Task6: Image Filter & Edge Detection

Name: Saeed Hatefi Ardakani
Degree: PH
ID: 20793159

Problem 1: 2D Convolution (10 points)

The general expression of a 2D convolution is

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u, j-v]$$

where G is the filtered image, F is the original image, and H is the kernel.

You first take a color picture or download any color image on the web and resize the image to have 200 pixel in the shortest side. For example, if your image 5000 x 4000 pixel, it becomes 250x200 pixel. Then, please do the convolution of the resized image and the kernel H:

$$h = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

(a) Please compute G using `conv2`. In the shape option, you set `same`.

The picture I have used for doing convolution is

After doing convolution (using `conv2`), we have

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%---------- Problem 1 ----------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%% (a) compute G using conv2 %%%%%%%%%%
clc;
clear all;
close all;

imgBoardFile = 'fig1.JPG';
imgBoard = imread(imgBoardFile); % read an image
F = imresize(imgBoard, 200/size(imgBoard,1)); % resize the image to have 200
pixel in the shortest side
F = im2double(F);
figure(1);
imshow(F);

imSize = [size(F,1) size(F,2)];

F_CR = F(:,:,1);
F_CG = F(:,:,2);
F_CB = F(:,:,3);

% applying convolution
H = [0 -1 0;-1 4 -1;0 -1 0];
G_CR = conv2(F_CR, H, 'same');
G_CG = conv2(F_CG, H, 'same');
G_CB = conv2(F_CB, H, 'same');

G_CR = uint8(G_CR*255);
G_CG = uint8(G_CG*255);
G_CB = uint8(G_CB*255);

% three dimension matrix
G_imgCRImg = cat(3,G_CR, zeros(imSize, 'uint8'), zeros(imSize, 'uint8')); % how
to use cat
G_imgCGImg = cat(3, zeros(imSize, 'uint8'), G_CG, zeros(imSize, 'uint8'));
G_imgCBImg = cat(3, zeros(imSize, 'uint8'), zeros(imSize, 'uint8'), G_CB);

figure(2);
imshow(G_imgCRImg+G_imgCGImg+G_imgCBImg);
imwrite(G_imgCRImg+G_imgCGImg+G_imgCBImg, 'fig1_a.jpg');
```

(b) Please compute G using your own code (use of a for-loop structure).

Note that answers for (a) and (b) must be the same.

Here, I wrote my own code to compute G. In fact, I wrote `Compute_conv` function for computing convolution.

So, after doing convolution, we have

By comparing the results related to the sections (a) and (b), you can see that the answers are the same.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%---------- Problem 1 ----------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%% (b) compute G using my own code (use of a for-loop structure) %%%%%
clc;
clear all;
close all;

imgBoardFile = 'fig1.JPG';
imgBoard = imread(imgBoardFile); % read an image
F = imresize(imgBoard, 200/size(imgBoard,1)); % resize the image to have 200
pixel in the shortest side
F = im2double(F);
figure(1);
imshow(F);

imSize = [size(F,1) size(F,2)];

F_CR = F(:,:,1);
F_CG = F(:,:,2);
F_CB = F(:,:,3);

% applying convolution
H = [0 -1 0;-1 4 -1;0 -1 0];
G_CR = Compute_conv(F_CR, H);
G_CG = Compute_conv(F_CG, H);
G_CB = Compute_conv(F_CB, H);

G_CR = uint8(G_CR*255);
G_CG = uint8(G_CG*255);
G_CB = uint8(G_CB*255);

% three dimension matrix
G_imgCRImg = cat(3,G_CR, zeros(imSize, 'uint8'), zeros(imSize, 'uint8')); % how
to use cat
```

```
G_imgCGImg = cat(3, zeros(imSize, 'uint8'), G_CG, zeros(imSize, 'uint8'));
G_imgCBImg = cat(3, zeros(imSize, 'uint8'), zeros(imSize, 'uint8'), G_CB);

figure(3);
imshow(G_imgCRImg+G_imgCGImg+G_imgCBImg);
imwrite(G_imgCRImg+G_imgCGImg+G_imgCBImg, 'fig1_b.jpg');



% Convolution function
function G = Compute_conv(F, H)

k = (size(H,1)-1)/2;
G_conv = zeros(size(F,1),size(F,2));
F_conv =
zeros(max([size(F,1)+size(H,1)-1,size(F,1),size(H,1)]),max([size(F,2)+size(H,2)-
1,size(F,2),size(H,2)]));
F_conv(1+k:size(F_conv,1)-k,1+k:size(F_conv,2)-k) = F;

for ii = 1 : size(F,1)
    for jj = 1 :  size(F,2)
        for u = -k : k
            for v = -k : k
                G_conv(ii,jj) = G_conv(ii,jj) + H(u+k+1,v+k+1)*F_conv(ii-u+k,jj-
v+k);
            end
        end
    end
end
G = G_conv;

end
```

Problem 2: Different Kernels (20 points)

You are going to filter your image using the following kernels. Please conduct convolution of your image (used in Problem 1) with the kernel H1, H2, H3, and H4 respectively. Then, you need to explain the effect of filtering with the each kernel.

$$H_1 = \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix},\ H_2 = \begin{bmatrix}0 & 1 & 0\\ 1 & -4 & 1\\ 0 & 1 & 0\end{bmatrix},\ H_3 = \begin{bmatrix}-0.55 & -0.55 & -0.55\\ -0.55 & 5.40 & -0.55\\ -0.55 & -0.55 & -0.55\end{bmatrix},$$

$$H_4 = \begin{bmatrix}0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030\\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133\\ 0.0219 & 0.0983 & 0.1621 & 0.0983 & 0.0219\\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133\\ 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030\end{bmatrix}$$

I used my image from problem 1 to conduct convolution on it:

Using the kernel H1:

Blurring: This is a box blur filter in which each pixel in the resulting image has a value equal to the average value of its neighboring pixels in the input image.

Using the kernel H2:

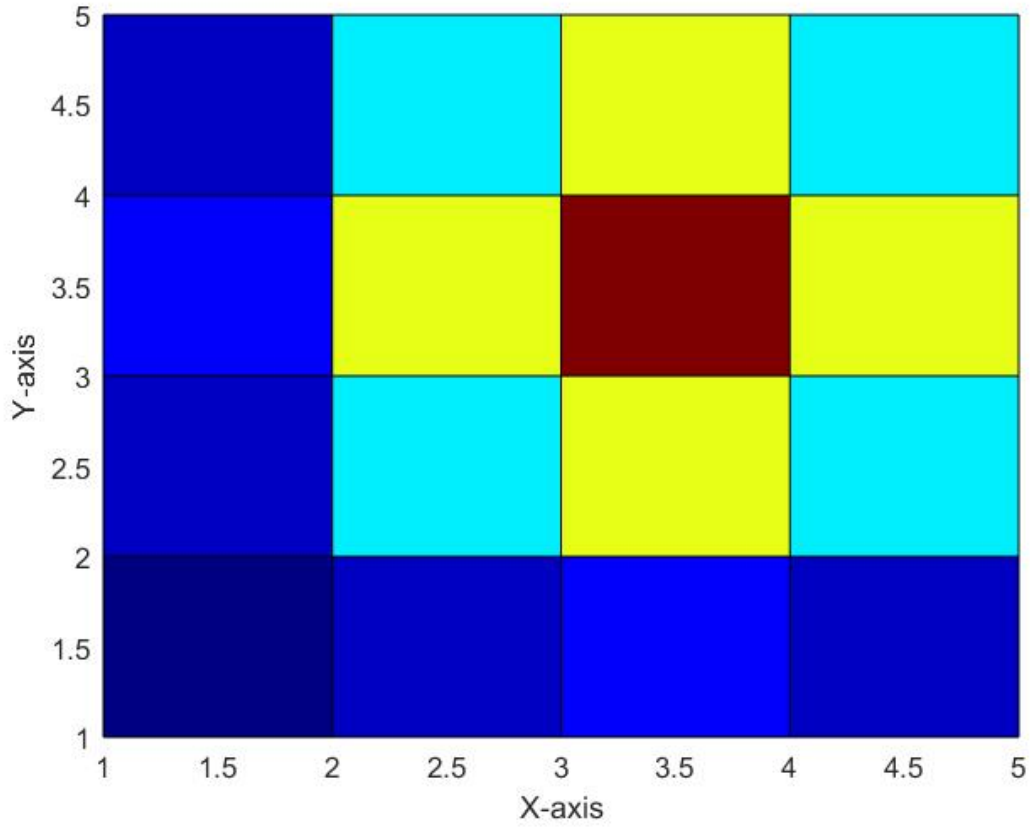
Edge detection: This kernel is the laplacian of gaussian and is often used for edge detection.

Using the kernel H3:

Sharpening: The sharpen kernel emphasizes differences in adjacent pixel values. In fact, sharpening an image increases the contrast between bright and dark regions to bring out features.

Using the kernel H4:

Blurring: this kernel operates like a Gaussian blur (also known as Gaussian smoothing), resulting in blurring an image. It is used to reduce image noise and reduce detail. If we plot this matrix, we have a plot like a Gaussian function:



```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%---------- Problem 2 ----------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
clc;
clear all;
close all;

imgBoardFile = 'fig1.JPG';
imgBoard = imread(imgBoardFile); % read an image
F = imresize(imgBoard, 200/size(imgBoard,1)); % resize the image to have 200
pixel in the shortest side
F = im2double(F);
figure(1);
imshow(F);

imSize = [size(F,1) size(F,2)];

F_CR = F(:,:,1);
F_CG = F(:,:,2);
F_CB = F(:,:,3);

%% applying Filter: H = (1/9)*[1 1 1;1 1 1;1 1 1]
H = (1/9)*[1 1 1;1 1 1;1 1 1];
G_CR = conv2(F_CR, H, 'same');
G_CG = conv2(F_CG, H, 'same');
G_CB = conv2(F_CB, H, 'same');

G_CR = uint8(G_CR*255);
G_CG = uint8(G_CG*255);
G_CB = uint8(G_CB*255);

% three dimension matrix
G_imgCRImg = cat(3,G_CR, zeros(imSize, 'uint8'), zeros(imSize, 'uint8')); % how
to use cat
G_imgCGImg = cat(3, zeros(imSize, 'uint8'), G_CG, zeros(imSize, 'uint8'));
G_imgCBImg = cat(3, zeros(imSize, 'uint8'), zeros(imSize, 'uint8'), G_CB);

figure(2);
imshow(G_imgCRImg+G_imgCGImg+G_imgCBImg);
imwrite(G_imgCRImg+G_imgCGImg+G_imgCBImg, 'fig2_a.jpg');


%% applying Filter: H = [0 1 0;1 -4 1;0 1 0]
H = [0 1 0;1 -4 1;0 1 0];
G_CR = conv2(F_CR, H, 'same');
G_CG = conv2(F_CG, H, 'same');
G_CB = conv2(F_CB, H, 'same');

G_CR = uint8(G_CR*255);
G_CG = uint8(G_CG*255);
G_CB = uint8(G_CB*255);

% three dimension matrix
G_imgCRImg = cat(3,G_CR, zeros(imSize, 'uint8'), zeros(imSize, 'uint8')); % how
to use cat
G_imgCGImg = cat(3, zeros(imSize, 'uint8'), G_CG, zeros(imSize, 'uint8'));
G_imgCBImg = cat(3, zeros(imSize, 'uint8'), zeros(imSize, 'uint8'), G_CB);

figure(3);
imshow(G_imgCRImg+G_imgCGImg+G_imgCBImg);
```

```
imwrite(G_imgCRImg+G_imgCGImg+G_imgCBImg, 'fig2_b.jpg');


%% applying Filter: H = [-0.55 -0.55 -0.55;-0.55 5.4 -0.55;-0.55 -0.55 -0.55]
H = [-0.55 -0.55 -0.55;-0.55 5.4 -0.55;-0.55 -0.55 -0.55];
G_CR = conv2(F_CR, H, 'same');
G_CG = conv2(F_CG, H, 'same');
G_CB = conv2(F_CB, H, 'same');

G_CR = uint8(G_CR*255);
G_CG = uint8(G_CG*255);
G_CB = uint8(G_CB*255);

% three dimension matrix
G_imgCRImg = cat(3,G_CR, zeros(imSize, 'uint8'), zeros(imSize, 'uint8')); % how
to use cat
G_imgCGImg = cat(3, zeros(imSize, 'uint8'), G_CG, zeros(imSize, 'uint8'));
G_imgCBImg = cat(3, zeros(imSize, 'uint8'), zeros(imSize, 'uint8'), G_CB);

figure(4);
imshow(G_imgCRImg+G_imgCGImg+G_imgCBImg);
imwrite(G_imgCRImg+G_imgCGImg+G_imgCBImg, 'fig2_c.jpg');


%% applying Filter: H = [0.003 0.0133 0.0219 0.0133 0.003;
                        % 0.0133 0.0596 0.0983 0.0596 0.0133;
                        % 0.0219 0.0983 0.1621 0.0983 0.0219;
                        % 0.0133 0.0596 0.0983 0.0596 0.0133;
                        %0.003 0.0133 0.0219 0.0133 0.003]
H = [0.003 0.0133 0.0219 0.0133 0.003;
     0.0133 0.0596 0.0983 0.0596 0.0133;
     0.0219 0.0983 0.1621 0.0983 0.0219;
     0.0133 0.0596 0.0983 0.0596 0.0133;
     0.003 0.0133 0.0219 0.0133 0.003];
G_CR = conv2(F_CR, H, 'same');
G_CG = conv2(F_CG, H, 'same');
G_CB = conv2(F_CB, H, 'same');

G_CR = uint8(G_CR*255);
G_CG = uint8(G_CG*255);
G_CB = uint8(G_CB*255);

% three dimension matrix
G_imgCRImg = cat(3,G_CR, zeros(imSize, 'uint8'), zeros(imSize, 'uint8')); % how
to use cat
G_imgCGImg = cat(3, zeros(imSize, 'uint8'), G_CG, zeros(imSize, 'uint8'));
G_imgCBImg = cat(3, zeros(imSize, 'uint8'), zeros(imSize, 'uint8'), G_CB);

figure(5);
imshow(G_imgCRImg+G_imgCGImg+G_imgCBImg);
imwrite(G_imgCRImg+G_imgCGImg+G_imgCBImg, 'fig2_d.jpg');
```

Problem 3: Gaussian Kernel (20 points)

See tutorials!!

(a) Write a code to create the following Gaussian kernel h without using `fspecial`.

```
h = fspecial('gaussian', 5, 2);
```

You need to use theoretical Gaussian curve to get the value of each element in h. Your code produces the kernel close to h obtained from the `fspecial` function.

Please ignore the boundary effect. You can do either zero-padding (`same` option in `conv2` in MATLAB) or get the valid area (`valid` option in `conv2` in MATLAB).

