Task06: Image Processing

Name: Jing Zhang
Degree: PH
ID: 20984622

The main goal of this task is to understand the fundamental of the linear filter, edge detection, and Hough transform. You must study the tutorials at edge detection, linear filter, and image morphology before solving these problems. Please answer all sub-questions in each problem. You should write your own code to solve these questions.

📣📣 **PLEASE INCLUDE YOUR RESULTING IMAGES OR GRAPHS IN THE REPORT**

Problem 1: 2D Convolution (20 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 colour picture or download any colour image on the web, and resize the image to have 400 pixel in the shortest side. For example, if your image 5000 x 4000 pixel, it becomes 500x400 pixel. Then, please do the convolution of the resized image and the kernel H:

$$h = \begin{bmatrix} 0 & -0.5 & 0 \\ -0.5 & 2 & -0.5 \\ 0 & -0.5 & 0 \end{bmatrix}$$

(a) Please compute G using `conv2`. In the shape option, you set `same`. For Python users, you can use `scipy.signal.convolve2d`

(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.

```
clc
clear
close all

% import imgage
img_path = './Img/psyduck.jpg';
img = imread(img_path);
img = im2double(img);
info = imfinfo(img_path);
N = info.NumberOfSamples;
Short_size = 400;
img_new = imresize(img, Short_size/min(info.Width,info.Height));
% kernel
H = [0, -0.5, 0; -0.5, 2, -0.5; 0, -0.5, 0];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% (a) using conv2
```

```
for a = 1:N
    img_conv2(:,:,a) = conv2(img_new(:,:,a), H, 'same');
end
imwrite(img_conv2, 'img/1_1.jpg');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% (b) using for-loop
k = (length(H)-1)/2;
size_Resized = size(img_new);
img_Height = size_Resized(1);
img_Width = size_Resized(2);
img_Padded = zeros(img_Height+2, img_Width+2, N);
img_Padded(2:end-1,2:end-1,:) = img_new(:,:,:);
img_new_2 = zeros(img_Height, img_Width, N);
% Perform convolution in a loop
for a = 1:N
    for ii = 1:img_Height
        for jj = 1:img_Width
            for u = -k:k
                for v = -k:k
                    img_new_2(ii,jj,a) = img_new_2(ii,jj,a) + H((2*k)+u,
(2*k)+v)*img_Padded(ii-u+1,jj-v+1,a);
                end
            end
        end
    end
end
imwrite(img_new_2, 'img/1_2.jpg');

```

Original Figure：

(a) Figure using conv2 function

(b) Figure using for-loop structure

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, H4, and H5, respectively. Then, you need to explain the effect of filtering with the each kernel with the resulting images.

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

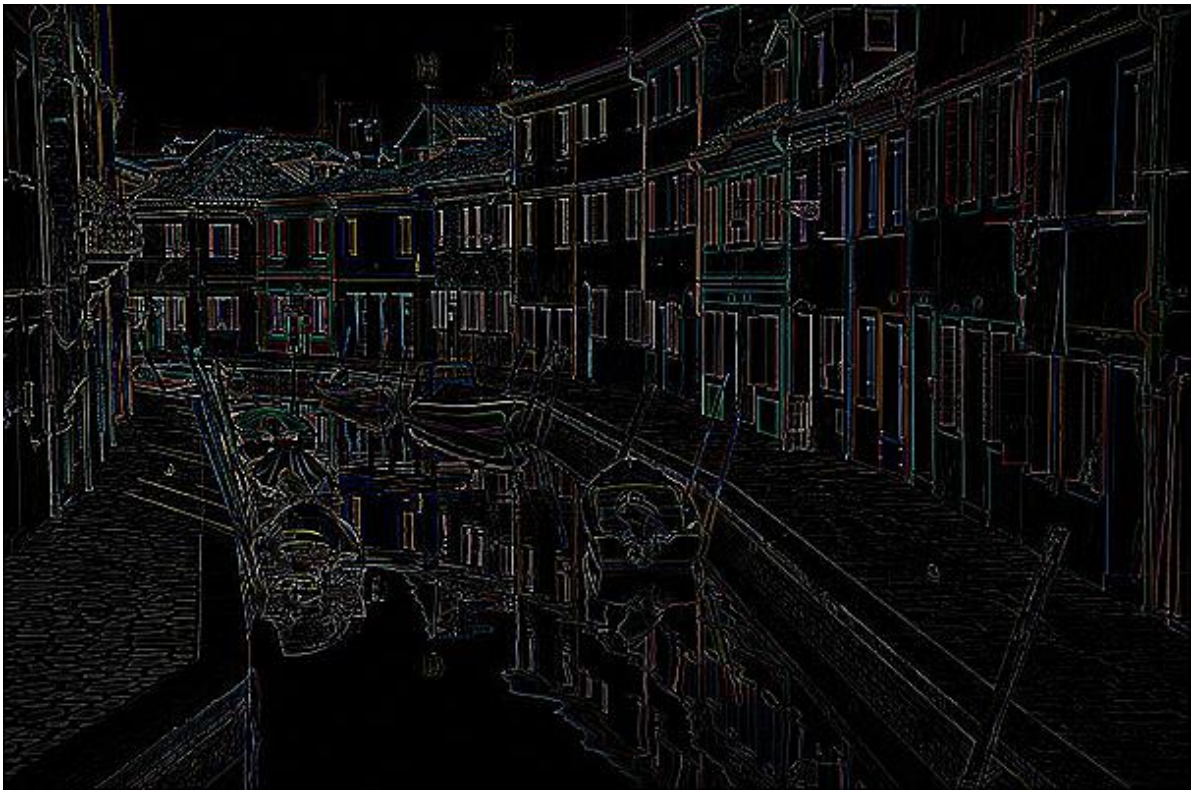
$$H_4 = \begin{bmatrix}0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030\\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133\\ 0.0219 & 0.0983 & 0.1621 & 0.0983 & 0.0219\\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133\\ 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030\end{bmatrix},\quad H_5 = \begin{bmatrix}0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 1\\ 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0\end{bmatrix}$$

```
clc
clear
close all

% import imgage
img_path = './Img/pic.jpg';
img = imread(img_path);
img = im2double(img);
info = imfinfo(img_path);
N = info.NumberOfSamples;
Short_size = 400;
img_new = imresize(img, Short_size/min(info.Width,info.Height));

% 5 kernels
H1 = (1/9)*ones(3,3);
H2 = [0, 1, 0; 1, -4, 1; 0, 1, 0];
H3 = (-0.55)*ones(3,3);
H3(2,2) = 5.40;
H4 = [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];
H5 = zeros(5,5);
H5(3,5) = 1;

for i = 1:N
    img_H1(:,:,i) = conv2(img_new(:,:,i), H1, 'same');
    img_H2(:,:,i) = conv2(img_new(:,:,i), H2, 'same');
    img_H3(:,:,i) = conv2(img_new(:,:,i), H3, 'same');
    img_H4(:,:,i) = conv2(img_new(:,:,i), H4, 'same');
    img_H5(:,:,i) = conv2(img_new(:,:,i), H5, 'same');
end

imwrite(img_H1, 'img/2_1.jpg');
imwrite(img_H2, 'img/2_2.jpg');
imwrite(img_H3, 'img/2_3.jpg');
imwrite(img_H4, 'img/2_4.jpg');
imwrite(img_H5, 'img/2_5.jpg');

```

Original Figure:

H_1: The filter H1 averages the value of each pixel in a 3x3 kernel and then assigns the resulting mean to the value of the center point. Therefore, this value is equal to the average value of its neighboring pixels in the input image. Through this calculation, the output image achieved a uniform blurring effect application.

