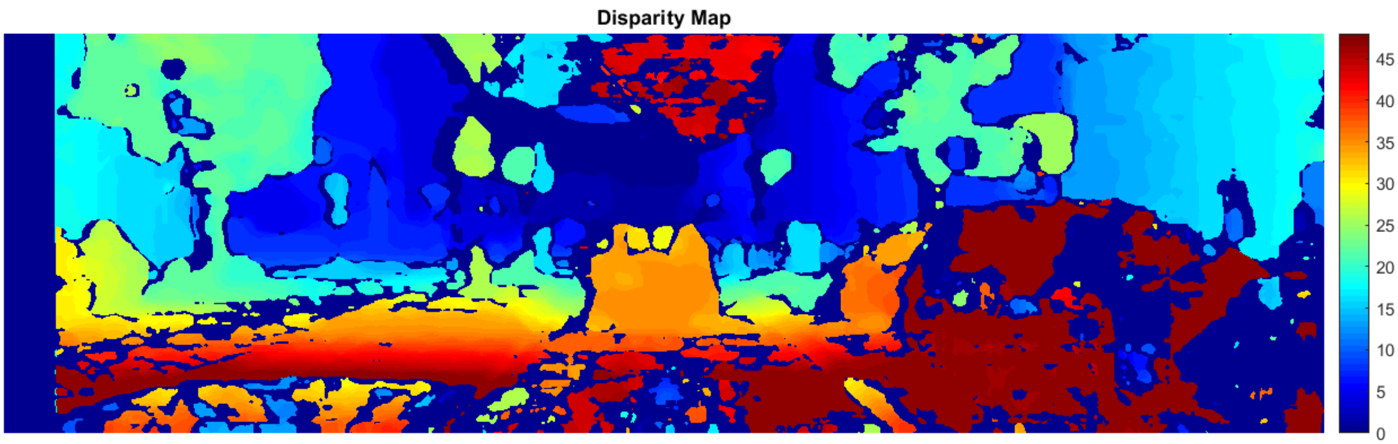
```
B = stereoAnaglyph(I1,I2);
figure;
imshow(B);
title('Red-Cyan composite view of the rectified stereo pair image');
```

Red-Cyan composite view of the rectified stereo pair image

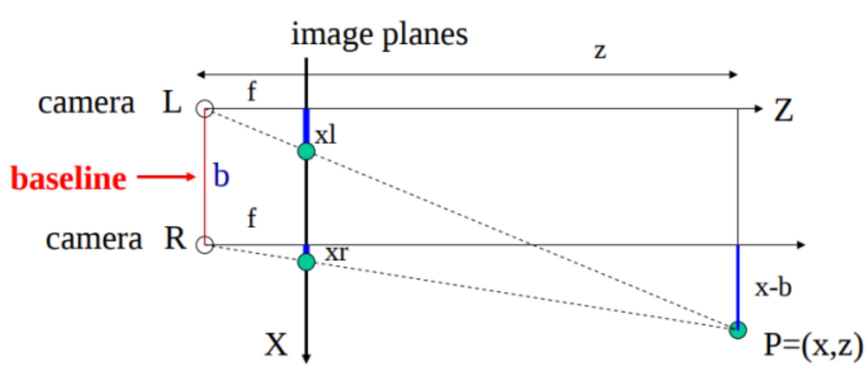


Then disparity map is generated using the below code:

```
disparityRange = [0 48];
disparityMap = disparity(I1,I2,'DisparityRange',disparityRange,'UniquenessThreshold',20);
figure;
imshow(disparityMap,disparityRange);
title('Disparity Map');
colormap jet;
colorbar;
```



Using stereo images triangulation, the depth of any point can be estimated.



Stereo images triangulation

The disparity can be estimated as:

$$disparity = x_l - x_r = \frac{b * f}{depth(z)}$$

where b is the baseline (10 cm) and f is the focal length (2.8 mm). Using this formula, the depth of any point can be estimated.

$$depth(z) = \frac{b * f}{x_l - x_r}$$

Task2: Object Center Depth

In this task, RealSense depth camera 435i is used to get a RGB image and depth map of a yellow cube. The depth map (Depth.png) generated is 320×240 and it is accompanied with .raw image of the same size. While the (RGB.png) image is 640×480.