I have written the below code for calculating Gaussian kernel h without using `fspecial`:

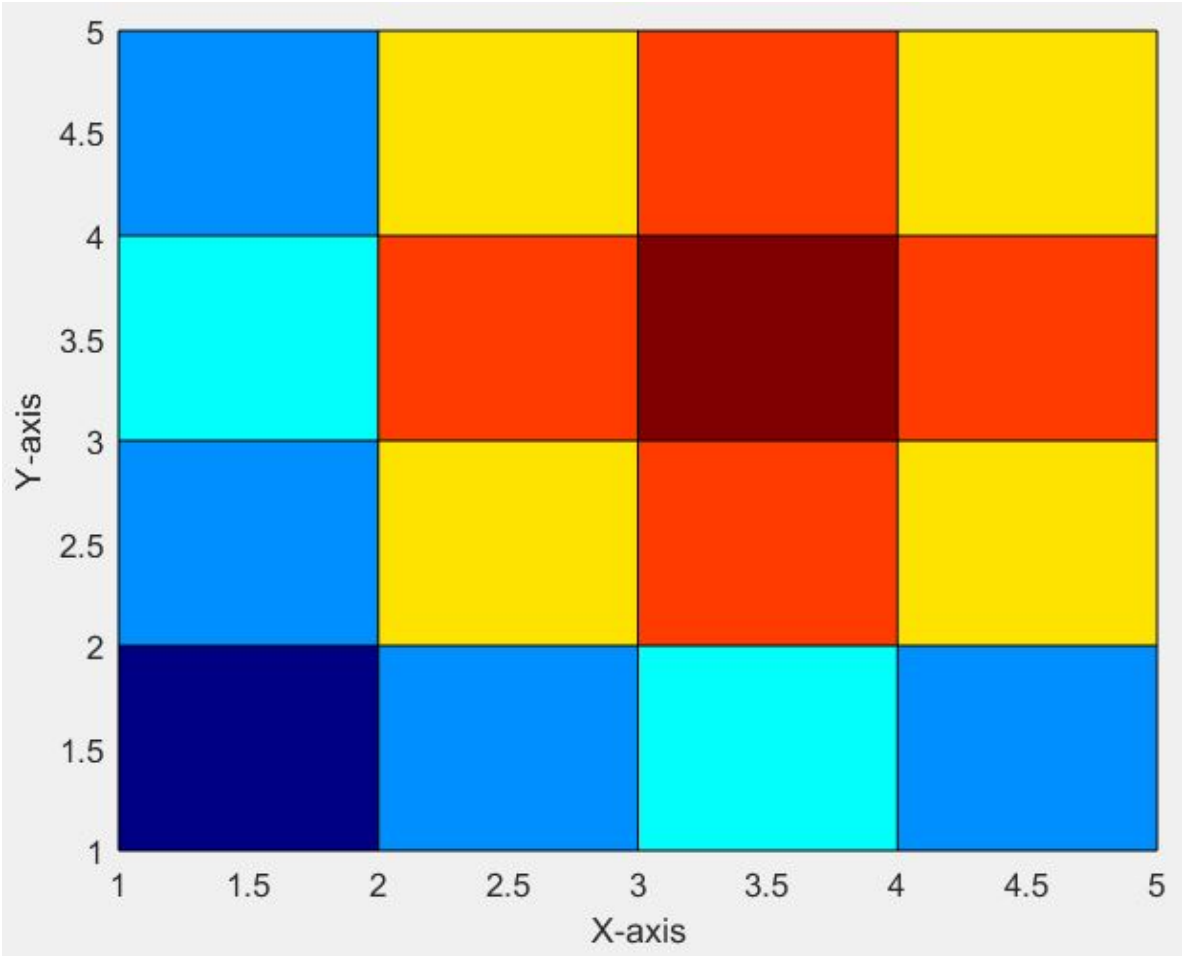
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%---------- Problem 3a ----------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc;
clear all;
close all;

% Calculating Gaussian kernel h without using `fspecial`
numSize = 5;
[x, y] = meshgrid(1:numSize);
x = x - (round(numSize/2));
y = y - (round(numSize/2));
sigma = 2;
G = (1/2/pi/sigma^2)*exp(-(x.^2+y.^2)/2/sigma^2);
G = G/sum(G,'all');
% -------------------------------------------------------

% Calculating Gaussian kernel h by using `fspecial`
h = fspecial('gaussian',5,2);
% -------------------------------------------------------

% plot matrices
figure(1);
PlotMat(G,gca,'float');
title('\bf Calculating Gaussian kernel h without using
`fspecial`','fontsize',13);
figure(2);
PlotMat(h,gca,'float');
title('\bf Calculating Gaussian kernel h by using `fspecial`','fontsize',13);

figure(3)
surf(G);
view(0,90);
axis tight;
colormap(jet);
xlabel('X-axis');
ylabel('Y-axis');


function PlotMat(matrix, axes_handle ,numType)

ax = axes_handle;
```

```
[c,r] = size(matrix); % it should be 2D.
imagesc(ax, matrix);            % Create a colored plot of the matrix values
colormap(ax, white);

hold on;
% horizontal line
for ii=0:c
   line(ax, [0 r], [ii,ii], 'color', 'k',  'linewidth', 1.5);
end

% vertical line
for ii=0:r
   line(ax, [ii ii], [0,c], 'color', 'k', 'linewidth', 1.5);
end

if strcmp(numType, 'integer')
    textStrings = num2str(matrix(:), '%d');       % Create strings from the
matrix values
    textStrings = strtrim(cellstr(textStrings));  % Remove any space padding
else
    textStrings = num2str(matrix(:), '%3.2f');       % Create strings from the
matrix values
    textStrings = strtrim(cellstr(textStrings));  % Remove any space padding
end

[x, y] = meshgrid(1:c, 1:r);
text(ax, x(:)-0.5, y(:)-0.5, textStrings(:), ...  % Plot the strings
                'HorizontalAlignment', 'center', ...
                'FontSize', 10, ...
                'FontWeight', 'bold');
axis(ax,'equal'); hold off;
xlim(ax,[0 r]);
ylim(ax,[0 c]);
set(ax, 'XTick', 1:r, 'YTick', 1:c, ...
    'XTickLabel',[], 'YTickLabel', [])
end
```

The result is

Calculating Gaussian kernel h without using `fspecial`

0.02	0.03	0.04	0.03	0.02
0.03	0.05	0.06	0.05	0.03
0.04	0.06	0.06	0.06	0.04
0.03	0.05	0.06	0.05	0.03
0.02	0.03	0.04	0.03	0.02



In addition, if we calculate Gaussian kernel h with `fspecial` function, we have the below matrix which is the same with the calculated kernel without using `fspecial`:

Calculating Gaussian kernel h by using `fspecial`

0.02	0.03	0.04	0.03	0.02
0.03	0.05	0.06	0.05	0.03
0.04	0.06	0.06	0.06	0.04
0.03	0.05	0.06	0.05	0.03
0.02	0.03	0.04	0.03	0.02

(b) What is the difference between the kernel h1 and h2 in terms of the effect on the filtered images with these kernels? You can explain it with sample results.

```
h1 = fspecial('gaussian', 20, 2);
h2 = fspecial('gaussian', 20, 3);
```

The difference between the kernel h1 and h2 is the amount of sigma. The role of sigma (standard deviation) in the Gaussian filter is to control the variation around its mean value. If you increase standard deviation in normal distribution, the distribution will be more spread out, and the peak will be less spiky. Thus, if you increase the variance, it makes the image more blurry.

I used these two kernels for the image related to the problem 1. As you can see, by increasing the amount of sigma, the image would be more blurry.

Applying Gaussian filter h1:

Applying Gaussian filter h2:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%---------- Problem 3b ----------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc;
clear all;
close all;

imgBoardFile = 'fig1.JPG';
imgBoard = imread(imgBoardFile); % read an image
F = im2double(imgBoard);
figure(1);
imshow(F);

imSize = [size(F,1) size(F,2)];

F_CR = F(:,:,1);
F_CG = F(:,:,2);
F_CB = F(:,:,3);
```

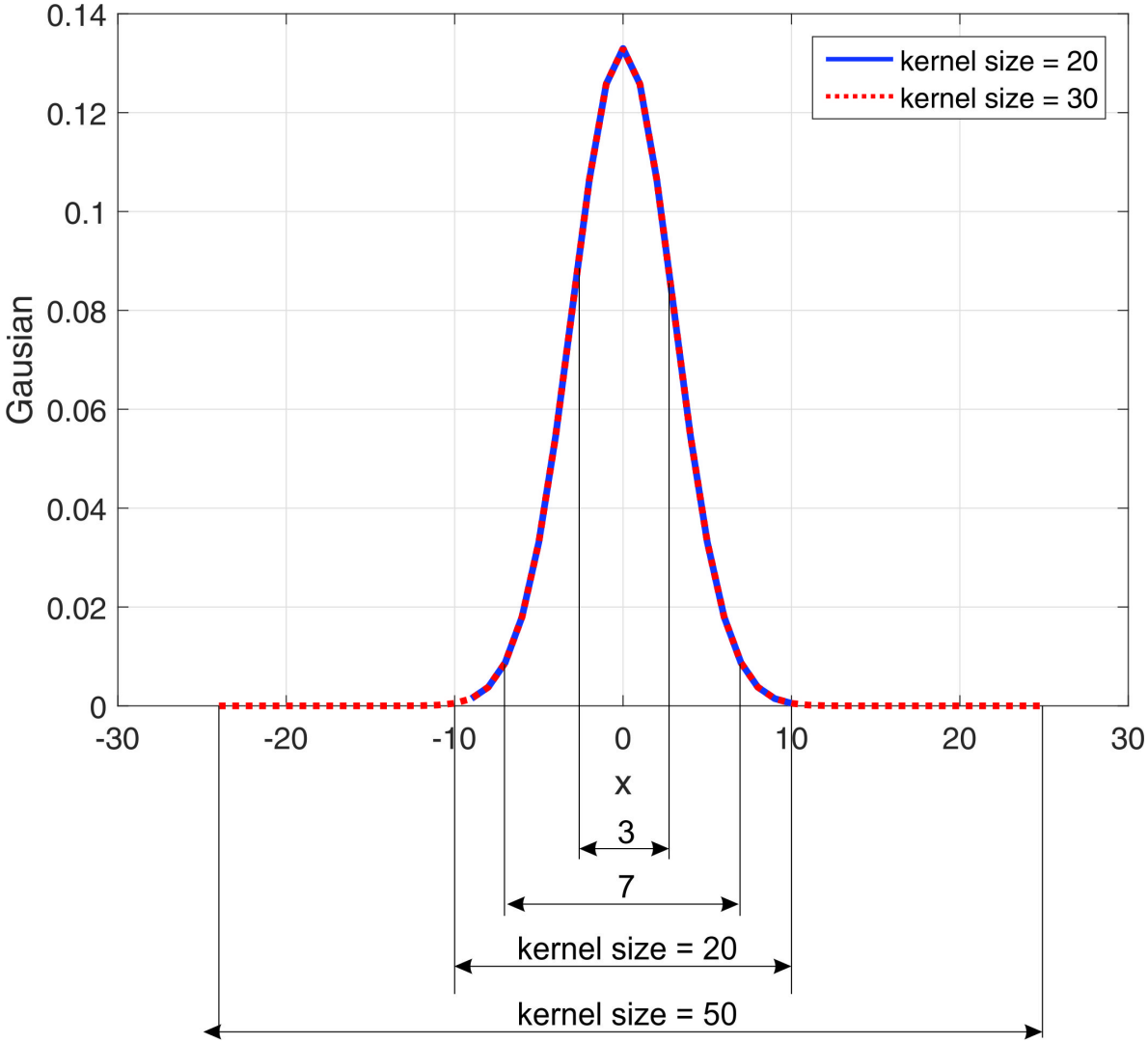
```
%defining kernels: the difference between these two kernels is the amount of
sigma
h1 = fspecial('gaussian',20,2);
h2 = fspecial('gaussian',20,3);

% applying the kernel 1
G_CR1 = conv2(F_CR, h1, 'same');
G_CG1 = conv2(F_CG, h1, 'same');
G_CB1 = conv2(F_CB, h1, 'same');

G_CR1 = uint8(G_CR1*255);
G_CG1 = uint8(G_CG1*255);
G_CB1 = uint8(G_CB1*255);

% three dimension matrix
G_imgCRImg1 = cat(3,G_CR1, zeros(imSize, 'uint8'), zeros(imSize, 'uint8')); %
how to use cat
G_imgCGImg1 = cat(3, zeros(imSize, 'uint8'), G_CG1, zeros(imSize, 'uint8'));
G_imgCBImg1 = cat(3, zeros(imSize, 'uint8'), zeros(imSize, 'uint8'), G_CB1);

figure(2);
imshow(G_imgCRImg1+G_imgCGImg1+G_imgCBImg1);
imwrite(G_imgCRImg1+G_imgCGImg1+G_imgCBImg1, 'fig3b1.jpg');


% applying the kernel 2
G_CR2 = conv2(F_CR, h2, 'same');
G_CG2 = conv2(F_CG, h2, 'same');
G_CB2 = conv2(F_CB, h2, 'same');

G_CR2 = uint8(G_CR2*255);
G_CG2 = uint8(G_CG2*255);
G_CB2 = uint8(G_CB2*255);

% three dimension matrix
G_imgCRImg2 = cat(3,G_CR2, zeros(imSize, 'uint8'), zeros(imSize, 'uint8')); %
how to use cat
G_imgCGImg2 = cat(3, zeros(imSize, 'uint8'), G_CG2, zeros(imSize, 'uint8'));
G_imgCBImg2 = cat(3, zeros(imSize, 'uint8'), zeros(imSize, 'uint8'), G_CB2);

figure(3);
imshow(G_imgCRImg2+G_imgCGImg2+G_imgCBImg2);
imwrite(G_imgCRImg2+G_imgCGImg2+G_imgCBImg2, 'fig3b2.jpg');
```

(c) What is the difference between the kernel h2 and h3 in terms of filtered images with these kernels? Are they different? You can explain it with sample results.

```
h2 = fspecial('gaussian', 20, 3);
h3 = fspecial('gaussian', 50, 3);
```

The difference between the kernel h2 and h3 is the amount of kernel size. Kernel size means filter size. In general, larger kernels would remove more noise from the image. But they will also mean more undesirable artifacts as well.

Here, although the results would be blurry, we cannot see obvious difference between the results related to these two kernels. The reason is that the size of kernels is so large in these kernels, leading to the same behavior as a filter. I have plotted these two kernels in the below figure. As you can see, there is no difference between these two filters.



I used these two kernels for the image related to the problem 1. As you can see, the results would be blurry, but there is no difference between their results.

Applying Gaussian filter h2:

Applying Gaussian filter h3:

note!!: It should be mentioned that, if we use different kernel size in low values like 3 and 7, we can see an obvious different between the results:

Applying Gaussian filter with the kernel size equal to 3 (fspecial('gaussian', 3, 3)):

Applying Gaussian filter with the kernel size equal to 7 (fspecial('gaussian', 7, 3)):

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%---------- Problem 3c ----------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc;
clear all;
close all;

imgBoardFile = 'fig1.JPG';
imgBoard = imread(imgBoardFile); % read an image
F = im2double(imgBoard);
figure(1);
imshow(F);

imSize = [size(F,1) size(F,2)];

F_CR = F(:,:,1);
F_CG = F(:,:,2);
F_CB = F(:,:,3);


%defining kernels: the difference between these two kernels is the amount
%of kernel size
h2 = fspecial('gaussian',20,3);
h3 = fspecial('gaussian',50,3);

% applying the kernel 2
G_CR2 = conv2(F_CR, h2, 'same');
G_CG2 = conv2(F_CG, h2, 'same');
G_CB2 = conv2(F_CB, h2, 'same');

G_CR2 = uint8(G_CR2*255);
G_CG2 = uint8(G_CG2*255);
G_CB2 = uint8(G_CB2*255);

% three dimension matrix
G_imgCRImg2 = cat(3,G_CR2, zeros(imSize, 'uint8'), zeros(imSize, 'uint8')); %
how to use cat
G_imgCGImg2 = cat(3, zeros(imSize, 'uint8'), G_CG2, zeros(imSize, 'uint8'));
```

```
G_imgCBImg2 = cat(3, zeros(imSize, 'uint8'), zeros(imSize, 'uint8'), G_CB2);

figure(2);
imshow(G_imgCRImg2+G_imgCGImg2+G_imgCBImg2);
imwrite(G_imgCRImg2+G_imgCGImg2+G_imgCBImg2, 'fig3c1.jpg');


% applying the kernel 3
G_CR3 = conv2(F_CR, h3, 'same');
G_CG3 = conv2(F_CG, h3, 'same');
G_CB3 = conv2(F_CB, h3, 'same');

G_CR3 = uint8(G_CR3*255);
G_CG3 = uint8(G_CG3*255);
G_CB3 = uint8(G_CB3*255);

% three dimension matrix
G_imgCRImg3 = cat(3,G_CR3, zeros(imSize, 'uint8'), zeros(imSize, 'uint8')); %
how to use cat
G_imgCGImg3 = cat(3, zeros(imSize, 'uint8'), G_CG3, zeros(imSize, 'uint8'));
G_imgCBImg3 = cat(3, zeros(imSize, 'uint8'), zeros(imSize, 'uint8'), G_CB3);

figure(3);
imshow(G_imgCRImg3+G_imgCGImg3+G_imgCBImg3);
imwrite(G_imgCRImg3+G_imgCGImg3+G_imgCBImg3, 'fig3c2.jpg');
```

(d) What is the difference between the kernel h2 and h4 in terms of filtered images with these kernels? Are they different? You can explain it with sample results.