H_2: The 3x3 kernel filter compares the pixel value in the centre point to its four direct neighbour pixels. If these all pixels have the same colour value in any region, this field will present black in the outcome image. On contrary, the kind of areas will become more highlighted if it has a difference between the centre pixel and its adjacent pixels. In summary, the algorithm will reach highlight the edges effect of an image.



H_3: The 3x3 filter enlarges the comparison in areas where adjacent pixels have different values. The method is increasing the value at the pixel located in the centre point while equally subtracting all the rest eight of the pixel values in the kernel Then, the total sum pixel value in the kernel is one. So, the aim of the kernel is to sharpen the image. In detail, if neighbouring pixels are the same value in any region, there is no obvious change in the area of the output image. However, the colour discontinuity at the edges of objects is exaggerated through using this kernel. Hence, the edges look sharper.

H_4: The filter weights values utilizing a 5x5 kernel. Besides, the kernel includes values from a Gaussian distribution function. Therefore, the outcome is the application of the Gaussian blur effect into the image. Although the final outcome is similar to the output image whose computation uses the first kernel in this sub-problem, they arrive at the effect of blurring the

image. Even though, due to the larger kernel size and gradual kernel value change in the Gaussian kernel, the image produced by the kernel looks a little bit smoother and flatter.

H_5: According to the definition of convolution, the effect is to shift the whole picture to the right by 2 pixels. The reason is that in the 5*5 kernel, only the position of H5 (3, 5) is 1 while the value of other positions is 0. Additionally, although the kernel application effect is not obvious as the previous ones, you can still find there exist a clear black edge line on the left of the outcome image. Consequently, This filter shifts the pixels right.

Problem 3: Gaussian Kernel (20 points)

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

```
h = fspecial('gaussian', 11, 3);
```

You need to use a 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.

```
clc
clear
close all

h0 = fspecial('gaussian', 11, 3)


f_size = 11;
sigma = 3;
h = zeros(f_size, f_size, 3);

for i = 1:f_size
    h(i,:,1) = (i-ceil(f_size/2))^2;
    h(:,i,2) = (i-ceil(f_size/2))^2;
end
h(:,:,3) = h(:,:,1)+h(:,:,2);
h = h(:,:,3);

h = ((1/(2*pi*(sigma^2)))*exp(-h/(2*(sigma^2))));
h = h./sum(sum(h))


```

From the results, The h0 and h are same through fspecial(h0) and theoretical way(h).

```
h0 =

    0.0013    0.0021    0.0031    0.0040    0.0048    0.0050    0.0048    0.0040    0.0031    0.0021    0.0013
    0.0021    0.0034    0.0050    0.0067    0.0079    0.0083    0.0079    0.0067    0.0050    0.0034    0.0021
    0.0031    0.0050    0.0074    0.0098    0.0116    0.0123    0.0116    0.0098    0.0074    0.0050    0.0031
    0.0040    0.0067    0.0098    0.0130    0.0153    0.0162    0.0153    0.0130    0.0098    0.0067    0.0040
    0.0048    0.0079    0.0116    0.0153    0.0181    0.0192    0.0181    0.0153    0.0116    0.0079    0.0048
    0.0050    0.0083    0.0123    0.0162    0.0192    0.0202    0.0192    0.0162    0.0123    0.0083    0.0050
    0.0048    0.0079    0.0116    0.0153    0.0181    0.0192    0.0181    0.0153    0.0116    0.0079    0.0048
    0.0040    0.0067    0.0098    0.0130    0.0153    0.0162    0.0153    0.0130    0.0098    0.0067    0.0040
    0.0031    0.0050    0.0074    0.0098    0.0116    0.0123    0.0116    0.0098    0.0074    0.0050    0.0031
    0.0021    0.0034    0.0050    0.0067    0.0079    0.0083    0.0079    0.0067    0.0050    0.0034    0.0021
    0.0013    0.0021    0.0031    0.0040    0.0048    0.0050    0.0048    0.0040    0.0031    0.0021    0.0013


h =

    0.0013    0.0021    0.0031    0.0040    0.0048    0.0050    0.0048    0.0040    0.0031    0.0021    0.0013
    0.0021    0.0034    0.0050    0.0067    0.0079    0.0083    0.0079    0.0067    0.0050    0.0034    0.0021
    0.0031    0.0050    0.0074    0.0098    0.0116    0.0123    0.0116    0.0098    0.0074    0.0050    0.0031
    0.0040    0.0067    0.0098    0.0130    0.0153    0.0162    0.0153    0.0130    0.0098    0.0067    0.0040
    0.0048    0.0079    0.0116    0.0153    0.0181    0.0192    0.0181    0.0153    0.0116    0.0079    0.0048
    0.0050    0.0083    0.0123    0.0162    0.0192    0.0202    0.0192    0.0162    0.0123    0.0083    0.0050
    0.0048    0.0079    0.0116    0.0153    0.0181    0.0192    0.0181    0.0153    0.0116    0.0079    0.0048
    0.0040    0.0067    0.0098    0.0130    0.0153    0.0162    0.0153    0.0130    0.0098    0.0067    0.0040
    0.0031    0.0050    0.0074    0.0098    0.0116    0.0123    0.0116    0.0098    0.0074    0.0050    0.0031
    0.0021    0.0034    0.0050    0.0067    0.0079    0.0083    0.0079    0.0067    0.0050    0.0034    0.0021
    0.0013    0.0021    0.0031    0.0040    0.0048    0.0050    0.0048    0.0040    0.0031    0.0021    0.0013

fx >>
```

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

```
h1 = fspecial('gaussian', 20, 2);
h2 = fspecial('gaussian', 20, 4);
```

```
clc
clear
close all

% import imgage
img_path = './Img/psyduck.jpg';
img = imread(img_path);
img = im2double(img);
info = imfinfo(img_path);
N = info.NumberOfSamples;
Short_size = 400;
img_new = imresize(img, Short_size/min(info.Width,info.Height));

h1 = fspecial('gaussian', 20, 2);
h2 = fspecial('gaussian', 20, 4);

for i = 1:N
    img_h1(:,:,i) = conv2(img_new(:,:,i), h1, 'same');
    img_h2(:,:,i) = conv2(img_new(:,:,i), h2, 'same');
end

imwrite(img_h1, 'img/3_b1.jpg');
imwrite(img_h2, 'img/3_b2.jpg');

```

`h1 = fspecial('gaussian', 20, 2);`	`h2 = fspecial('gaussian', 20, 4);`
	

Hihger sigma value filters out more high frequency components and make the image more blurry.

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

```
h2 = fspecial('gaussian', 11, 3);
h3 = fspecial('gaussian', 51, 3);
```

```
clc
clear
close all

% import imgage
img_path = './Img/psyduck.jpg';
img = imread(img_path);
img = im2double(img);
info = imfinfo(img_path);
N = info.NumberOfSamples;
Short_size = 400;
img_new = imresize(img, Short_size/min(info.Width,info.Height));

h2 = fspecial('gaussian', 11, 3);
h3 = fspecial('gaussian', 51, 3);

for i = 1:N
    img_h2(:,:,i) = conv2(img_new(:,:,i), h2, 'same');
    img_h3(:,:,i) = conv2(img_new(:,:,i), h3, 'same');
end

imwrite(img_h2, 'img/3_c1.jpg');
imwrite(img_h3, 'img/3_c2.jpg');

```

`h2 = fspecial('gaussian', 11, 3);`	`h3 = fspecial('gaussian', 51, 3);`
	

With the same sigma value, filter size has little impact on image quality.