The idea is to extract the object from the RGB image through color thresholding, then estimate the center of that object. Finally, using the raw dat for depth map generated from the camera, actual depth of the object center should be calculated and compared with the ground truth which is measured with ruler.

The implementation of this task goes as follows:

2.1. Read depth map .raw file

```
%% 1- read depth map .raw file
fid = fopen('Depth.raw', 'r');
depth_data = fread(fid, [240,320]);
fclose(fid);
```

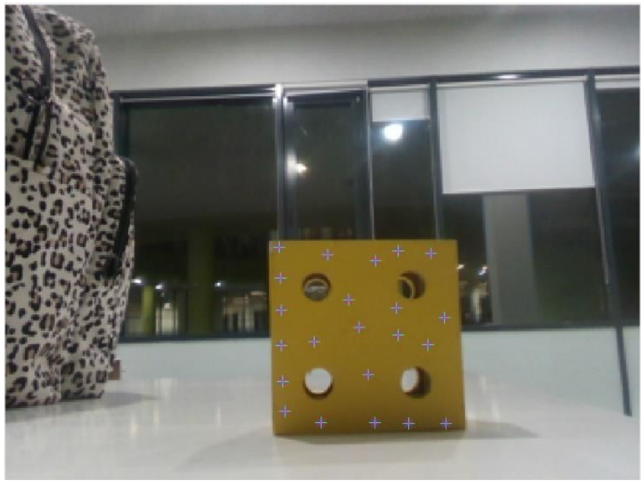
2.2. Read RGB image

In this section, RGB is uploaded and resized to the depth map size.

```
%% 2- read rgp image
rgb_img = imread('RGB.png');
figure(1);
imshow(rgb_img);
title('RGB Image');
% resize our image to 320 x 240 to match our depth map
rgb_img_resized = imresize(rgb_img,[240,320]);
figure(2)
imshow(rgb_img_resized);
title('Resized RGB Image');
```

1.3. Image thresholding

Using impixel() function, random points are selected on the object in order to get the range for each channel.



After that, .xlsx file is exported (threshold.xlsx) contains the range of each channel from 0 to 255.

This step would help in selecting the appropriate range for thresholding.

Below is the algorithm for thresholding.

```
I = rgb_img_resized;
red = I(:,:,1); green = I(:,:,2); blue = I(:,:,3);
% d = impixel(I);
thres = xlsread('threshold.xlsx');
out1 = red > min(thres(:,1)) & red < max(thres(:,1))...
    & green > min(thres(:,2)) & green < max(thres(:,2)) & ...
    blue > min(thres(:,3)) & blue < max(thres(:,3));

figure(3);
subplot(2,2,1);
imshow(out1);
title('output 1');
```

A	B	C
132	108	35
139	110	25
138	111	33
140	112	32
138	108	44
121	97	38
104	80	24
96	73	22
94	71	26
93	70	27
101	80	31
111	90	36
131	104	40
133	110	38
135	108	36
133	106	30
130	104	33
130	105	30
133	107	29
137	108	28
137	109	30
136	110	28
135	108	28