```
h2 = fspecial('gaussian', 20, 3);
h4 = fspecial('gaussian', 20, 20);
```

The difference between the kernel h2 and h4 is the amount of sigma. The role of sigma (standard deviation) in the Gaussian filter is to control the variation around its mean value. If you increase standard deviation in normal distribution, the distribution will be more spread out, and the peak will be less spiky. Thus, if you increase the variance, it makes the image more blurry. In other word, larger sigma would remove more noise ad detail from the image, but they will produce more undesirable artifacts as well. So, we will miss more details in large values of sigma.

I used these two kernels for the image related to the problem 1. As you can see, by increasing the amount of sigma, the image would be more blurry. Also, in large value of sigma (sigma=20), we will miss most of details about the image, and more undesirable artifacts will be produced.

Applying Gaussian filter h2:

Applying Gaussian filter h4:

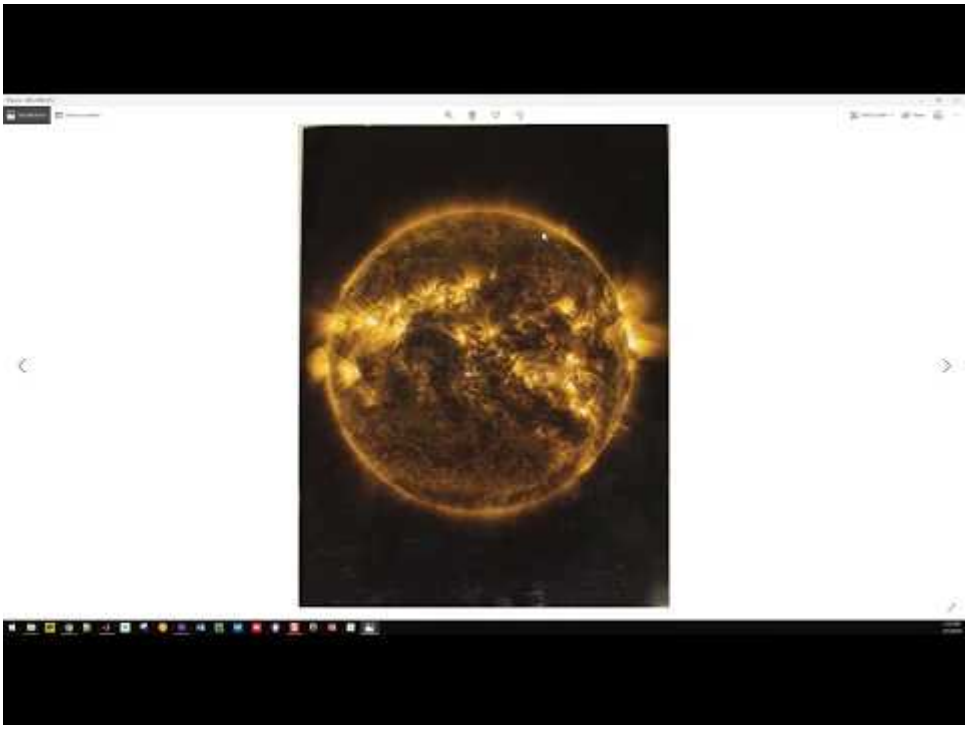
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%---------- Problem 3d ----------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc;
clear all;
close all;

imgBoardFile = 'fig1.JPG';
imgBoard = imread(imgBoardFile); % read an image
F = im2double(imgBoard);
figure(1);
imshow(F);

imSize = [size(F,1) size(F,2)];

F_CR = F(:,:,1);
F_CG = F(:,:,2);
F_CB = F(:,:,3);
```

```
%defining kernels: the difference between these two kernels is the amount
%of kernel size
h2 = fspecial('gaussian',20,3);
h4 = fspecial('gaussian',20,20);

% applying the kernel 2
G_CR2 = conv2(F_CR, h2, 'same');
G_CG2 = conv2(F_CG, h2, 'same');
G_CB2 = conv2(F_CB, h2, 'same');

G_CR2 = uint8(G_CR2*255);
G_CG2 = uint8(G_CG2*255);
G_CB2 = uint8(G_CB2*255);

% three dimension matrix
G_imgCRImg2 = cat(3,G_CR2, zeros(imSize, 'uint8'), zeros(imSize, 'uint8')); %
how to use cat
G_imgCGImg2 = cat(3, zeros(imSize, 'uint8'), G_CG2, zeros(imSize, 'uint8'));
G_imgCBImg2 = cat(3, zeros(imSize, 'uint8'), zeros(imSize, 'uint8'), G_CB2);

figure(2);
imshow(G_imgCRImg2+G_imgCGImg2+G_imgCBImg2);
imwrite(G_imgCRImg2+G_imgCGImg2+G_imgCBImg2, 'fig3d1.jpg');


% applying the kernel 4
G_CR4 = conv2(F_CR, h4, 'same');
G_CG4 = conv2(F_CG, h4, 'same');
G_CB4 = conv2(F_CB, h4, 'same');

G_CR4 = uint8(G_CR4*255);
G_CG4 = uint8(G_CG4*255);
G_CB4 = uint8(G_CB4*255);

% three dimension matrix
G_imgCRImg4 = cat(3,G_CR4, zeros(imSize, 'uint8'), zeros(imSize, 'uint8')); %
how to use cat
G_imgCGImg4 = cat(3, zeros(imSize, 'uint8'), G_CG4, zeros(imSize, 'uint8'));
G_imgCBImg4 = cat(3, zeros(imSize, 'uint8'), zeros(imSize, 'uint8'), G_CB4);

figure(3);
imshow(G_imgCRImg4+G_imgCGImg4+G_imgCBImg4);
imwrite(G_imgCRImg4+G_imgCGImg4+G_imgCBImg4, 'fig3d2.jpg');
```

Problem 4: Hough Transform (25 points)

This problem is to write your own program that extracts a booklet image from each image provided and remove its projective distortion in an automated way.

Here is a sample demo.



A sample code is provided. You need to write your own code for extracting four corners of the booklet using edge detection and hough transform. In other word, you need to write your own `FindCorner` function. In addition, you need to reuse your `ComputeH`, which was used in Task 3. You only remove perspective distortion, meaning that resulting images may be holizontally or vertically flipped.

Hint: I used functions of `edge`, `imgaussfilt`, `houghpeaks`, `hough`, `cross`, `norm` in MATLAB to write the `FindCorner`. However, you can use any functions in OpenCV or MATLAB. You can use `fitgeotrans` in MATLAB and `getPerspectiveTransform` in Python.

I used two different automated methods to extract a booklet image from all the images provided.

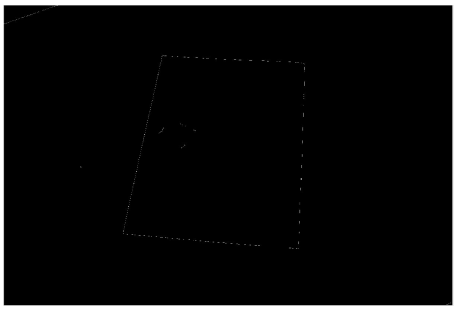
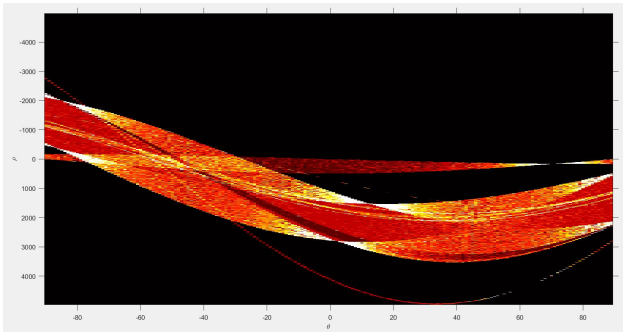
Method 1: First, I resized img files to 0.8*img by the function "imresize(img,0.8)". Then, by using edge detector function and proper threshold "edge(rgb2gray(img), 'Canny',[0.1 0.5])", I found the edges. It should be mentioned that the code works correctly for all five images in an automated way.

Method 2: In this method, first, I cropped the images by function "imcrop(img,[0.15*size(img,2) 0 0.85*size(img,2) size(img,1)])" to remove the parts related to the edges of the table. Then, I used Gaussian filter and Canny edge detector functions to find the edges related to the booklet. Also, it should be mentioned that the code works correctly for all five images in an automated way.

In the below part, I have presented the results associated to two methods. As you can see, the results of these two methods are the same.

Results for method 1:

Image "IMG_0080.JPG":

edge detection	1st peak (Hough Transform)
	

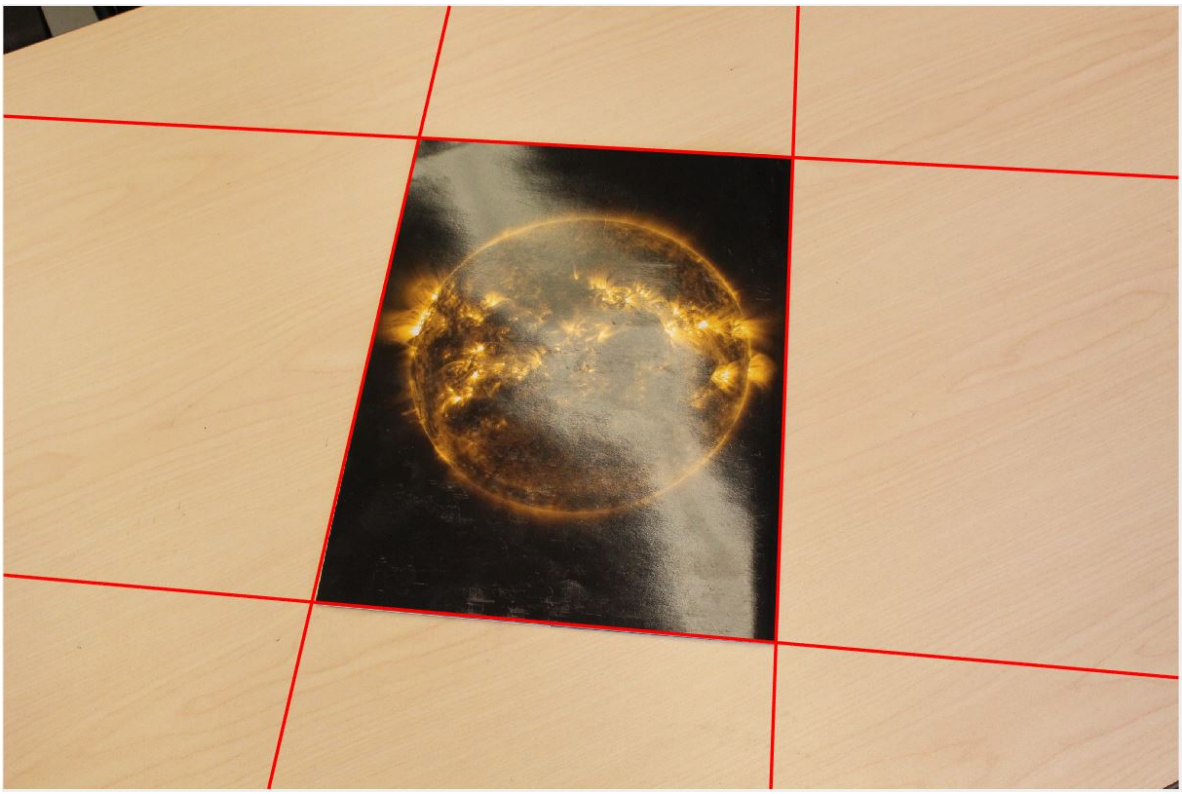
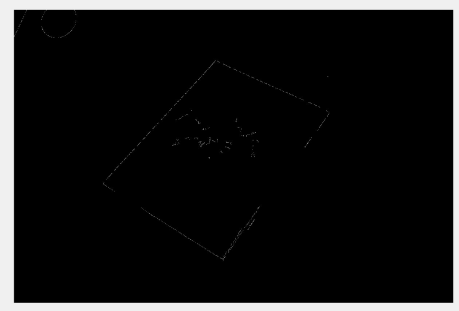
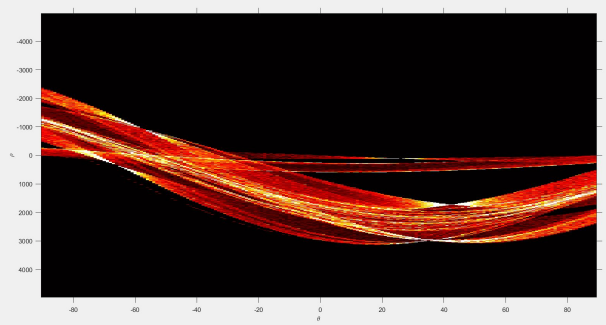
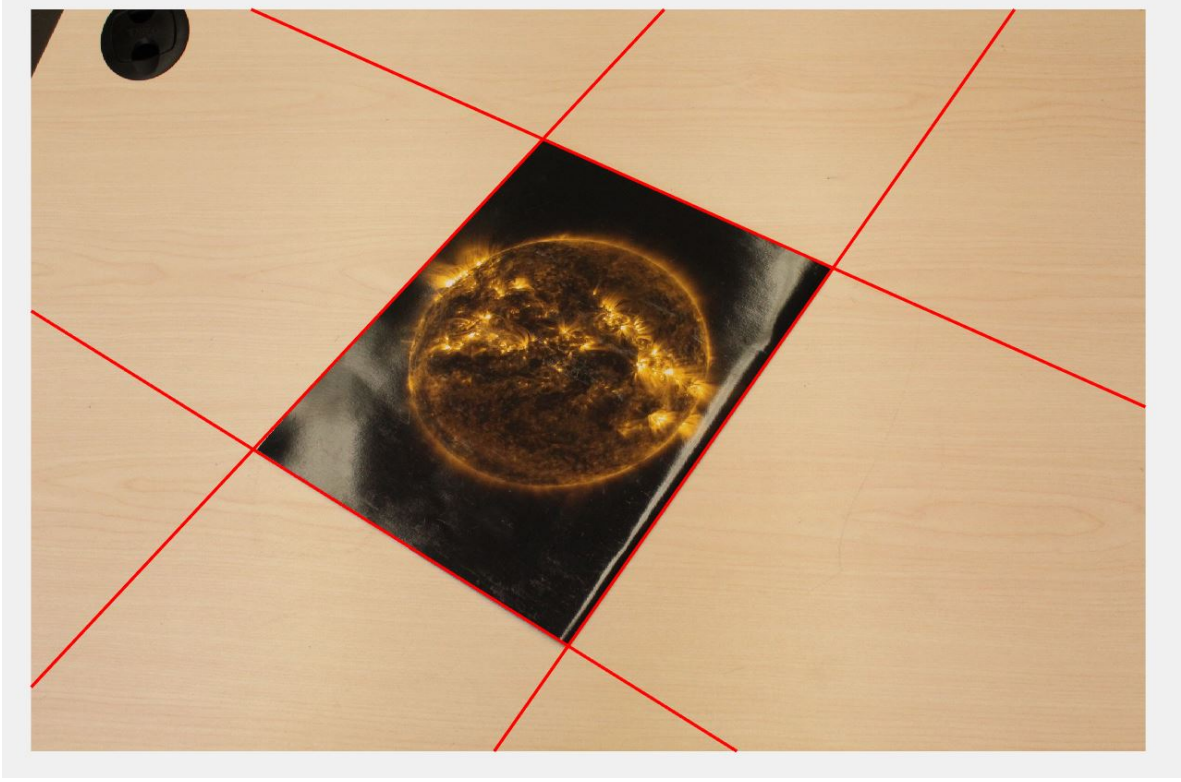


Image "IMG_0081.JPG":

edge detection	1st peak (Hough Transform)
	



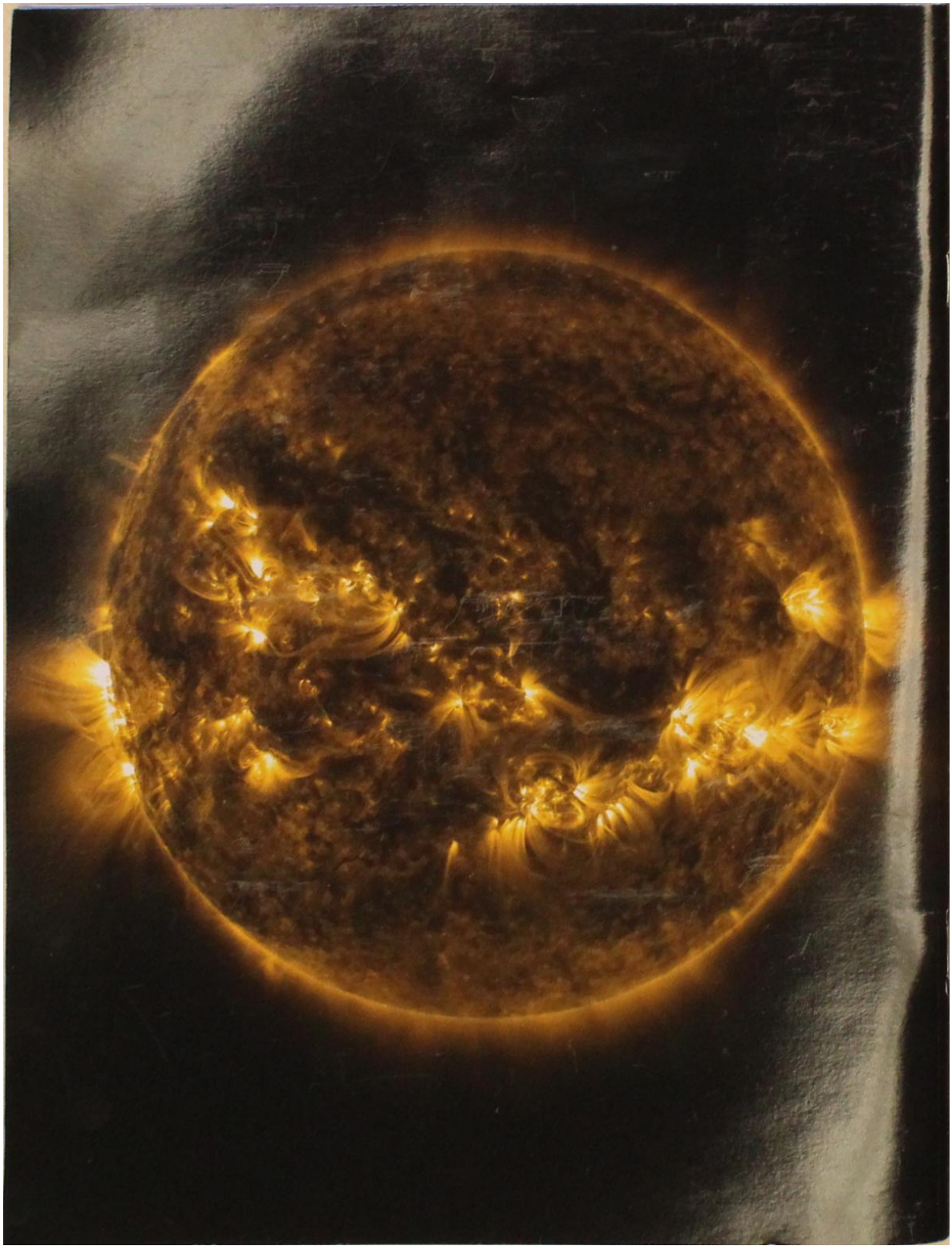
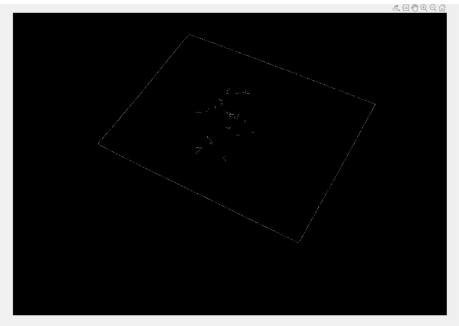
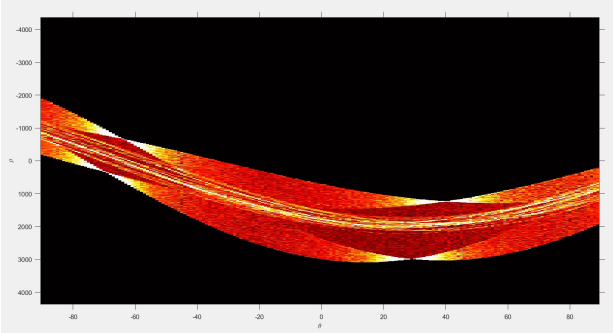