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

```
h2 = fspecial('gaussian', 11, 3);
h4 = fspecial('gaussian', 21, 21);
```

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

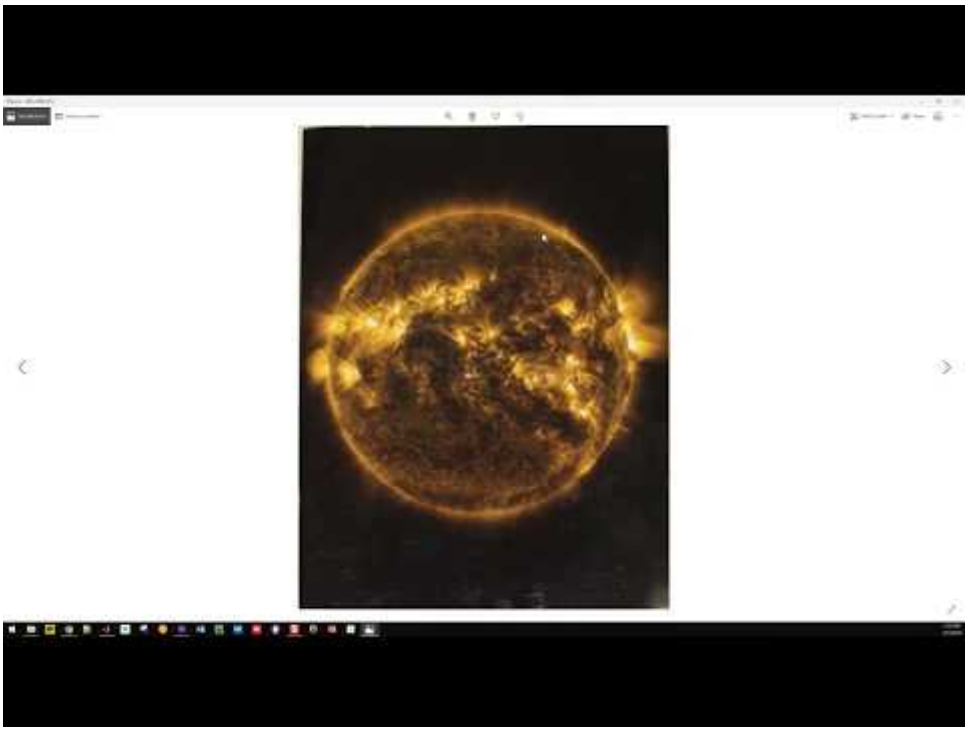
```
clc
clear
close all

% import imgage
img_path = './Img/psyduck.jpg';
img = imread(img_path);
img = im2double(img);
info = imfinfo(img_path);
N = info.NumberOfSamples;
Short_size = 400;
img_new = imresize(img, Short_size/min(info.Width,info.Height));

h2 = fspecial('gaussian', 11, 3);
h4 = fspecial('gaussian', 21, 21);

for i = 1:N
    img_h2(:,:,i) = conv2(img_new(:,:,i), h2, 'same');
    img_h3(:,:,i) = conv2(img_new(:,:,i), h4, 'same');
end

imwrite(img_h2, 'img/3_d1.jpg');
imwrite(img_h4, 'img/3_d2.jpg');

```

`h2 = fspecial('gaussian', 11, 3);`	`h4 = fspecial('gaussian', 21, 21);`
	

The image processed by a kernel with higher sigma value is more blurry

Problem 4: Hough Transform (20 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 in **problem4**. You need to write your own code for extracting four corners of the booklet using edge detection and Hough transform on the given images. In other word, you need to write your own `FindCorner` function. In addition, you might need your `ComputeH`, which was used in Task05. Note that you only remove perspective distortion, meaning that resulting images could be horizontally or vertically flipped.

Next, you need to take photos and demonstrate your code using your own images. Certainly, your code should work on your own images as well. Please include both results and codes in your submission.

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.

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

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

    [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

function Corner = FindCorner(img)

% Change to Gray image
img = rgb2gray(img);
img = imgaussfilt(img,3);

% edge function
BW = edge(img, 'canny', [0.1 0.3]);
rho_res = 0.6;
theta_rng = -90:0.5:89;

% hough function
[H,T,R] = hough(BW,'RhoResolution',rho_res,'Theta',theta_rng);

 % find peaks in the hough matrix
p = houghpeaks(H,4,'Threshold',0.1*max(H(:)));

% extract line
lines = houghlines(BW,T,R,p);
pk_lines = [lines.theta;lines.rho];

for ii=1:size(pk_lines,2)
    theta = pk_lines(1,ii);
    rho = pk_lines(2,ii);
    m(ii) = -cosd(theta)/sind(theta);
    b(ii) = rho/sind(theta);
end
[m_uni,ind_uni] = uniquetol(m,0.0001);
b_uni = b(ind_uni);

% Line representation in HC
edges = (-1)*ones(3,numel(m_uni));
edges(1,:) = m_uni; edges(3,:) = b_uni;

% Find points of intersection
pts_size = nchoosek(size(edges,2),2);
pts = zeros(3,pts_size);
count = 1;
for ii = 1:size(edges,2)
    for jj = (ii+1):size(edges,2)
        pts(:,count) = cross(edges(:,ii),edges(:,jj));
        count = count+1;
    end
end
% Points in R2
pts_out = zeros(2,size(pts,2));
pts_out(1,:) = pts(1,:)./pts(3,:);
pts_out(2,:) = pts(2,:)./pts(3,:);
ii = 1;
while ii<=size(pts_out,2)
    if any(pts_out(:,ii)<0)||any(pts_out(:,ii)>size(img,2))
        pts_out(:,ii)=[];
        ii = ii;
```