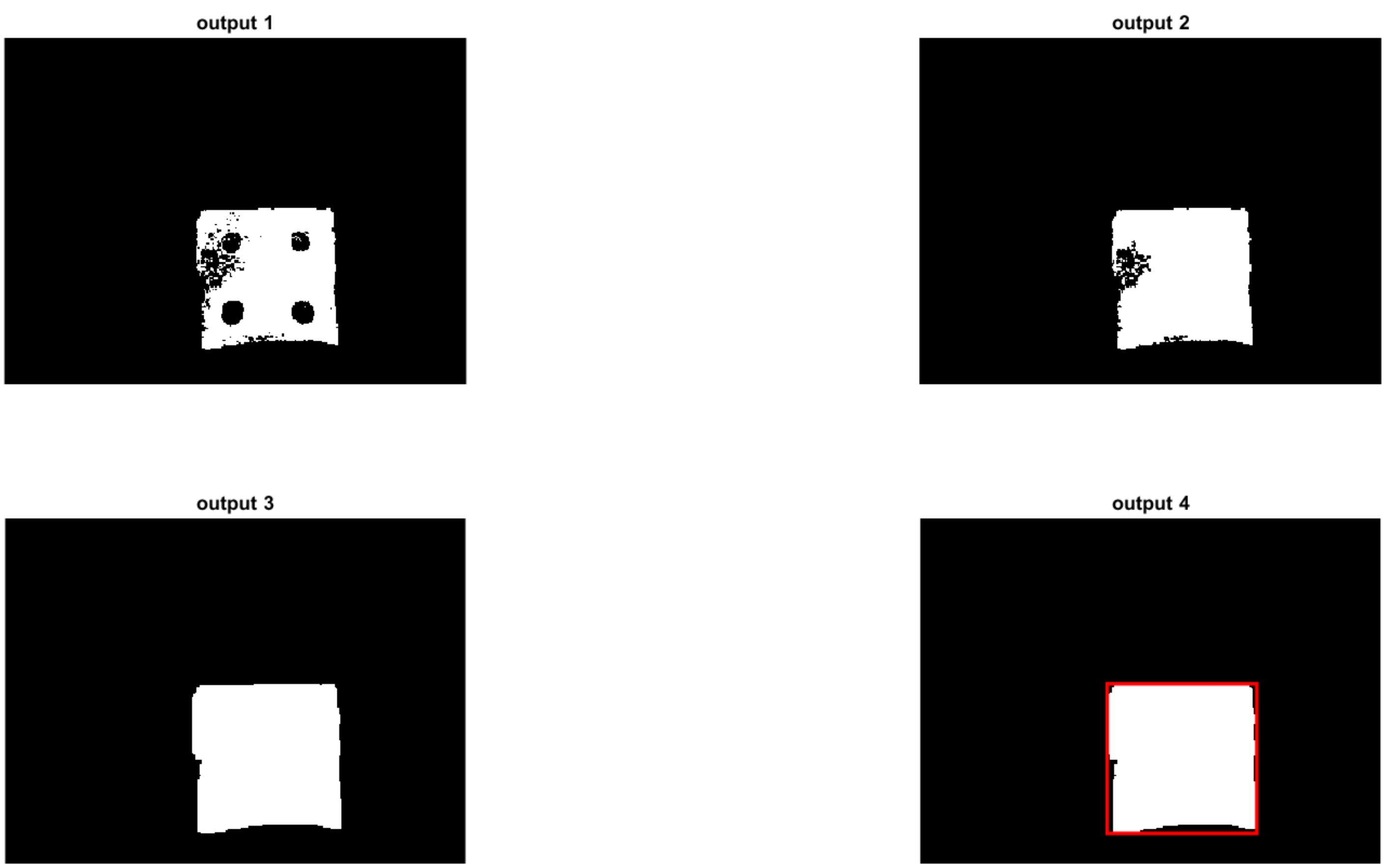
By this step, the resulted thresholding is not precise enough. In the following steps, more improvements will be applied such as filling the holes, and image dilation.

```
out2 = imfill(out1,'holes');
subplot(2,2,2);
imshow(out2);
title('output 2');

out3 = bwmorph(out2,'dilate',3);
subplot(2,2,3);
imshow(out3);
title('output 3');

out4 = imfill(out3,'holes');
subplot(2,2,4);
imshow(out4);
title('output 4');
```

After that, regionprops()is used to get the bounded box and extrema points for detected object as follows:



A comparison has been conducted between using bounding box or extrema points to fit the object shape in the RGB image.