Image "IMG_0082.JPG":

edge detection	1st peak (Hough Transform)
	

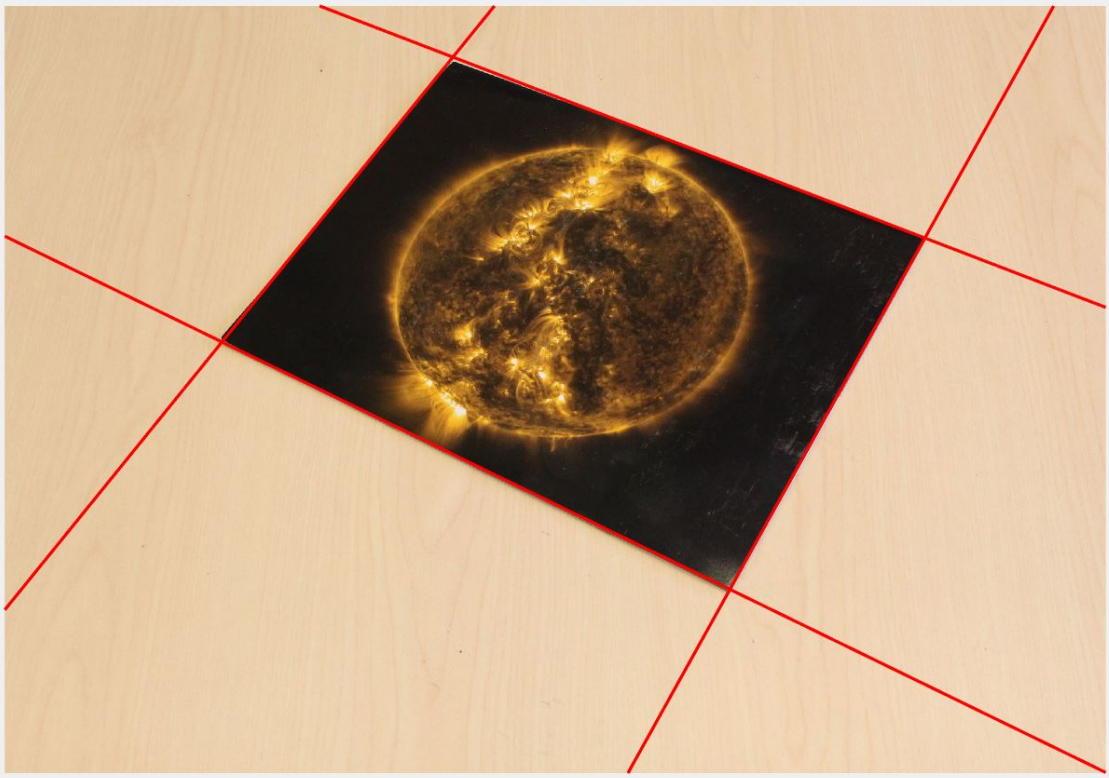
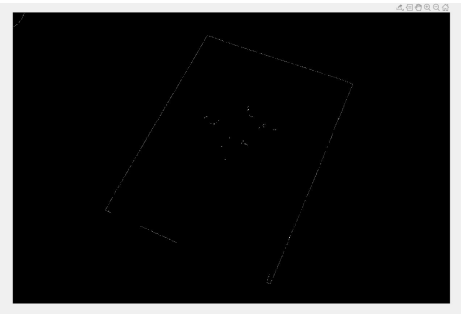
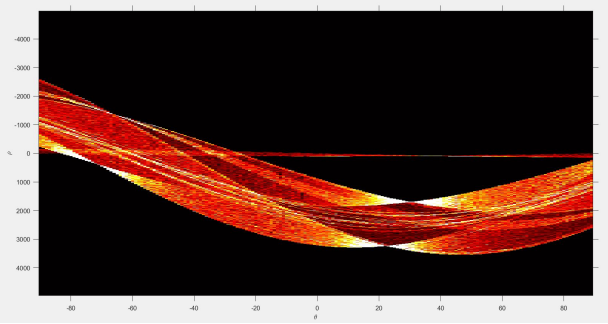
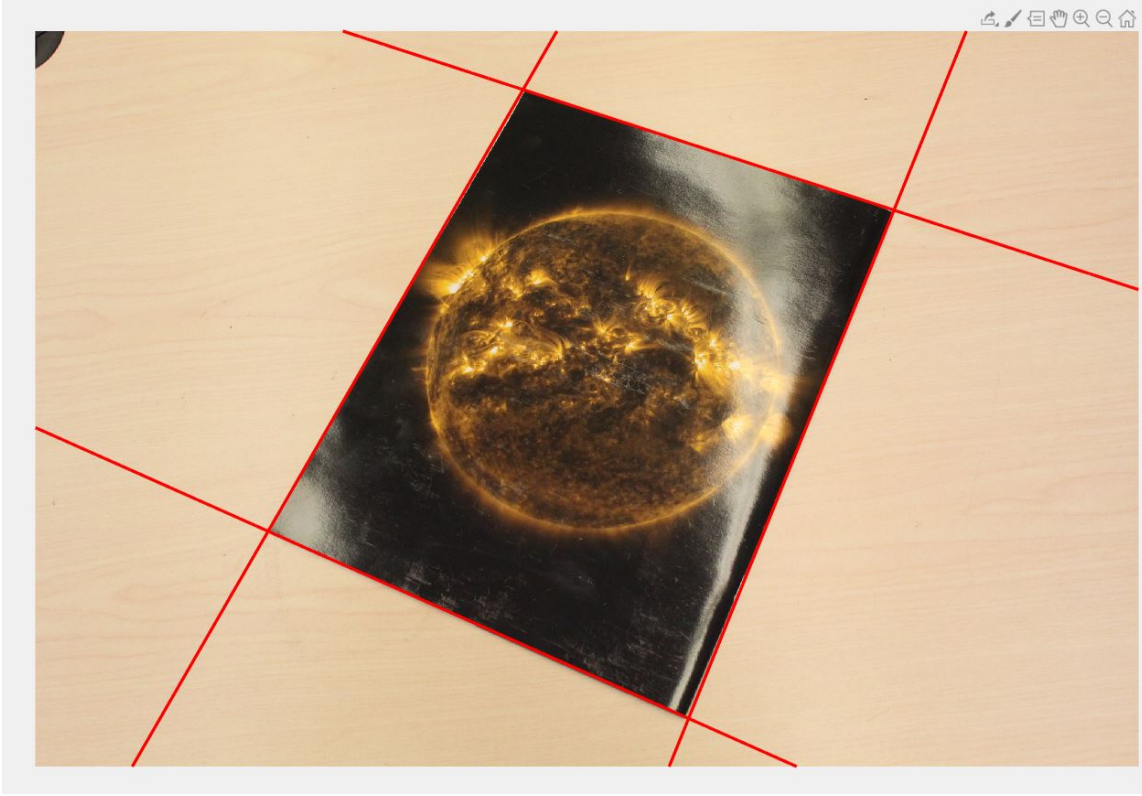


Image "IMG_0084.JPG":

edge detection	1st peak (Hough Transform)
	



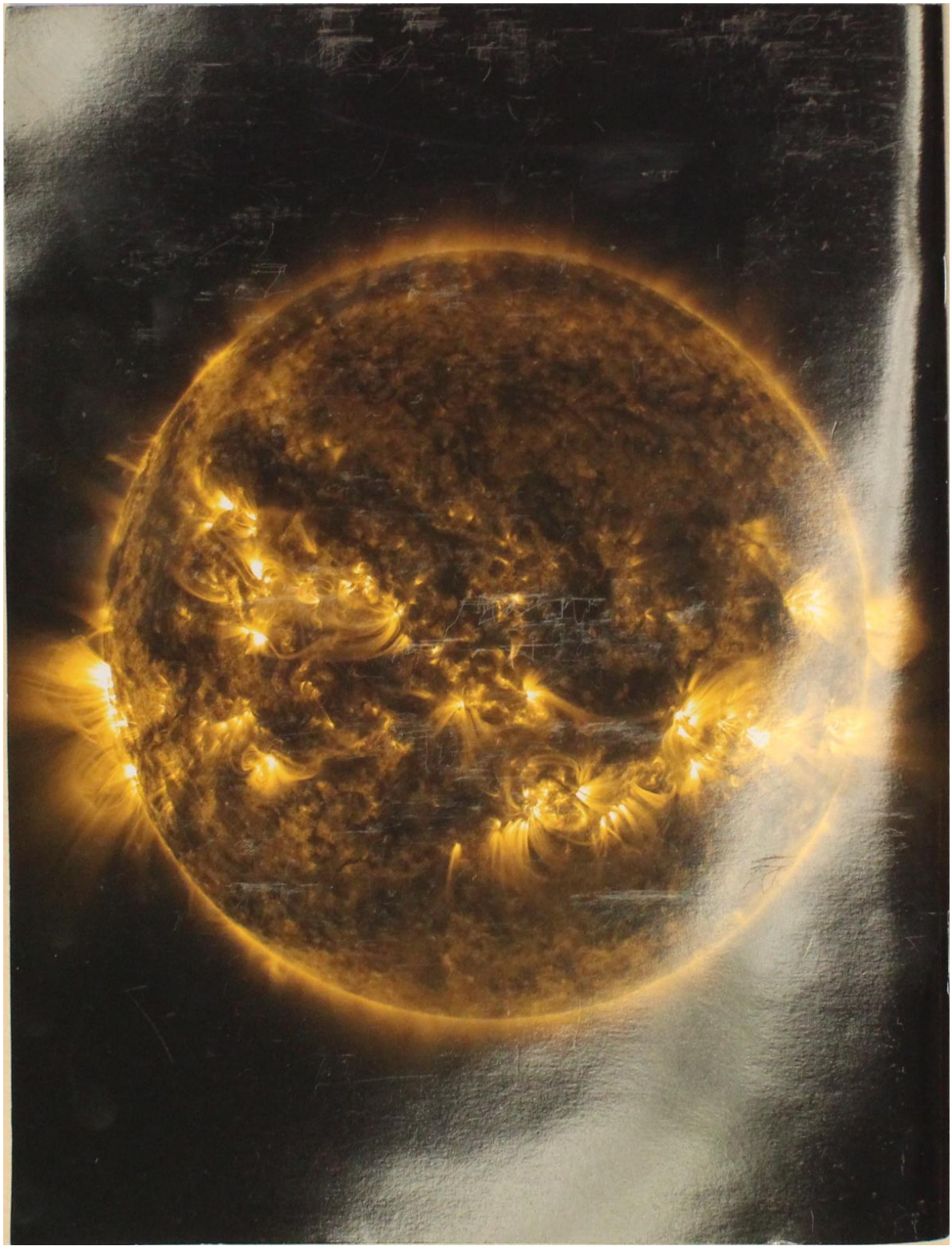
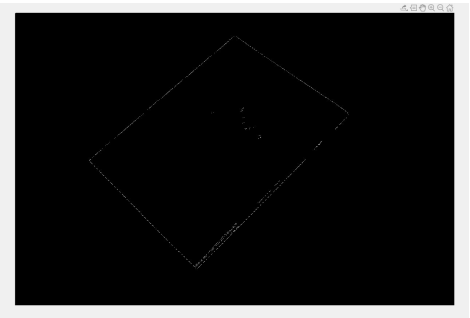
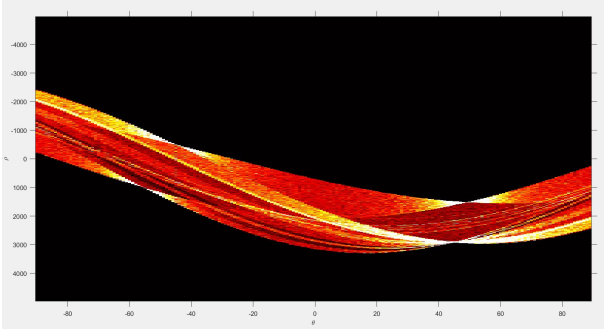
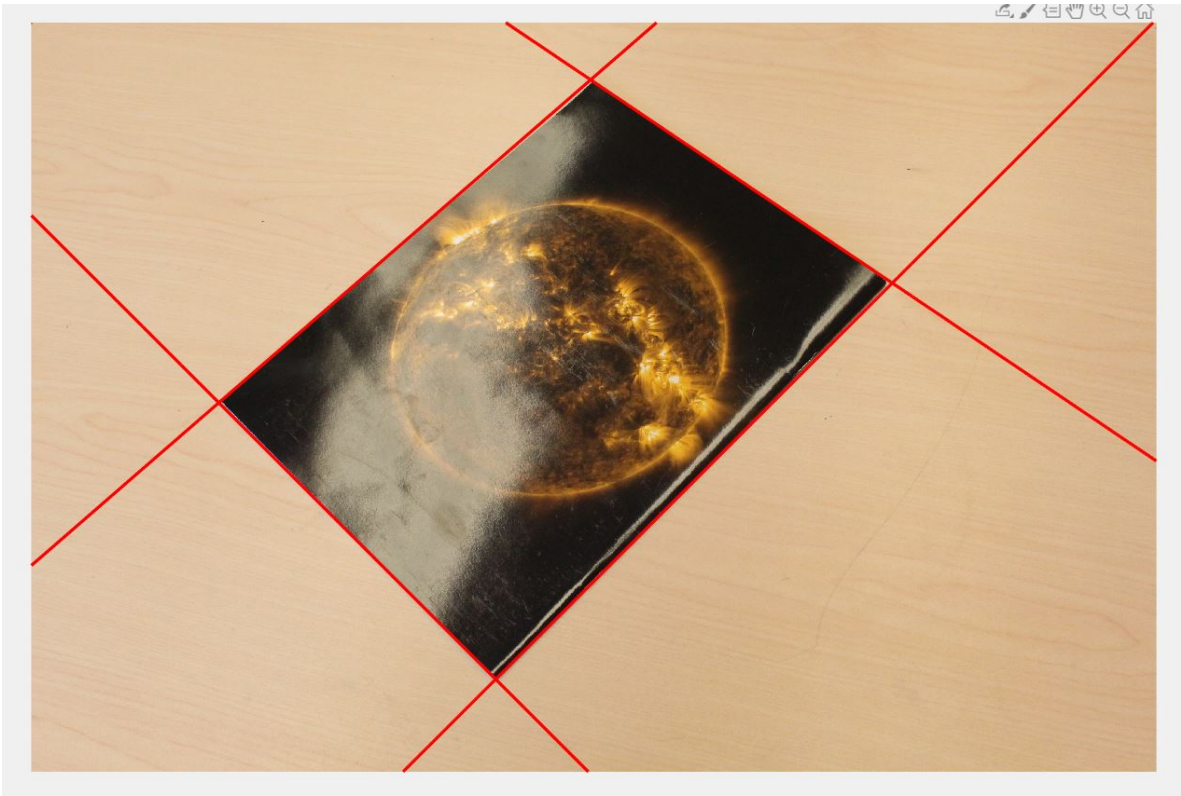
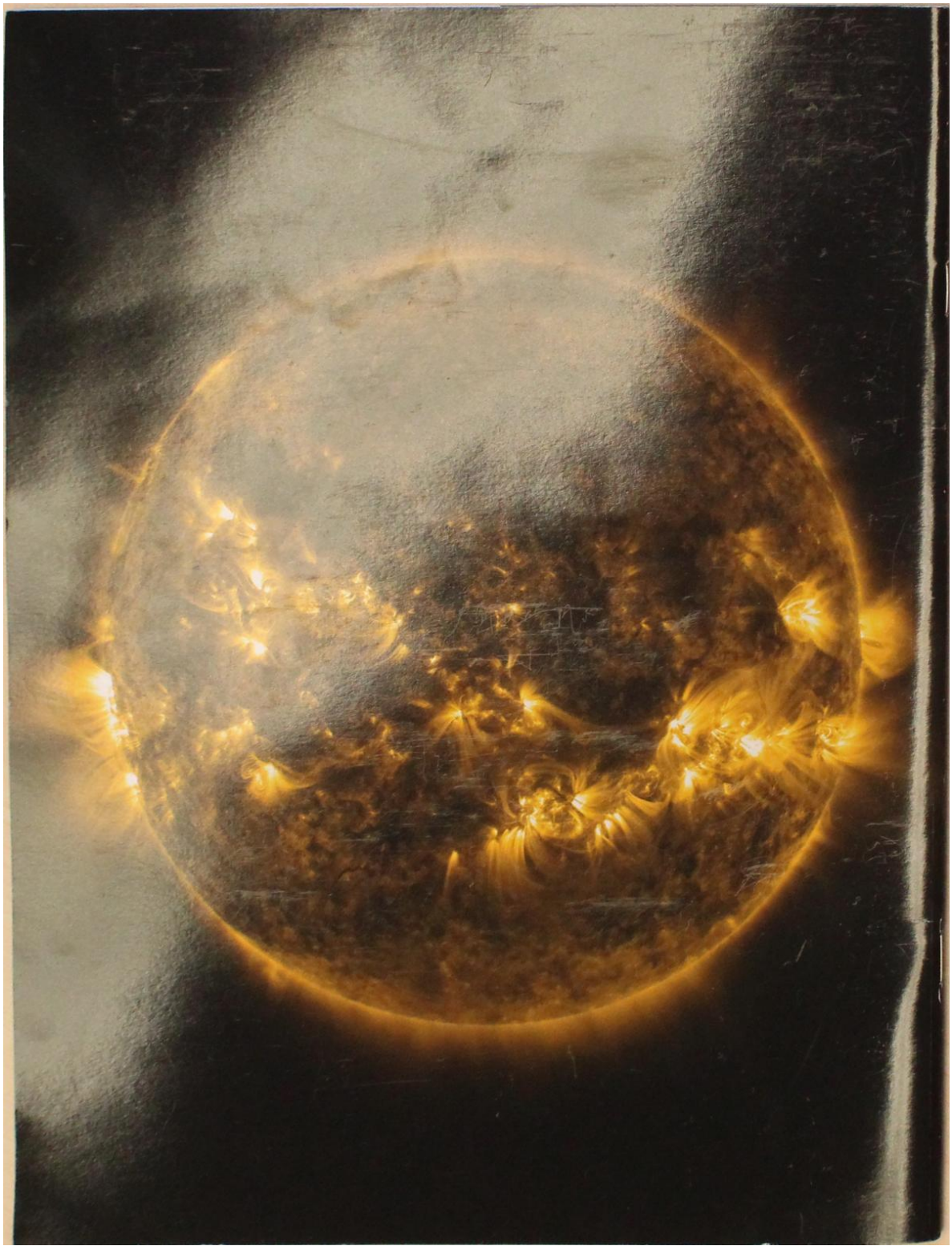