```
    else
        ii = ii+1;
    end
end
% Order of the points: bottom left->top left->top right->bottom right
pt1 = pts_out(:,1); pt2 = pts_out(:,2); pt3 = pts_out(:,3);
dist1 = sqrt((pt1(1)-pt2(1))^2+(pt1(2)-pt2(2))^2);
dist2 = sqrt((pt1(1)-pt3(1))^2+(pt1(2)-pt3(2))^2);
if dist1>dist2
    Corner = pts_out(:,[2 4 3 1]);
else
    Corner = pts_out(:,[4 3 1 2]);
end
Corner = Corner';
end
```

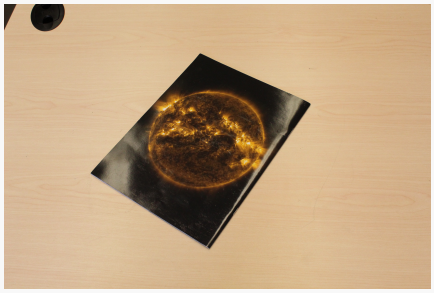
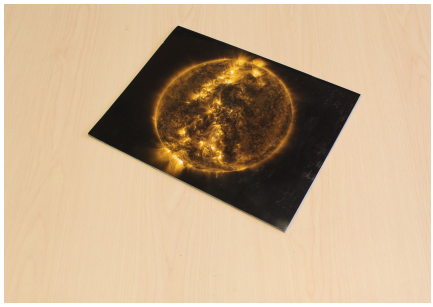
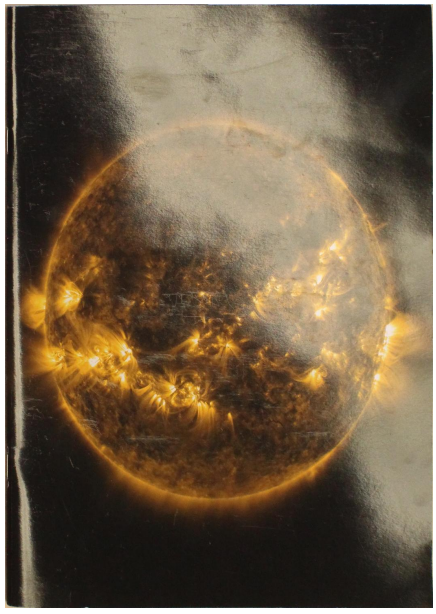
Image label	Original Image	Output
IMG_0080.JPG		
IMG_0081.JPG		
IMG_0082.JPG		

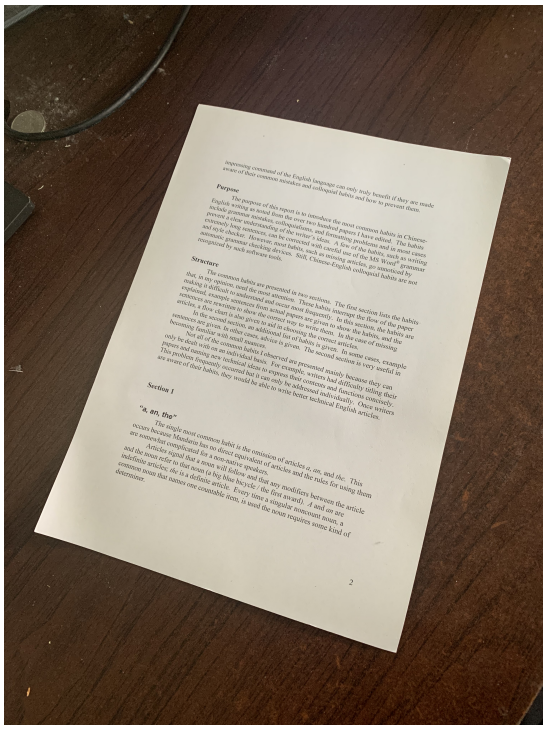
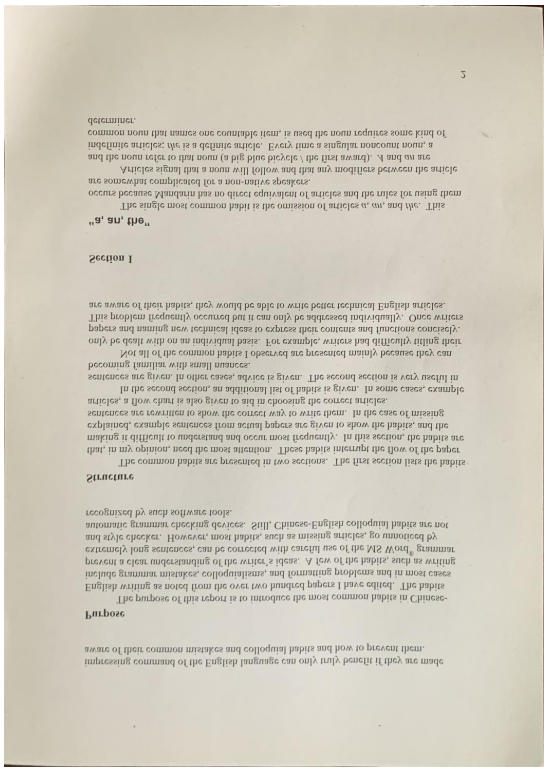
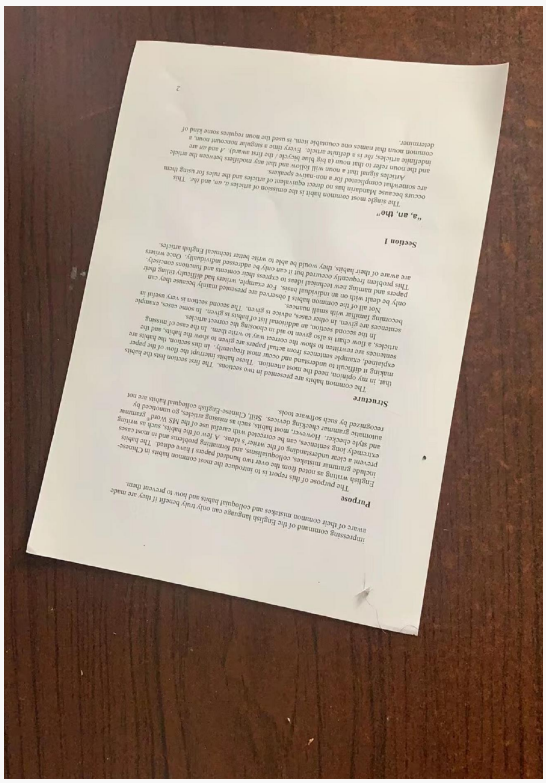
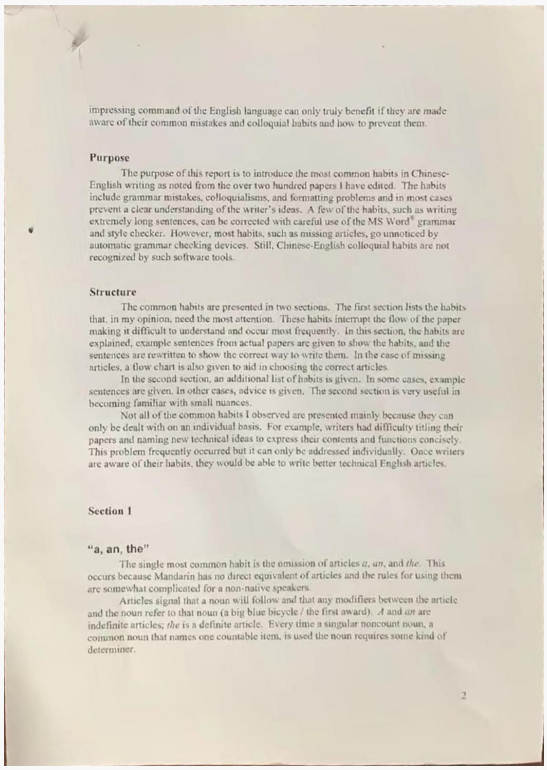
Image label	Original Image	Output
IMG_0084.JPG		
IMG_0085.JPG		

My own images Demo

```
clear; close all; clc; format shortG;

%% Parameter
% A4 paper:  210mm×297mm
bookletSize = [21 29.7]; % 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));
    corner = FindCorner(img); % find ordered four corners

```

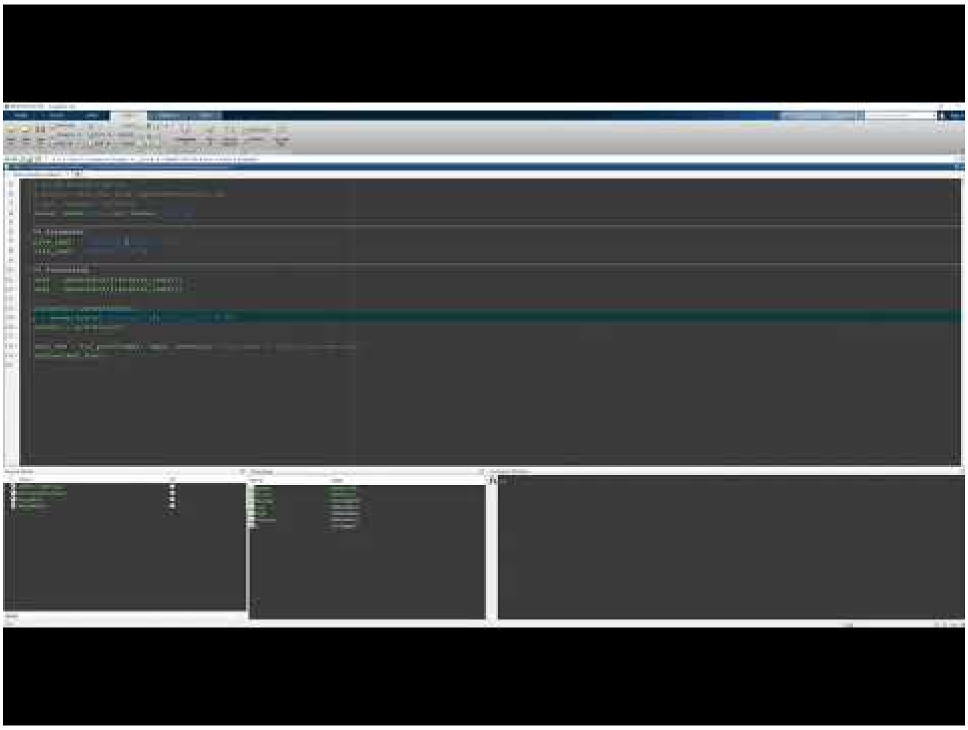
```
    % compute a homography
    H = ComputeH(corner, bookletImgSize);