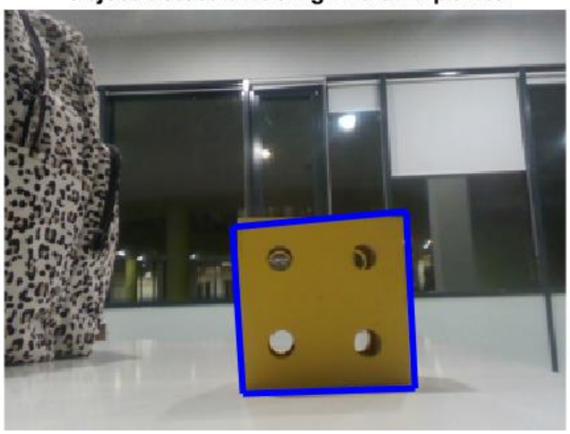
```
figure(4);
subplot(2,1,1)
imshow(I);
title('Object detection using Bounding Box');
hold on;
rectangle('Position', [BB(1),BB(2),BB(3),BB(4)],'EdgeColor','r','LineWidth',2);
hold off;

subplot(2,1,2);
imshow(I);
title('Object detection using Extrema points');
hold on;
for i=[2 4 6]
    line([Ex(i,1),Ex(i+2,1)],[Ex(i,2),Ex(i+2,2)],'color','b','LineWidth',3);
end
line([Ex(8,1),Ex(2,1)],[Ex(8,2),Ex(2,2)],'color','b','LineWidth',3);
hold off;
```

The outpus for this comparision is attached below. We can see that extrema points fit is more precise than bounding box.

Object detection using Bounding Box

Object detection using Extrema points



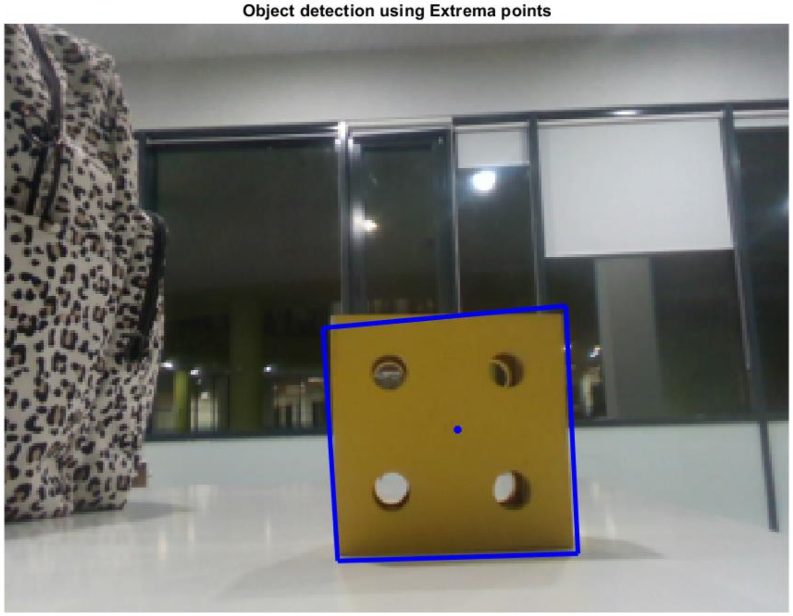
1.4. Object center location based on Extrema points

```
%% 4- locate object center based on Extrema points
x_center = (Ex(4,1)+Ex(6,1))/2;
y_center = (Ex(2,2)+Ex(4,2))/2;

figure(5);
imshow(I);
title('Object detection using Extrema points');
hold on;
for i=[2 4 6]
    line([Ex(i,1),Ex(i+2,1)],[Ex(i,2),Ex(i+2,2)],'color','b','LineWidth',3);
end
line([Ex(8,1),Ex(2,1)],[Ex(8,2),Ex(2,2)],'color','b','LineWidth',3);
plot(x_center,y_center, 'b+', 'MarkerSize', 5, 'LineWidth', 3);
hold off;
```

Largest lengths are selected to obtain the center. The result is attached below.

Object detection using Extrema points



```
Estimated center x: 185.000, y: 166.000
```

1.5. Depth of the estimated center

After getting the center, the attached depth is obtained from the depth map. However, depth map has values from 0 to 255 so, we need to map this range into the camera range which is 0.3 to 3 meter.

```
center_depth = depth_data(y_center,x_center);

% mapping from depth map range [0,255] to camera range [0.3,3] meter
real_depth = ((center_depth*2.7)/255)+0.3;
ground_truth = 0.3;
Error = ((real_depth-ground_truth)/ground_truth)*100;
```

Comparing the estimated depth to the ground truth measured by ruler, the error is about 7% which could be acceptable in this case.

```
Depth estimated from depth map: 0.321 m
Ground truth measured by ruler: 0.300 m
Error: 7.059%
```