Image "IMG_0085.JPG":

edge detection	1st peak (Hough Transform)
	





Matlab code for method 1:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%---------- Problem 4 - Method 1 ---------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Hough Transformation
% Author: Chul Min Yeum (cmyeum@uwaterloo.ca)
% Last update:: 03/07/2020 by Saeed Hatefi Ardakani
clear; close all; clc; format shortG;

%% Parameter
```

```
bookletSize = [24 31.5]; % cm
bookletImgSize = bookletSize*50; % output image size
dirImg = 'img'; % image folder
dirOut = 'out'; % output image folder
imgList = dir('img/*.JPG');
nImg = numel(imgList);

%% Processing
for ii=1:nImg
    img = imread(fullfile(dirImg, imgList(ii).name));
    img = imresize(img,0.8); % resize the image

    corner = FindCorner(img); % find ordered four corners

    H = ComputeH(bookletImgSize, corner); % compute a homography

    [imgTran, RA] = imwarp(img, projective2d(inv(H)));
    bookletImg = imcrop(imgTran, [-RA.XWorldLimits(1), -RA.YWorldLimits(1)
bookletImgSize]);
    imwrite(bookletImg, fullfile(dirOut,imgList(ii).name));
end


%%
% Homography function
function H = ComputeH(sizePic, corner)
points1 = [1,1;sizePic(1),1;sizePic(1),sizePic(2);1,sizePic(2)];
points2 = corner;
x_xp = points1(:,1).*points2(:,1);
x_yp = points1(:,1).*points2(:,2);
y_xp = points1(:,2).*points2(:,1);
y_yp = points1(:,2).*points2(:,2);

A = zeros(size(points1,1)*2,9);
A(1:2:end,3) = 1;
A(2:2:end,6) = 1;
A(1:2:end,1:2) = points1;
A(2:2:end,4:5) = points1;
A(1:2:end,7) = -x_xp;
A(1:2:end,8) = -y_xp;
A(2:2:end,7) = -x_yp;
A(2:2:end,8) = -y_yp;
A(1:2:end,9) = -points2(:,1);
A(2:2:end,9) = -points2(:,2);

T = null(A);
T = T/T(end);
H = (reshape(T,3,3));
end


%%
% function for finding ordered four corners of the booklet
function corner = FindCorner(img)
imshow(img);

% use Canny edge detector
BW_filt = edge(rgb2gray(img), 'Canny',[0.1 0.5]);
```

```
fig1 = figure(1);
imshow(BW_filt);
set(fig1,'Position', [100 100 800 600]);

% computing the Standard Hough Transform (SHT) of the binary image BW.
% The hough function is designed to detect lines.
[H,T,R] = hough(BW_filt);
figure(2);
imshow(imadjust(rescale(H)),
[],'XData',T,'YData',R,'InitialMagnification','fit');
xlabel('\theta'), ylabel('\rho');
axis on, axis normal, colormap(gca,hot);

% finding peaks in the Hough transform matrix, H, generated by the hough
function.
p = houghpeaks(H,4,'threshold',ceil(0.2*max(H(:))));

% finding ro-->d and theta related to each line
d = R(p(:,1))';
theta = T(p(:,2))';

% ------------------------------------------------------
% finding four ordered points that form a quadrangle from intersecting 4 lines
% ------------------------------------------------------
% writing homogenuous form of the four lines
for ii = 1 : size(d,1)
    m = -cosd(theta(ii))/sind(theta(ii));
    c = d(ii)/sind(theta(ii));
    line(:,ii) = [m -1 c];
end

% finding four corners using my own function (I wrote this function in task 5)
corner = search_corner(line);

% reordering corners in clockwise pattern:
%        4--------3
%        |        |
%        |        |
%        |        |
%        |        |
%        1--------2
mean_point = mean(corner);
ddd = corner - mean_point;
ttttt = atan2(ddd(:,2),ddd(:,1))*180/pi;
[tttt,I] = sort(ttttt,'descend');
corner = corner(I,:);

if norm(corner(2,:)-corner(1,:))>norm(corner(3,:)-corner(2,:))
    b = corner(1,:);
    corner(1:3,:) = corner(2:4,:);
    corner(4,:) = b;
end
% ------------------------------------------------------

figure(3);
imshow(img);
for ii = 1 : size(d,1)
    x = 1:size(img,2);
```

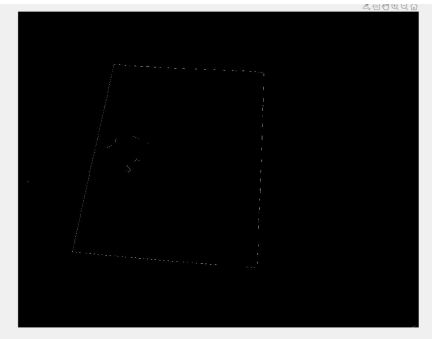
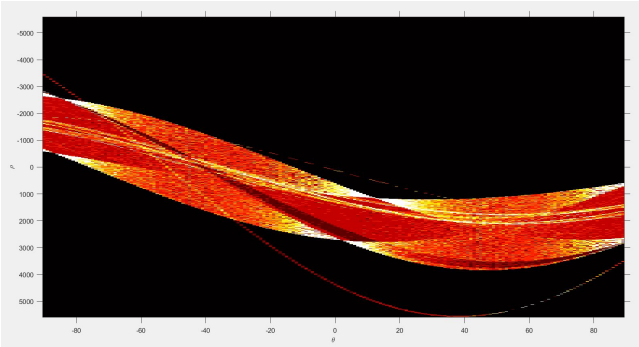
```
    y = 1/sind(theta(ii))*(d(ii)-x*cosd(theta(ii)));
    hold on;
    plot(x, y,'r', 'linewidth', 2);
end
hold off;
end


%%
% finding four points that form a quadrangle from intersecting 4 lines
function corner = search_corner(line)

% main four lines
l1 = line(:,1);
l2 = line(:,2);
l3 = line(:,3);
l4 = line(:,4);

% intersection of these four lines: 6 points
p1 = cross(l1,l2);
p2 = cross(l1,l3);
p3 = cross(l1,l4);
p4 = cross(l2,l3);
p5 = cross(l2,l4);
p6 = cross(l3,l4);
points = [p1 p2 p3 p4 p5 p6];


% finding all the combinations with 4 members (intersected points) in such a way
that
% each member (point) is exactly on two lines
combination = combnk(1:6,4);
combinations = [];
for ii = 1 : size(combination,1)
    intersect = [l1 l2 l3 l4]'*points(:,combination(ii,:));

    % check that all the points of the combination are exactly on two lines
    if length(find(abs(intersect(1,:))<1e-6))==2 &&
length(find(abs(intersect(2,:))<1e-6))==2 && length(find(abs(intersect(3,:))<1e-
6))==2 && length(find(abs(intersect(4,:))<1e-6))==2
        combinations = [combinations ; combination(ii,:)];
    end
end


% Here, we need to filter some combinations to find four points that form a
% quadrangle. We used two steps for filtering. First, we do step 1 to filter
% some combinations. Then, we apply step 2 to filter other combinations.
% Finally, we have only one combination as the final result.
quad_points = [];
for ii = 1 : size(combinations,1)
    index = 0;

    % step 1:
    comb1 = combnk(combinations(ii,:),3);
    for jj = 1 : size(comb1,1)
        p_in = setdiff(combinations(ii,:),comb1(jj,:));
        tri = points(:,[comb1(jj,:) comb1(jj,1)]);
```

```
        % to check that the point is inside the polygon or not
        in =
inpolygon(points(1,p_in)/points(3,p_in),points(2,p_in)/points(3,p_in),tri(1,:)./
tri(3,:),tri(2,:)./tri(3,:));
        if in ~= 0
            index = 1;
        end
    end

    % step 2:
    comb2 = [combinations(ii,1) combinations(ii,3) combinations(ii,2)
combinations(ii,4);
        combinations(ii,1) combinations(ii,3) combinations(ii,4)
combinations(ii,2);
        combinations(ii,1) combinations(ii,2) combinations(ii,3)
combinations(ii,4)];
    extra_points = setdiff([1 2 3 4 5 6],combinations(ii,:));
    for jj = 1 : size(comb2,1)
        tri = points(:,[comb2(jj,:) comb2(jj,1)]);
        in1 =
inpolygon(points(1,extra_points(1))/points(3,extra_points(1)),points(2,extra_poi
nts(1))/points(3,extra_points(1)),tri(1,:)./tri(3,:),tri(2,:)./tri(3,:));
        in2 =
inpolygon(points(1,extra_points(2))/points(3,extra_points(2)),points(2,extra_poi
nts(2))/points(3,extra_points(2)),tri(1,:)./tri(3,:),tri(2,:)./tri(3,:));
        if in1 ~= 0 || in2 ~= 0
            index = 1;
        end
    end

    % final combination
    if index == 0
        % four points that form a quadrangle in HG coordinate
        quad_points = [quad_points ; points(:,combinations(ii,:))];
        % selected combination as a final result
        selected_combination = combinations(ii,:);
    end

end

% final result in homogenuous coordinate
p1_quad_hg = points(:,selected_combination(1));
p2_quad_hg = points(:,selected_combination(2));
p3_quad_hg = points(:,selected_combination(3));
p4_quad_hg = points(:,selected_combination(4));

% final result in cartesian coordinate
p1_quad_car =
[points(1,selected_combination(1))/points(3,selected_combination(1))
points(2,selected_combination(1))/points(3,selected_combination(1))];
p2_quad_car =
[points(1,selected_combination(2))/points(3,selected_combination(2))
points(2,selected_combination(2))/points(3,selected_combination(2))];
p3_quad_car =
[points(1,selected_combination(3))/points(3,selected_combination(3))
points(2,selected_combination(3))/points(3,selected_combination(3))];
```

```
p4_quad_car =
[points(1,selected_combination(4))/points(3,selected_combination(4))
points(2,selected_combination(4))/points(3,selected_combination(4))];