    [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

function Corner = FindCorner(img)

% Change to Gray image
img = rgb2gray(img);
img = imgaussfilt(img,3);

% edge function
BW = edge(img, 'canny', [0.1 0.3]);
rho_res = 0.6;
theta_rng = -90:0.5:89;

% hough function
[H,T,R] = hough(BW,'RhoResolution',rho_res,'Theta',theta_rng);
 % find peaks in the hough matrix
p = houghpeaks(H,4,'Threshold',0.1*max(H(:)));

% extract line
lines = houghlines(BW,T,R,p);
pk_lines = [lines.theta;lines.rho];

for ii=1:size(pk_lines,2)
    theta = pk_lines(1,ii);
    rho = pk_lines(2,ii);
    m(ii) = -cosd(theta)/sind(theta);
    b(ii) = rho/sind(theta);
end
[m_uni,ind_uni] = uniquetol(m,0.0001);
b_uni = b(ind_uni);

% Line representation in HC
edges = (-1)*ones(3,numel(m_uni));
edges(1,:) = m_uni; edges(3,:) = b_uni;

% Find points of intersection
pts_size = nchoosek(size(edges,2),2);
pts = zeros(3,pts_size);
count = 1;
for ii = 1:size(edges,2)
    for jj = (ii+1):size(edges,2)
        pts(:,count) = cross(edges(:,ii),edges(:,jj));
        count = count+1;
    end
end
% Points in R2
pts_out = zeros(2,size(pts,2));
pts_out(1,:) = pts(1,:)./pts(3,:);
pts_out(2,:) = pts(2,:)./pts(3,:);
ii = 1;
while ii<=size(pts_out,2)
```

```
    if any(pts_out(:,ii)<0)||any(pts_out(:,ii)>size(img,2))
        pts_out(:,ii)=[];
        ii = ii;
    else
        ii = ii+1;
    end
end
% Order of the points: bottom left->top left->top right->bottom right
pt1 = pts_out(:,1); pt2 = pts_out(:,2); pt3 = pts_out(:,3);
dist1 = sqrt((pt1(1)-pt2(1))^2+(pt1(2)-pt2(2))^2);
dist2 = sqrt((pt1(1)-pt3(1))^2+(pt1(2)-pt3(2))^2);
if dist1>dist2
    Corner = pts_out(:,[2 4 3 1]);
else
    Corner = pts_out(:,[4 3 1 2]);
end
Corner = Corner';
end
```

Original Image	Output
	
	

Problem 5: Hough Transform & Image Overlay (30 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 `fun_prob5` function. A sample code is provided in **problem5**. You can reuse any code that you wrote for solving Problem 4 in Task 06 or Problem 5 or 6 in Task 5. Again, you only remove perspective distortion, meaning that resulting images could be horizontally or vertically flipped.

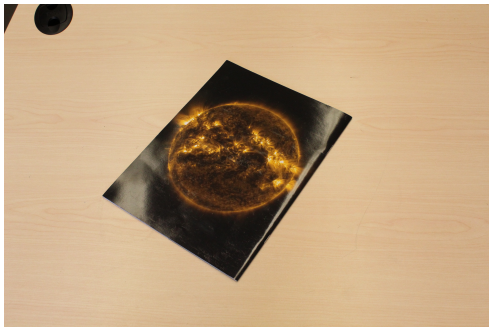
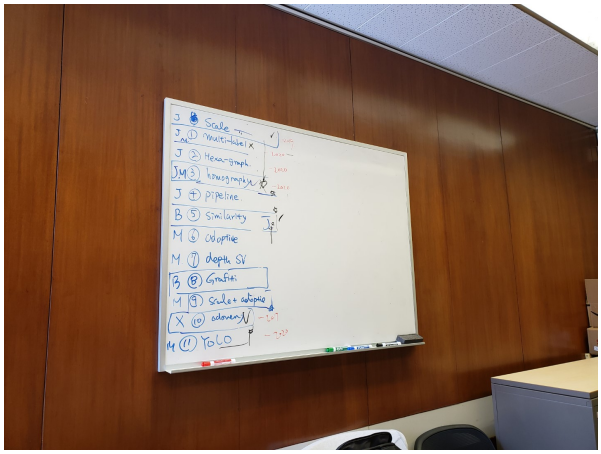
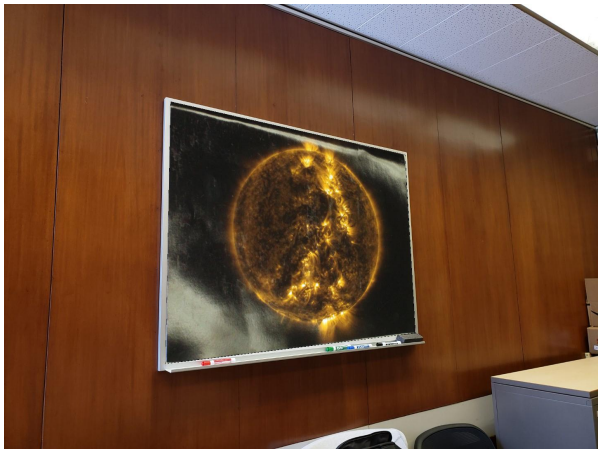
Next, you need to demonstrate the performance using your images captured in Problem 4 and overlay into a rectangle area. Thus, you need to take an image where a planar rectangular region is captured, like the whiteboard image above. Please include both results and codes in your submission.

```
clear; close all; clc; format shortG;

%% Parameter
file_img1 = '20200227_104101.jpg';
file_img2 = 'IMG_0081.JPG';

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

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!
imshow(img2_new);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function imgFinal = fun_prob5(imgBoard, img, boardcorner)

bookletSize = [24 31.5]; % cm
bookletImgSize = bookletSize*50; % output image siz
corners = FindCorner(img);
```

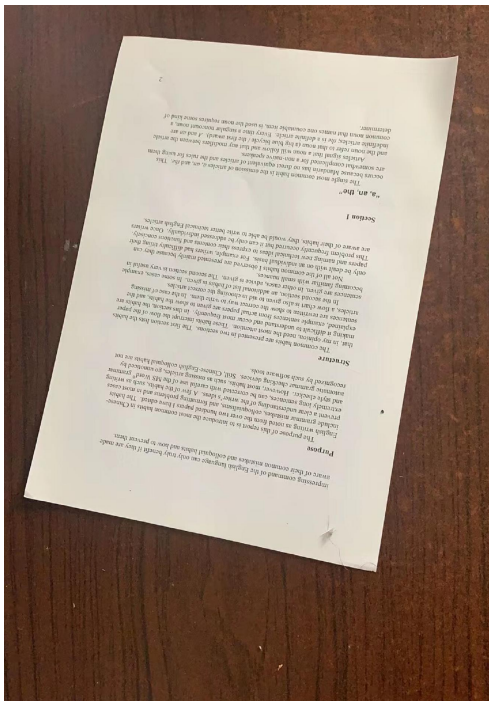
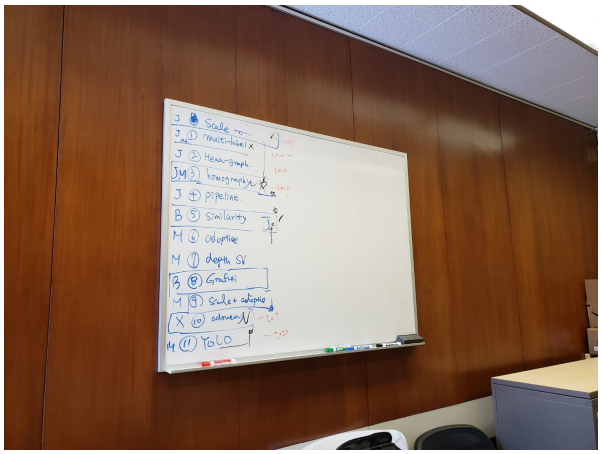
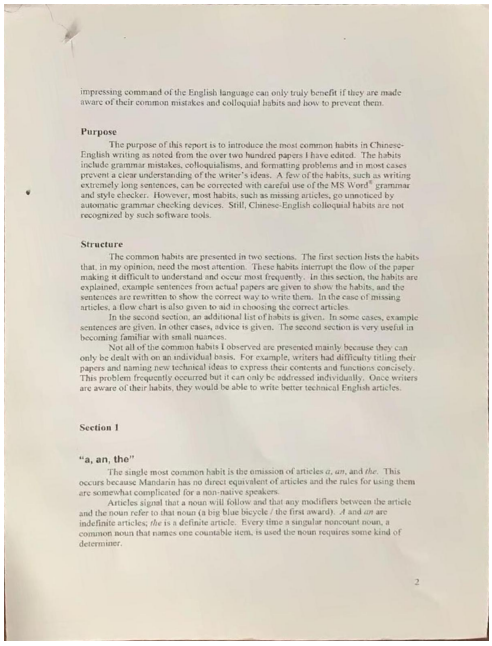
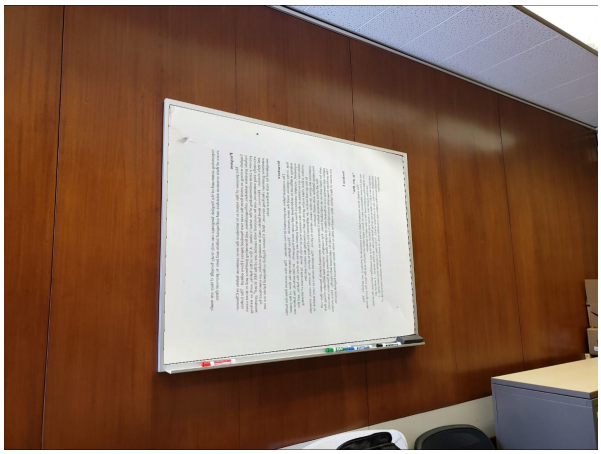
```
H = ComputeH(corners, bookletImgSize); % compute a homography
[imgTran, RA] = imwarp(img, projective2d(inv(H)));
bookletImg = imcrop(imgTran, [-RA.XWorldLimits(1), -RA.YWorldLimits(1)
bookletImgSize]);

% figure(1);
% imshow(bookletImg);
imwrite(bookletImg, '5_1.jpg');
% Overlay your picture
sizePic = [size(bookletImg,1), size(bookletImg,2)];
H2 = ComputeH(boardcorner, sizePic);
[imgPicTran, RB] = imwarp(bookletImg, projective2d(inv(H2)));
BWPic = roipoly(imgPicTran, boardcorner(:,1)-RB.XWorldLimits(1),
boardcorner(:,2)-RB.YWorldLimits(1));
BWBoard = ~roipoly(imgBoard, boardcorner(:,1), boardcorner(:,2));
RA = imref2d(size(BWBoard));
imgBoardMask = bsxfun(@times, imgBoard, cast(BWBoard, 'like', imgBoard));
imgPicTranMask = bsxfun(@times, imgPicTran, cast(BWPic, 'like',
imgPicTran));
imgFinal(:,:,1) = imfuse(imgBoardMask(:,:,1),RA,
imgPicTranMask(:,:,1),RB,'diff');
imgFinal(:,:,2) = imfuse(imgBoardMask(:,:,2),RA,
imgPicTranMask(:,:,2),RB,'diff');
imgFinal(:,:,3) = imfuse(imgBoardMask(:,:,3),RA,
imgPicTranMask(:,:,3),RB,'diff');
% figure(2);
% imshow(imgFinal);
imwrite(imgFinal, '5_2.jpg');
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function H = ComputeH(corner, sizePic)
points1 = [0,0;sizePic(1),0;sizePic(1),sizePic(2);0,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);
Tmpt = zeros(size(points1,1)*2,9);
Tmpt(1:2:end,3) = 1;
Tmpt(2:2:end,6) = 1;
Tmpt(1:2:end,1:2) = points1;
Tmpt(2:2:end,4:5) = points1;
Tmpt(1:2:end,7) = -x_xp;
Tmpt(1:2:end,8) = -y_xp;
Tmpt(2:2:end,7) = -x_yp;
Tmpt(2:2:end,8) = -y_yp;
Tmpt(1:2:end,9) = -points2(:,1);
Tmpt(2:2:end,9) = -points2(:,2);
T = null(Tmpt);
T = T/T(end);
H = (reshape(T,3,3));
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function Corner = FindCorner(img)

% Change to Gray image
img = rgb2gray(img);
```

```
img = imgaussfilt(img,3);

% edge function
BW = edge(img, 'canny', [0.2 0.3]);
rho_res = 0.6;
theta_rng = -90:0.5:89;
% hough function
[H,T,R] = hough(BW,'RhoResolution',rho_res,'Theta',theta_rng);
 % find peaks in the hough matrix
p = houghpeaks(H,4,'Threshold',0.1*max(H(:)));

% extract line
lines = houghlines(BW,T,R,p);
pk_lines = [lines.theta;lines.rho];

for ii=1:size(pk_lines,2)
    theta = pk_lines(1,ii);
    rho = pk_lines(2,ii);
    m(ii) = -cosd(theta)/sind(theta);
    b(ii) = rho/sind(theta);
end
[m_uni,ind_uni] = uniquetol(m,0.0001);
b_uni = b(ind_uni);
% Line representation in HC
edges = (-1)*ones(3,numel(m_uni));
edges(1,:) = m_uni; edges(3,:) = b_uni;
% Find points of intersection
pts_size = nchoosek(size(edges,2),2);
pts = zeros(3,pts_size);
count = 1;
for ii = 1:size(edges,2)
    for jj = (ii+1):size(edges,2)
        pts(:,count) = cross(edges(:,ii),edges(:,jj));
        count = count+1;
    end
end
% Points in R2
pts_out = zeros(2,size(pts,2));
pts_out(1,:) = pts(1,:)./pts(3,:);
pts_out(2,:) = pts(2,:)./pts(3,:);
ii = 1;
while ii<=size(pts_out,2)
    if any(pts_out(:,ii)
<0)||any(pts_out(:,ii)>max(size(img,2),size(img,1)))
        pts_out(:,ii)=[];
        ii = ii;
    else
        ii = ii+1;
    end
end
% Order of the points: bottom left->top left->top right->bottom right
pt1 = pts_out(:,1); pt2 = pts_out(:,2); pt3 = pts_out(:,3);
dist1 = sqrt((pt1(1)-pt2(1))^2+(pt1(2)-pt2(2))^2);
dist2 = sqrt((pt1(1)-pt3(1))^2+(pt1(2)-pt3(2))^2);
if dist1>dist2
    Corner = pts_out(:,[2 4 3 1]);
else
    Corner = pts_out(:,[4 3 1 2]);
```