corner = [p1_quad_car;p2_quad_car;p3_quad_car;p4_quad_car];
end
```

Results for method 2:

Image "IMG_0080.JPG":

edge detection	1st peak (Hough Transform)
	

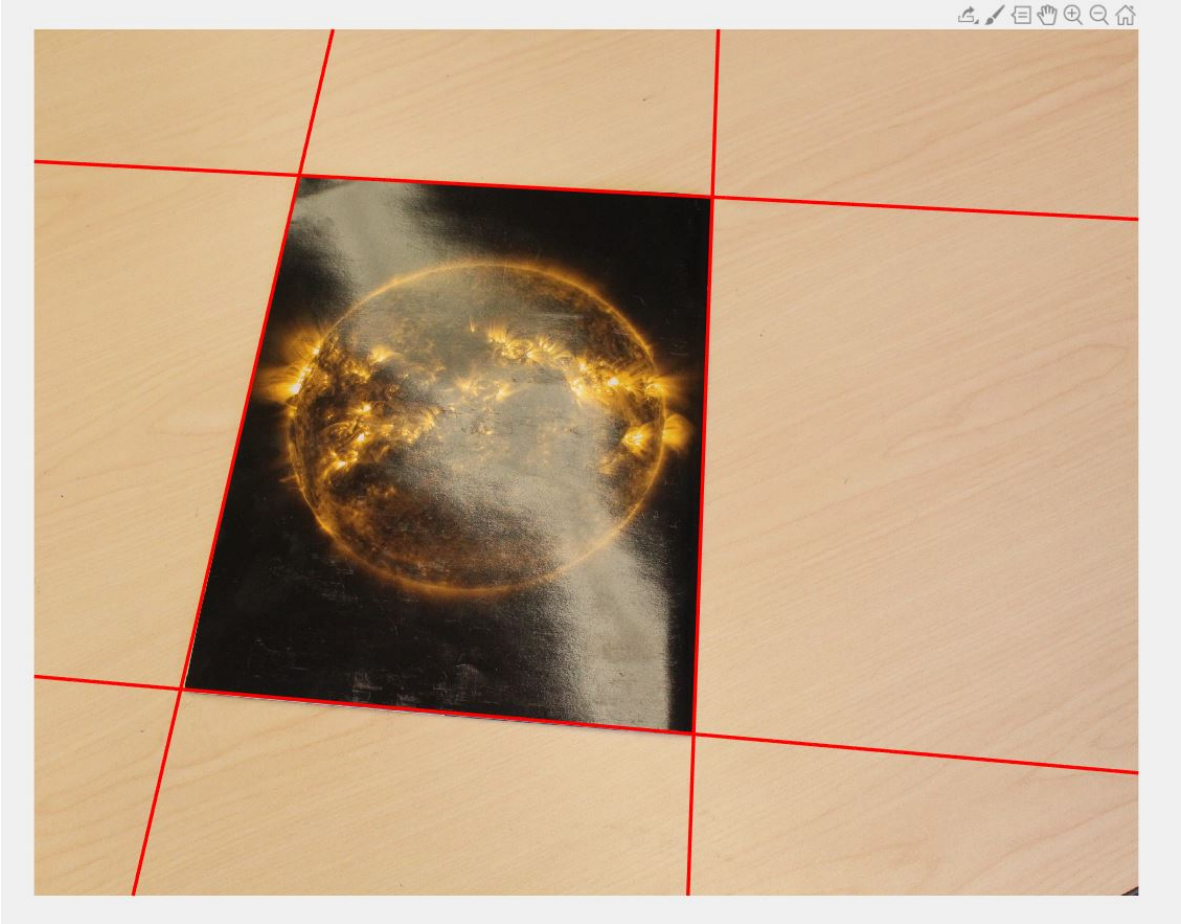
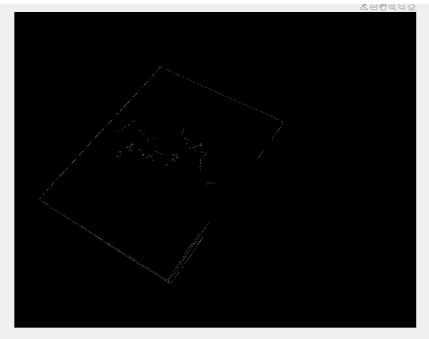
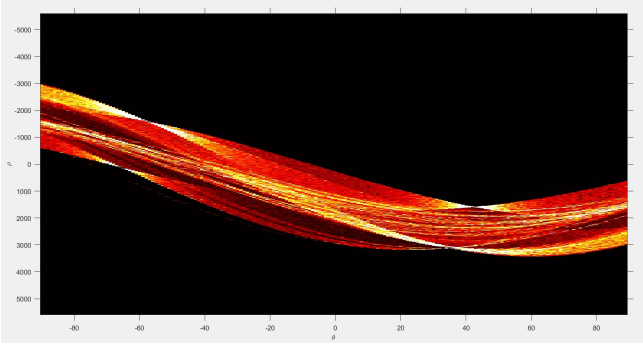
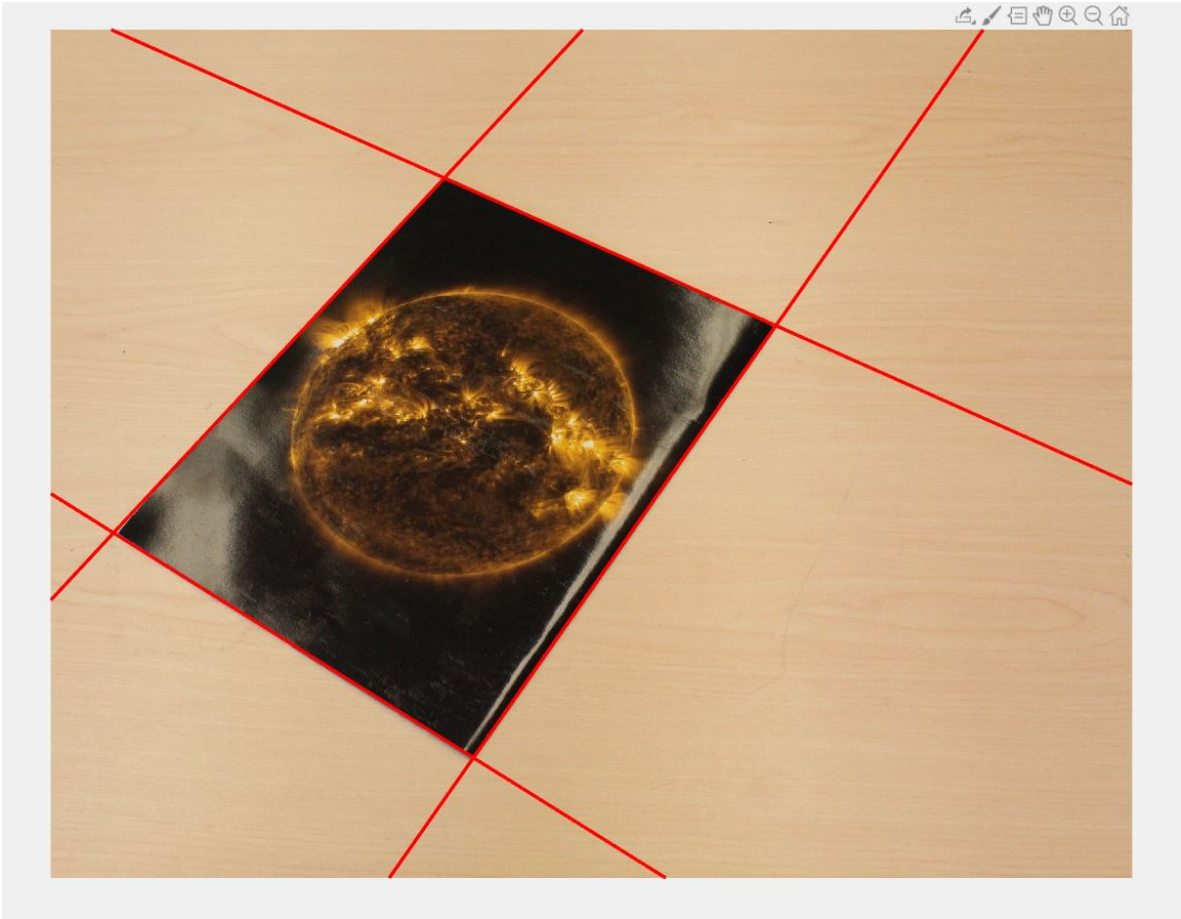


Image "IMG_0081.JPG":

edge detection	1st peak (Hough Transform)
	



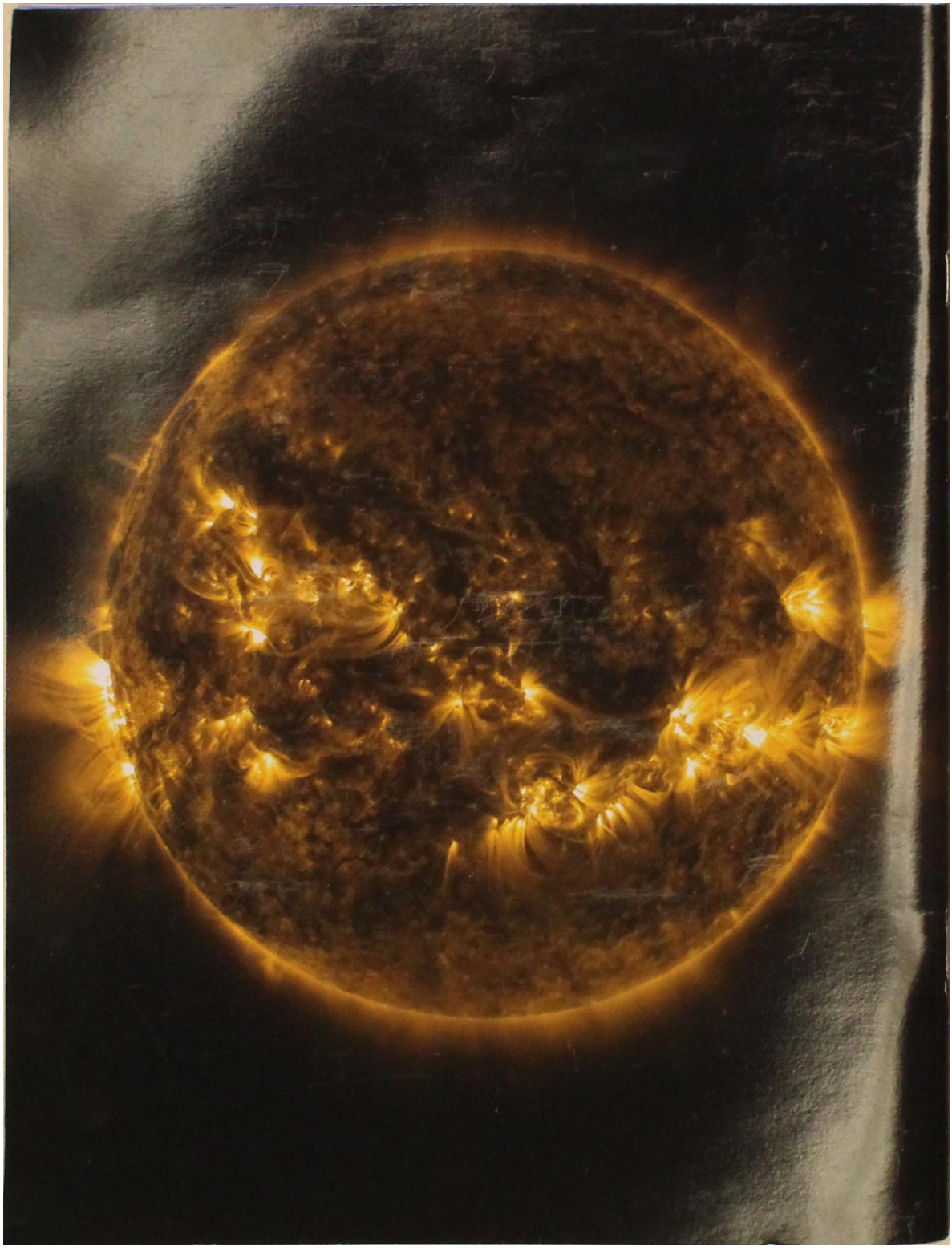
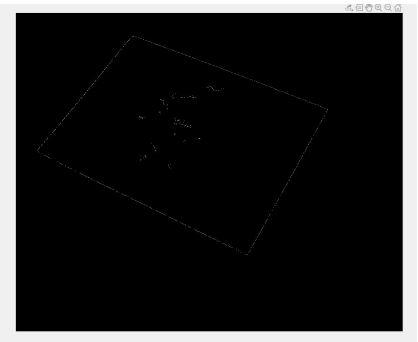
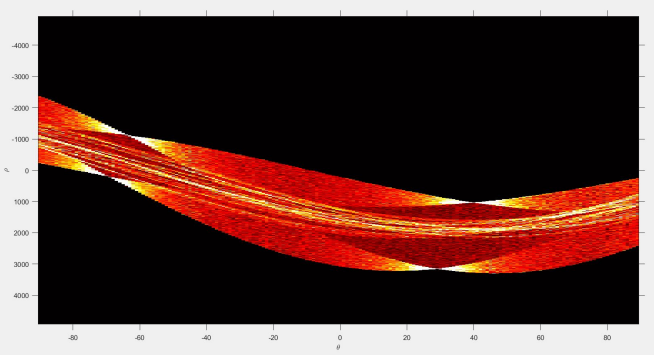


Image "IMG_0082.JPG":

edge detection	1st peak (Hough Transform)
	

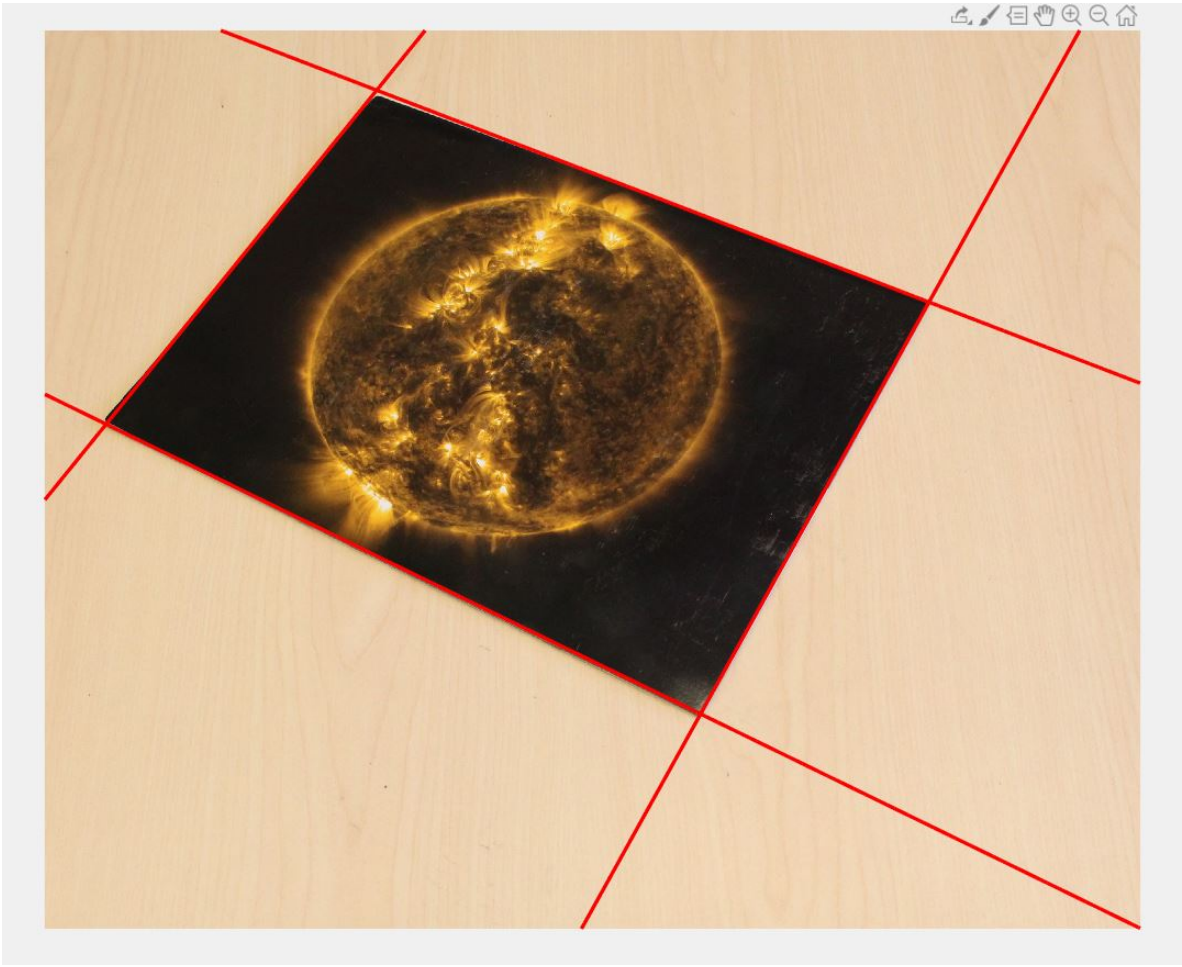
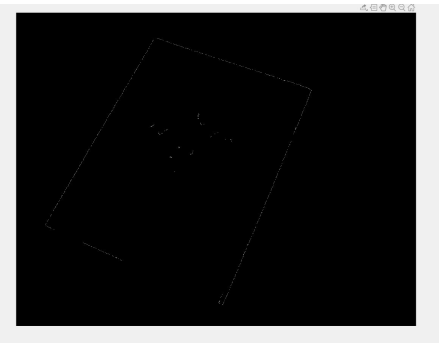
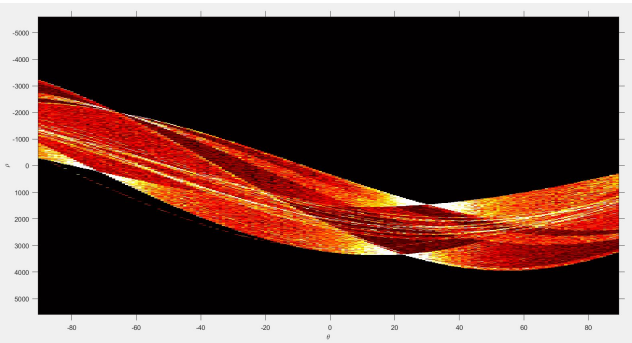
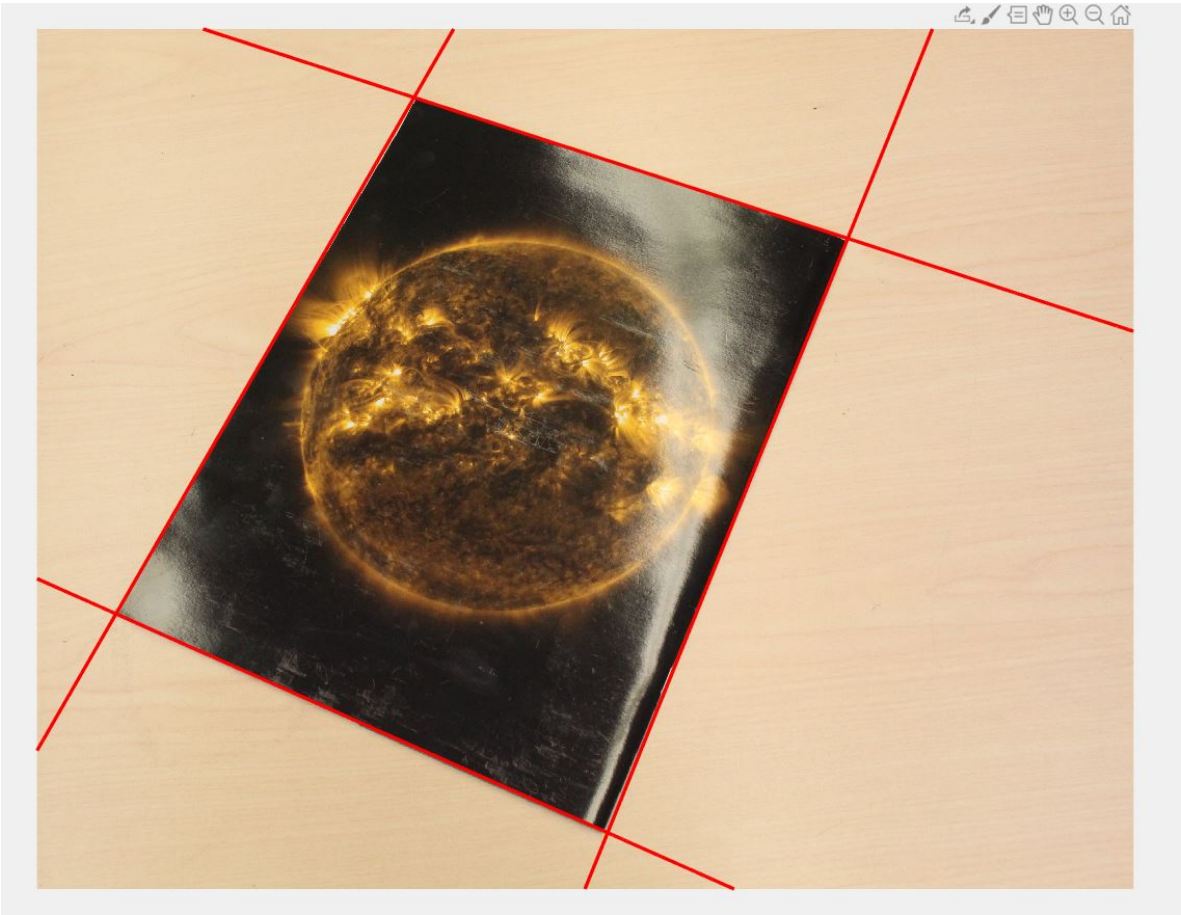


Image "IMG_0084.JPG":

edge detection	1st peak (Hough Transform)
	



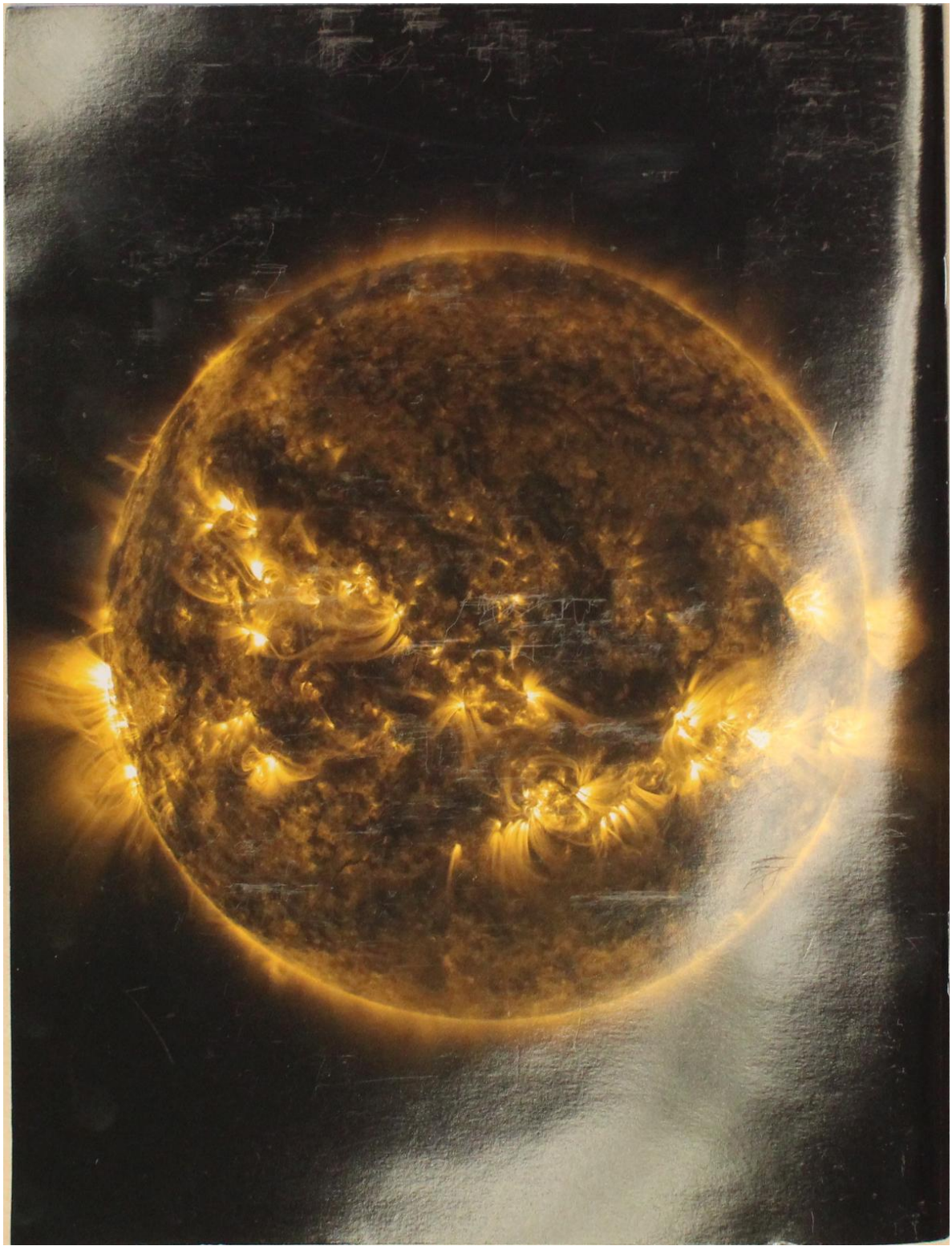
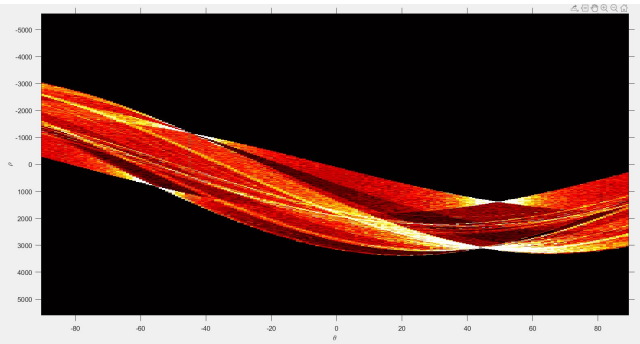
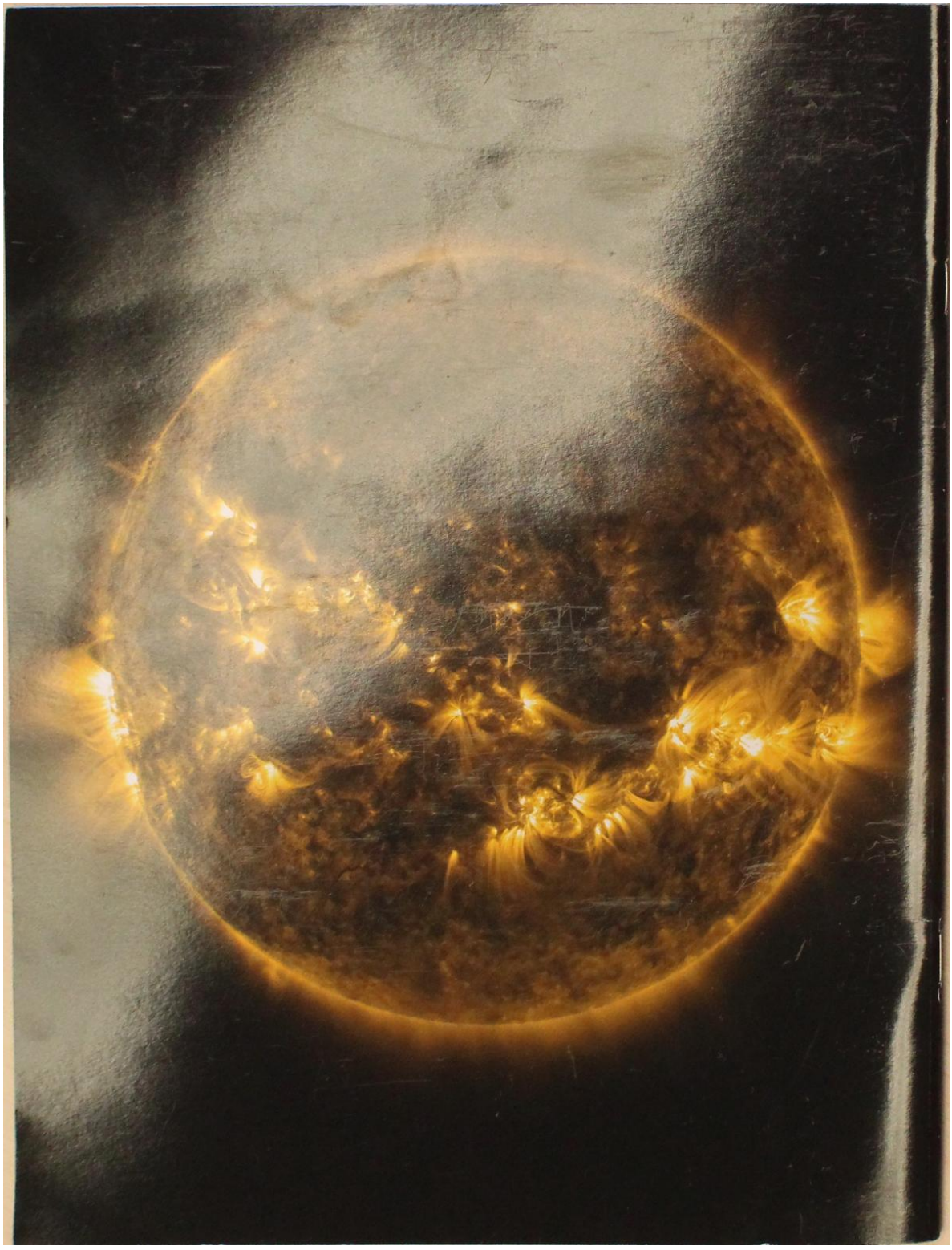


Image "IMG_0085.JPG":

edge detection	1st peak (Hough Transform)
	



Matlab code for method 2:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%---------- Problem 4 - Method 2 ---------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Hough Transformation
% Author: Chul Min Yeum (cmyeum@uwaterloo.ca)
% Last update:: 03/07/2020 by Saeed Hatefi Ardakani
clear; close all; clc; format shortG;

%% Parameter
```

```
bookletSize = [24 31.5]; % cm
bookletImgSize = bookletSize*50; % output image size
dirImg = 'img'; % image folder
dirOut = 'out'; % output image folder
imgList = dir('img/*.JPG');
nImg = numel(imgList);

%% Processing
for ii=1:nImg
    img = imread(fullfile(dirImg, imgList(ii).name));

    % crop the image to remove edge of the table
    img = imcrop(img,[0.15*size(img,2) 0 0.85*size(img,2) size(img,1)]);

    corner = FindCorner(img); % find ordered four corners

    H = ComputeH(bookletImgSize, corner); % compute a homography

    [imgTran, RA] = imwarp(img, projective2d(inv(H)));
    bookletImg = imcrop(imgTran, [-RA.XWorldLimits(1), -RA.YWorldLimits(1)
bookletImgSize]);
    imwrite(bookletImg, fullfile(dirOut,imgList(ii).name));
end


%%
% Homography function
function H = ComputeH(sizePic, corner)
points1 = [1,1;sizePic(1),1;sizePic(1),sizePic(2);1,sizePic(2)];
points2 = corner;
x_xp = points1(:,1).*points2(:,1);
x_yp = points1(:,1).*points2(:,2);
y_xp = points1(:,2).*points2(:,1);
y_yp = points1(:,2).*points2(:,2);

A = zeros(size(points1,1)*2,9);
A(1:2:end,3) = 1;
A(2:2:end,6) = 1;
A(1:2:end,1:2) = points1;
A(2:2:end,4:5) = points1;
A(1:2:end,7) = -x_xp;
A(1:2:end,8) = -y_xp;
A(2:2:end,7) = -x_yp;
A(2:2:end,8) = -y_yp;
A(1:2:end,9) = -points2(:,1);
A(2:2:end,9) = -points2(:,2);

T = null(A);
T = T/T(end);
H = (reshape(T,3,3));
end


%%
% function for finding ordered four corners of the booklet
function corner = FindCorner(img)

% applying Gaussian filter with a sigma=1.4
```

```
img_f1 = imgaussfilt(img,1.4);

% use Canny edge detector
BW_filt = edge(rgb2gray(img_f1), 'Canny',[0.1 0.5]);
fig1 = figure(1);
imshow(BW_filt);
set(fig1,'Position', [100 100 800 600]);

% computing the Standard Hough Transform (SHT) of the binary image BW.
% The hough function is designed to detect lines.
[H,T,R] = hough(BW_filt);
figure(2);
imshow(imadjust(rescale(H)),
[],'XData',T,'YData',R,'InitialMagnification','fit');
xlabel('\theta'), ylabel('\rho');
axis on, axis normal, colormap(gca,hot);

% finding peaks in the Hough transform matrix, H, generated by the hough
function.
p = houghpeaks(H,4,'threshold',ceil(0.2*max(H(:))));

% finding ro-->d and theta related to each line
d = R(p(:,1))';
theta = T(p(:,2))';


% -------------------------------------------------------
% finding four ordered points that form a quadrangle from intersecting 4 lines
% -------------------------------------------------------
% writing homogenuous form of the four lines
for ii = 1 : size(d,1)
    m = -cosd(theta(ii))/sind(theta(ii));
    c = d(ii)/sind(theta(ii));
    line(:,ii) = [m -1 c];
end

% finding four corners using my own function (I wrote this function in task 5)
corner = search_corner(line);

% reordering corners in clockwise pattern:
%        4--------3
%        |        |
%        |        |
%        |        |
%        |        |
%        1--------2
mean_point = mean(corner);
ddd = corner - mean_point;
ttttt = atan2(ddd(:,2),ddd(:,1))*180/pi;
[tttt,I] = sort(ttttt,'descend');
corner = corner(I,:);

if norm(corner(2,:)-corner(1,:))>norm(corner(3,:)-corner(2,:))
    b = corner(1,:);
    corner(1:3,:) = corner(2:4,:);
    corner(4,:) = b;
end
% -------------------------------------------------------
```

```
figure(3);
imshow(img);
for ii = 1 : size(d,1)
    x = 1:size(img,2);
    y = 1/sind(theta(ii))*(d(ii)-x*cosd(theta(ii)));
    hold on;
    plot(x, y,'r', 'linewidth', 2);
end
hold off;
end


%%
% finding four points that form a quadrangle from intersecting 4 lines
function corner = search_corner(line)

% main four lines
l1 = line(:,1);
l2 = line(:,2);
l3 = line(:,3);
l4 = line(:,4);

% intersection of these four lines: 6 points
p1 = cross(l1,l2);
p2 = cross(l1,l3);
p3 = cross(l1,l4);
p4 = cross(l2,l3);
p5 = cross(l2,l4);
p6 = cross(l3,l4);
points = [p1 p2 p3 p4 p5 p6];


% finding all the combinations with 4 members (intersected points) in such a way
that
% each member (point) is exactly on two lines
combination = combnk(1:6,4);
combinations = [];
for ii = 1 : size(combination,1)
    intersect = [l1 l2 l3 l4]'*points(:,combination(ii,:));

    % check that all the points of the combination are exactly on two lines
    if length(find(abs(intersect(1,:))<1e-6))==2 &&
length(find(abs(intersect(2,:))<1e-6))==2 && length(find(abs(intersect(3,:))<1e-
6))==2 && length(find(abs(intersect(4,:))<1e-6))==2
        combinations = [combinations ; combination(ii,:)];
    end
end


% Here, we need to filter some combinations to find four points that form a
% quadrangle. We used two steps for filtering. First, we do step 1 to filter
% some combinations. Then, we apply step 2 to filter other combinations.
% Finally, we have only one combination as the final result.
quad_points = [];
for ii = 1 : size(combinations,1)
    index = 0;
```