```
133 end
134 Corner = Corner';
135 end
```

IMG_0081.JPG	20200227_104101.jpg
	
Output1	**Final resuit**
	

My own demo:

IMG_0081.JPG	20200227_104101.jpg
	
Output1	**Final resuit**
	

impressing command of the English language can only truly benefit if they are made aware of their common mistakes and colloquial habits and how to prevent them.

Purpose

The purpose of this report is to introduce the most common habits in Chinese-English writing as noted from the over two hundred papers I have edited. The habits include grammar mistakes, colloquialisms, and formatting problems and in most cases prevent a clear understanding of the writer's ideas. A few of the habits, such as writing extremely long sentences, can be corrected with careful use of the MS Word® grammar and style checker. However, most habits, such as missing articles, go unnoticed by automatic grammar checking devices. Still, Chinese-English colloquial habits are not recognized by such software tools.

Structure

The common habits are presented in two sections. The first section lists the habits that, in my opinion, need the most attention. These habits interrupt the flow of the paper making it difficult to understand and occur most frequently. In this section, the habits are explained, example sentences from actual papers are given to show the habits, and the sentences are rewritten to show the correct way to write them. In the case of missing articles, a flow chart is also given to aid in choosing the correct articles.

In the second section, an additional list of habits is given. In some cases, example sentences are given, in other cases, advice is given. The second section is very useful in becoming familiar with small nuances.

Not all of the common habits I observed are presented mainly because they can only be dealt with on an individual basis. For example, writers had difficulty titling their papers and naming new technical ideas to express their contents and functions concisely. This problem frequently occurred but it can only be addressed individually. Once writers are aware of their habits, they would be able to write better technical English articles.

Section 1

"a, an, the"

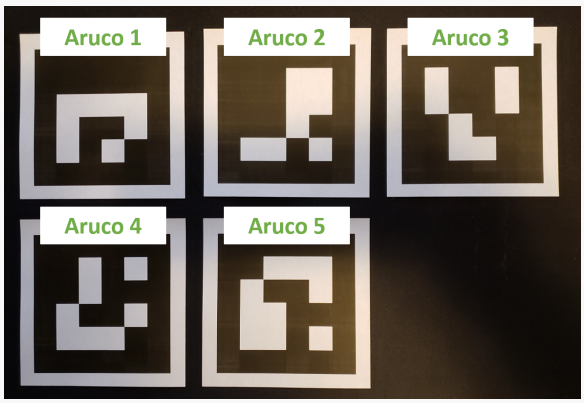
The single most common habit is the omission of articles *a, an,* and *the*. This occurs because Mandarin has no direct equivalent of articles and the rules for using them are somewhat complicated for a non-native speakers.

Articles signal that a noun will follow and that any modifiers between the article and the noun refer to that noun (a big blue bicycle / the first award). *A* and *an* are indefinite articles; *the* is a definite article. Every time a singular noncount noun, a common noun that names one countable item, is used the noun requires some kind of determiner.

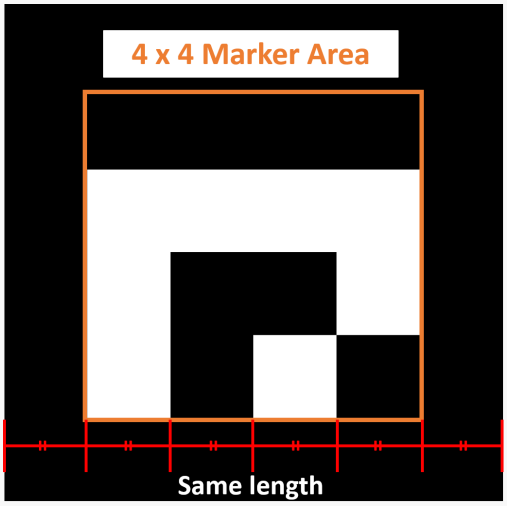
2

Problem 6: Fiduciary Marker Detection (40 points)

You are going to build a marker identification tool. Aruco marker (Augmented Reality University of Cordoba) is a popular fiducial marker widely used for augmented reality applications. Acruco marker is a binary square with black background and white generated pattern. Each marker has own unique id ranging from 0 to 999. The marker code is unique under a rotated environment. For example, although you rotate the marker 90 degrees, the code (white) pattern does not overlap with the ones from the rest of the 998 codes. I generated five 4 x 4 Aruco marker from here where ID is from 1 to 5 and print them.



(a) Five Aruco markers used in this problem

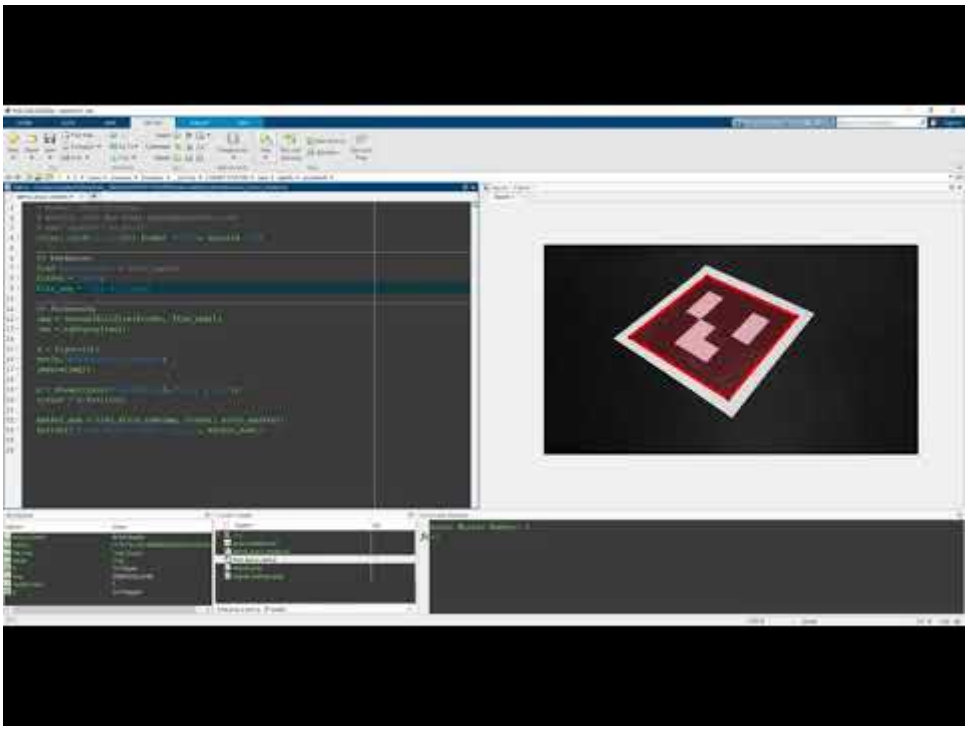


(b) a 4 x 4 maker design

The objective of the tool is to identify the id of a marker on a given image. You are going to select the outline of the marker and your tool identify the id of the maker.

A total of 7 images are given and you need to estimate the ids for the markers on these images. The Aruco marker binary pattern matrices for these markers are provided in `aruco_marker.mat`. You need to use this matrix to identify the id of the marker on each test image.

Here is a sample demo.



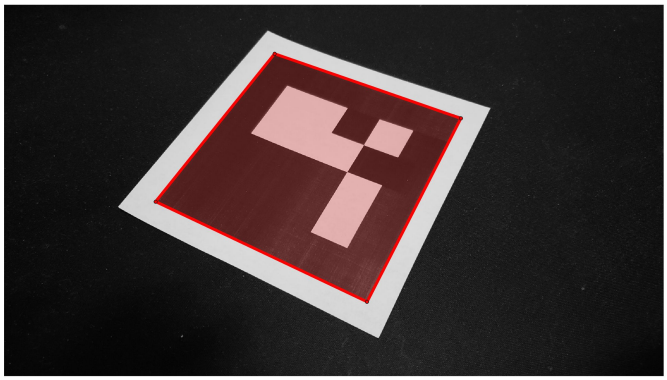
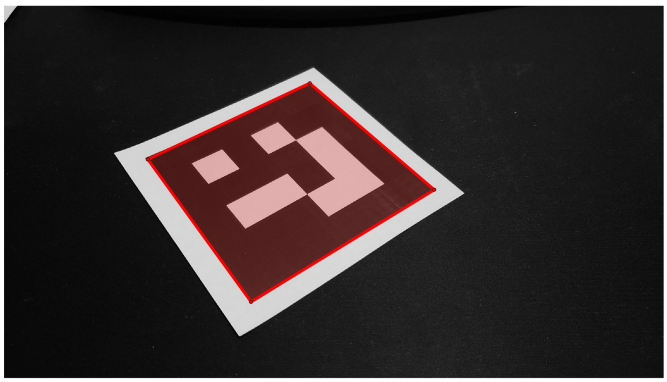
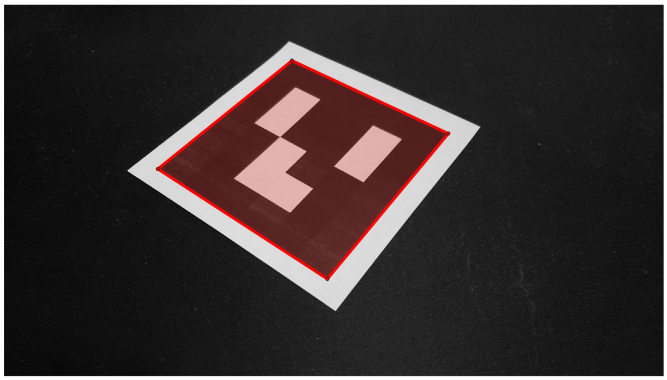
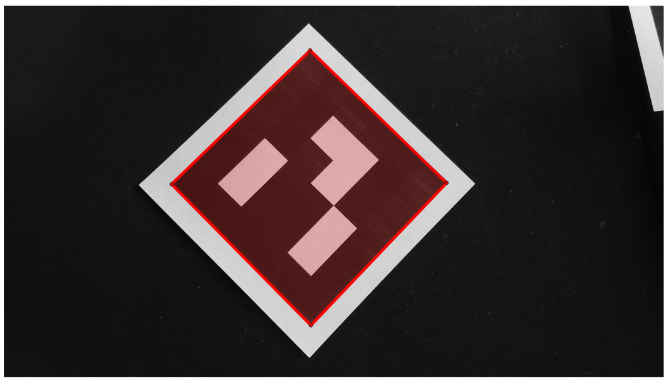
A sample script is provided in this problem. You are going to make your own `find_aruco_num` script, which is given as p-code. If you are a Python user, you need to make an equivalent function to identify marker ids. To do so, you need to import `aruco_marker.mat` to your Python script. Note that you should not use any computer vision library (including marker detection library) in MATLAB or Python. You can solve this using `ComputeH` that you made and some basic functions in MATLAB or Python. Note that this is not a feature matching problem so do not try to match features on the marker on the test image and original marker image.

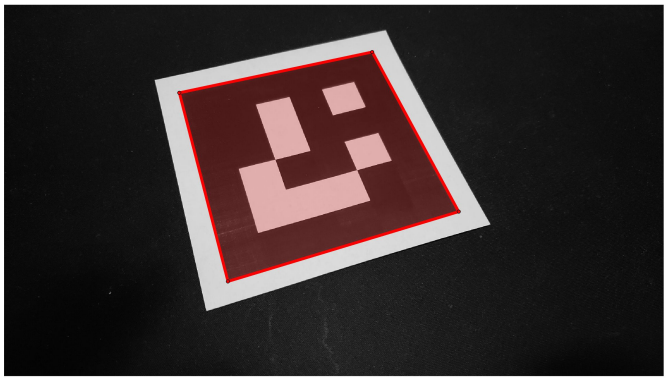
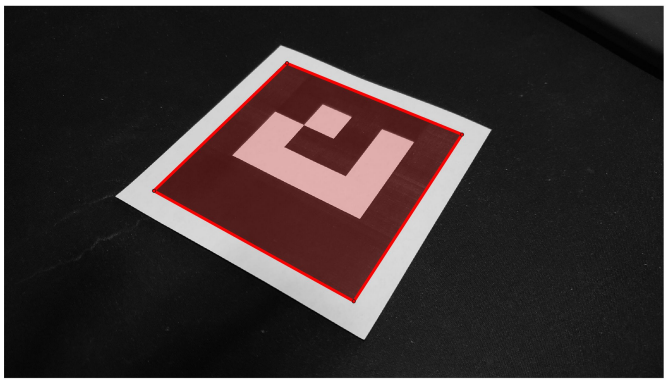
Note that if you can complete problems 4,5 and 6 using Python, you will receive extra 5 points.

```
clear; close all; clc; format shortG; warning off;

%% Parameter
load aruco_marker % load data
folder = 'img';
file_img = 'img (7).jpg';

%% Processing
img = imread(fullfile(folder, file_img));
img = rgb2gray(img);

h = figure(1);
% Block this for better aoto-saving
% set(h,'WindowStyle','docked')
imshow(img);

p = drawpolygon('LineWidth',5,'Color','red');
corner = p.Position;

marker_num = find_aruco_num(img, corner, aruco_marker);
fprintf('Aruco Marker Number: %d \n', marker_num);
saveas(gcf,string(marker_num)+'-'+file_img);



function marker_num = find_aruco_num(img, corner, aruco_marker)

% Initial homography 60*60
IniSize = [60 60];
H = ComputeH(corner,IniSize);
[imgTran, RA] = imwarp(img, projective2d(inv(H)));
ThisImg = imcrop(imgTran, [-RA.XWorldLimits(1), -RA.YWorldLimits(1)
IniSize]);

% Using a sharpening filter
ThisImg = im2double(ThisImg);
Hs = (-0.15)*ones(3,3);
Hs(2,2) = 3;
ThisImg = conv2(ThisImg,Hs,'same');

% Resize the ThisImg to a 7x7 pixel and Estimate the value
ThisImg = imresize(ThisImg,0.1);
index = (ThisImg>=1.0);
ThisImg(index) = 1;
ThisImg(~index) = 0;

% Remove boundary of ThisImg
ThisImg([1,end-1, end],:)=[];
ThisImg(:,[1,end-1,end])=[];

% Identify marker number
marker_num=0;%default results for no maching cases
```

```
for ii = 1:size(aruco_marker,3)
    for jj = 0:3
        if rot90(ThisImg,jj) == aruco_marker(:,:,ii)
            marker_num = ii;
            break
        end
    end

end
end
```

Results:

Aruco marker number	Image Name	Image
2	img (1).jpg	
4	img (2).jpg	
3	img (3).jpg	
3	img (4).jpg	

Aruco marker number	Image Name	Image
4	img (5).jpg	
1	img (6).jpg	
5	img (7).jpg