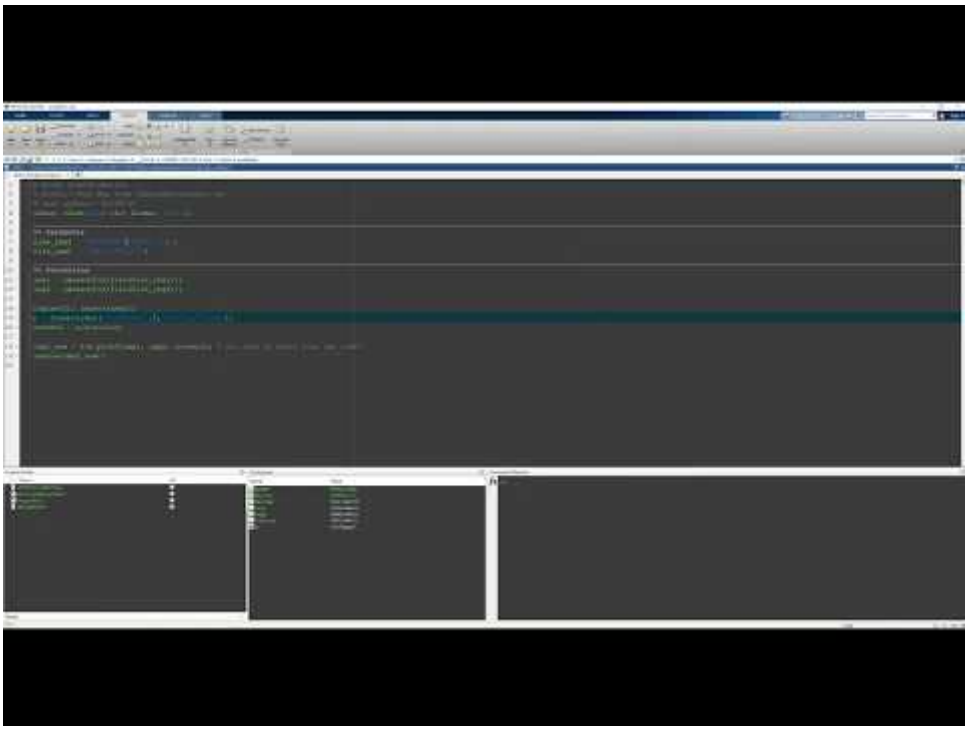
```
    % step 1:
    comb1 = combnk(combinations(ii,:),3);
    for jj = 1 : size(comb1,1)
        p_in = setdiff(combinations(ii,:),comb1(jj,:));
        tri = points(:,[comb1(jj,:) comb1(jj,1)]);

        % to check that the point is inside the polygon or not
        in =
inpolygon(points(1,p_in)/points(3,p_in),points(2,p_in)/points(3,p_in),tri(1,:)./
tri(3,:),tri(2,:)./tri(3,:));
        if in ~= 0
            index = 1;
        end
    end

    % step 2:
    comb2 = [combinations(ii,1) combinations(ii,3) combinations(ii,2)
combinations(ii,4);
        combinations(ii,1) combinations(ii,3) combinations(ii,4)
combinations(ii,2);
        combinations(ii,1) combinations(ii,2) combinations(ii,3)
combinations(ii,4)];
    extra_points = setdiff([1 2 3 4 5 6],combinations(ii,:));
    for jj = 1 : size(comb2,1)
        tri = points(:,[comb2(jj,:) comb2(jj,1)]);
        in1 =
inpolygon(points(1,extra_points(1))/points(3,extra_points(1)),points(2,extra_poi
nts(1))/points(3,extra_points(1)),tri(1,:)./tri(3,:),tri(2,:)./tri(3,:));
        in2 =
inpolygon(points(1,extra_points(2))/points(3,extra_points(2)),points(2,extra_poi
nts(2))/points(3,extra_points(2)),tri(1,:)./tri(3,:),tri(2,:)./tri(3,:));
        if in1 ~= 0 || in2 ~= 0
            index = 1;
        end
    end

    % final combination
    if index == 0
        % four points that form a quadrangle in HG coordinate
        quad_points = [quad_points ; points(:,combinations(ii,:))];
        % selected combination as a final result
        selected_combination = combinations(ii,:);
    end

end

% final result in homogenuous coordinate
p1_quad_hg = points(:,selected_combination(1));
p2_quad_hg = points(:,selected_combination(2));
p3_quad_hg = points(:,selected_combination(3));
p4_quad_hg = points(:,selected_combination(4));

% final result in cartesian coordinate
p1_quad_car =
[points(1,selected_combination(1))/points(3,selected_combination(1))
points(2,selected_combination(1))/points(3,selected_combination(1))];
```

```
p2_quad_car =
[points(1,selected_combination(2))/points(3,selected_combination(2))
points(2,selected_combination(2))/points(3,selected_combination(2))];
p3_quad_car =
[points(1,selected_combination(3))/points(3,selected_combination(3))
points(2,selected_combination(3))/points(3,selected_combination(3))];
p4_quad_car =
[points(1,selected_combination(4))/points(3,selected_combination(4))
points(2,selected_combination(4))/points(3,selected_combination(4))];

corner = [p1_quad_car;p2_quad_car;p3_quad_car;p4_quad_car];
end
```

Problem 5 (Challenging): Hough Transform & Image Overlay (25 points)

This problem is to write your own program that extracts a booklet image from an image provided and overlay the extracted image into the white board. Note the height of the booklet image should be placed at the width of the board.

Here is a sample demo.



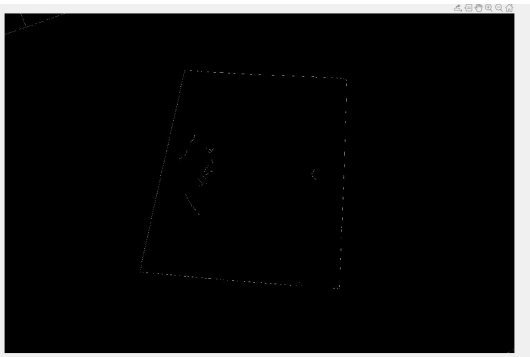
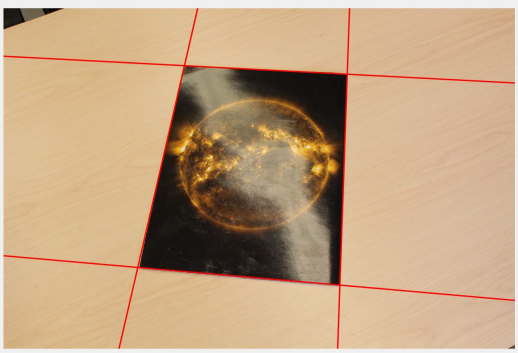
Your goal is to write your own `fun_prob5` function. A sample code is provided in problem5. You can reuse any code that you wrote for solving Problem 4 in Task 6 or Problem 5 or 6 in Task 5.

In order to extract a booklet image from the image provided and to overlay it into the white board, I used two steps in function `fun_prob`.

First, I Overlay the extracted booklet image into an image with an arbitrary size (for example an image with the size of [24 31.5]*50).

Then, in the step 2, I Overlay the booklet image (output of step 1) into the white board.

The result of step 1:

edge detection	edge detection
	

Final result (The result of step 2):

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%---------- Problem 5 ---------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Hough Transformation
% Author: Chul Min Yeum (cmyeum@uwaterloo.ca)
% Last update:: 03/08/2020 by Saeed Hatefi Ardakani
clear; close all; clc; format shortG;

%% Parameter
file_img1 = '20200227_104101.jpg';
file_img2 = 'IMG_0080.JPG';

%% Processing
img1 = imread(fullfile(file_img1));
img2 = imread(fullfile(file_img2));

% Please click on the 4 corners in the following order:
% upper left, lower left, lower right, upper right
figure(1); imshow(img1);
p = drawpolygon('LineWidth',5,'Color','black');
corner1 = p.Position;

img2_new = fun_prob5(img1, img2, corner1); % you need to write your own code!
figure(5)
imshow(img2_new);
imwrite(img2_new, 'result.jpg');
```

```
%%
function img2_new = fun_prob5(img1, img2, corner1)

% Step1: Overlaying the extracted booklet image into an image with
% arbitrary size
bookletSize = [24 31.5]; % cm
bookletImgSize = bookletSize*50; % output image size

corner2 = FindCorner(img2); % find ordered four corners of the booklet

H = ComputeH(bookletImgSize, corner2); % compute a homography

[imgTran, RA] = imwarp(img2, projective2d(inv(H)));
bookletImg = imcrop(imgTran, [-RA.XWorldLimits(1), -RA.YWorldLimits(1)
bookletImgSize]);


% Step2: Overlaying the booklet image (output of step 1) into the white
% board
H = ComputeH(bookletImgSize, corner1); % compute a homography

[imgPicTran, RB] = imwarp(bookletImg, projective2d(H));
BWPic = roipoly(imgPicTran, corner1(:,1)-RB.XWorldLimits(1), corner1(:,2)-
RB.YWorldLimits(1));

BWBoard = ~roipoly(img1, corner1(:,1), corner1(:,2));
RA = imref2d(size(BWBoard));

imgBoardMask = bsxfun(@times, img1, cast(BWBoard, 'like', img1));
imgPicTranMask = bsxfun(@times, imgPicTran, cast(BWPic, 'like', imgPicTran));

img2_new(:,:,1) = imfuse(imgBoardMask(:,:,1),RA,
imgPicTranMask(:,:,1),RB,'diff');
img2_new(:,:,2) = imfuse(imgBoardMask(:,:,2),RA,
imgPicTranMask(:,:,2),RB,'diff');
img2_new(:,:,3) = imfuse(imgBoardMask(:,:,3),RA,
imgPicTranMask(:,:,3),RB,'diff');
end


%%
% Homography function
function H = ComputeH(sizePic, corner)
points1 = [1,1;sizePic(1),1;sizePic(1),sizePic(2);1,sizePic(2)];
points2 = corner;
x_xp = points1(:,1).*points2(:,1);
x_yp = points1(:,1).*points2(:,2);
y_xp = points1(:,2).*points2(:,1);
y_yp = points1(:,2).*points2(:,2);

A = zeros(size(points1,1)*2,9);
A(1:2:end,3) = 1;
A(2:2:end,6) = 1;
A(1:2:end,1:2) = points1;
A(2:2:end,4:5) = points1;
A(1:2:end,7) = -x_xp;
```

```
A(1:2:end,8) = -y_xp;
A(2:2:end,7) = -x_yp;
A(2:2:end,8) = -y_yp;
A(1:2:end,9) = -points2(:,1);
A(2:2:end,9) = -points2(:,2);

T = null(A);
T = T/T(end);
H = (reshape(T,3,3));
end


%%
% function for finding ordered four corners of the booklet
function corner = FindCorner(img)
imshow(img);

% applying Gaussian filter with a sigma=1.4
img_f1 = imgaussfilt(img,5);

% use Canny edge detector
BW_filt = edge(rgb2gray(img_f1), 'Canny',[0.1 0.5]);
fig2 = figure(2);
imshow(BW_filt);
set(fig2,'Position', [100 100 800 600]);

% computing the Standard Hough Transform (SHT) of the binary image BW.
% The hough function is designed to detect lines.
[H,T,R] = hough(BW_filt);
figure(3);
imshow(imadjust(rescale(H)),
[],'XData',T,'YData',R,'InitialMagnification','fit');
xlabel('\theta'), ylabel('\rho');
axis on, axis normal, colormap(gca,hot);

% finding peaks in the Hough transform matrix, H, generated by the hough
function.
p = houghpeaks(H,4,'threshold',ceil(0.2*max(H(:))));

% finding ro-->d and theta related to each line
d = R(p(:,1))';
theta = T(p(:,2))';

% ------------------------------------------------------
% finding four ordered points that form a quadrangle from intersecting 4 lines
% ------------------------------------------------------
% writing homogenuous form of the four lines
for ii = 1 : size(d,1)
    m = -cosd(theta(ii))/sind(theta(ii));
    c = d(ii)/sind(theta(ii));
    line(:,ii) = [m -1 c];
end

% finding four corners using my own function (I wrote this function in task 5)
corner = search_corner(line);

% reordering corners in clockwise pattern:
%        4--------3
```

```
%        |        |
%        |        |
%        |        |
%        |        |
%        1--------2
mean_point = mean(corner);
ddd = corner - mean_point;
ttttt = atan2(ddd(:,2),ddd(:,1))*180/pi;
[tttt,I] = sort(ttttt,'ascend');
corner = corner(I,:);

if norm(corner(2,:)-corner(1,:))>norm(corner(3,:)-corner(2,:))
    b = corner(1,:);
    corner(1:3,:) = corner(2:4,:);
    corner(4,:) = b;
end
% ----------------------------------------------------

figure(4);
imshow(img);
for ii = 1 : size(d,1)
    x = 1:size(img,2);
    y = 1/sind(theta(ii))*(d(ii)-x*cosd(theta(ii)));
    hold on;
    plot(x, y,'r', 'linewidth', 2);
end
hold off;
end


%%
% finding four points that form a quadrangle from intersecting 4 lines
function corner = search_corner(line)
% main four lines
l1 = line(:,1);
l2 = line(:,2);
l3 = line(:,3);
l4 = line(:,4);

% intersection of these four lines: 6 points
p1 = cross(l1,l2);
p2 = cross(l1,l3);
p3 = cross(l1,l4);
p4 = cross(l2,l3);
p5 = cross(l2,l4);
p6 = cross(l3,l4);
points = [p1 p2 p3 p4 p5 p6];


% finding all the combinations with 4 members (intersected points) in such a way
that
% each member (point) is exactly on two lines
combination = combnk(1:6,4);
combinations = [];
for ii = 1 : size(combination,1)
    intersect = [l1 l2 l3 l4]'*points(:,combination(ii,:));

    % check that all the points of the combination are exactly on two lines
```

```
    if length(find(abs(intersect(1,:))<1e-6))==2 &&
length(find(abs(intersect(2,:))<1e-6))==2 && length(find(abs(intersect(3,:))<1e-
6))==2 && length(find(abs(intersect(4,:))<1e-6))==2
        combinations = [combinations ; combination(ii,:)];
    end
end


% Here, we need to filter some combinations to find four points that form a
% quadrangle. We used two steps for filtering. First, we do step 1 to filter
% some combinations. Then, we apply step 2 to filter other combinations.
% Finally, we have only one combination as the final result.
quad_points = [];
for ii = 1 : size(combinations,1)
    index = 0;

    % step 1:
    comb1 = combnk(combinations(ii,:),3);
    for jj = 1 : size(comb1,1)
        p_in = setdiff(combinations(ii,:),comb1(jj,:));
        tri = points(:,[comb1(jj,:) comb1(jj,1)]);

        % to check that the point is inside the polygon or not
        in =
inpolygon(points(1,p_in)/points(3,p_in),points(2,p_in)/points(3,p_in),tri(1,:)./
tri(3,:),tri(2,:)./tri(3,:));
        if in ~= 0
            index = 1;
        end
    end

    % step 2:
    comb2 = [combinations(ii,1) combinations(ii,3) combinations(ii,2)
combinations(ii,4);
        combinations(ii,1) combinations(ii,3) combinations(ii,4)
combinations(ii,2);
        combinations(ii,1) combinations(ii,2) combinations(ii,3)
combinations(ii,4)];
    extra_points = setdiff([1 2 3 4 5 6],combinations(ii,:));
    for jj = 1 : size(comb2,1)
        tri = points(:,[comb2(jj,:) comb2(jj,1)]);
        in1 =
inpolygon(points(1,extra_points(1))/points(3,extra_points(1)),points(2,extra_poi
nts(1))/points(3,extra_points(1)),tri(1,:)./tri(3,:),tri(2,:)./tri(3,:));
        in2 =
inpolygon(points(1,extra_points(2))/points(3,extra_points(2)),points(2,extra_poi
nts(2))/points(3,extra_points(2)),tri(1,:)./tri(3,:),tri(2,:)./tri(3,:));
        if in1 ~= 0 || in2 ~= 0
            index = 1;
        end
    end

    % final combination
    if index == 0
        % four points that form a quadrangle in HG coordinate
        quad_points = [quad_points ; points(:,combinations(ii,:))];
        % selected combination as a final result
        selected_combination = combinations(ii,:);
```

```
    end

end

% final result in homogenuous coordinate
p1_quad_hg = points(:,selected_combination(1));
p2_quad_hg = points(:,selected_combination(2));
p3_quad_hg = points(:,selected_combination(3));
p4_quad_hg = points(:,selected_combination(4));

% final result in cartesian coordinate
p1_quad_car =
[points(1,selected_combination(1))/points(3,selected_combination(1))
points(2,selected_combination(1))/points(3,selected_combination(1))];
p2_quad_car =
[points(1,selected_combination(2))/points(3,selected_combination(2))
points(2,selected_combination(2))/points(3,selected_combination(2))];
p3_quad_car =
[points(1,selected_combination(3))/points(3,selected_combination(3))
points(2,selected_combination(3))/points(3,selected_combination(3))];
p4_quad_car =
[points(1,selected_combination(4))/points(3,selected_combination(4))
points(2,selected_combination(4))/points(3,selected_combination(4))];

corner = [p1_quad_car;p2_quad_car;p3_quad_car;p4_quad_car];
end
```