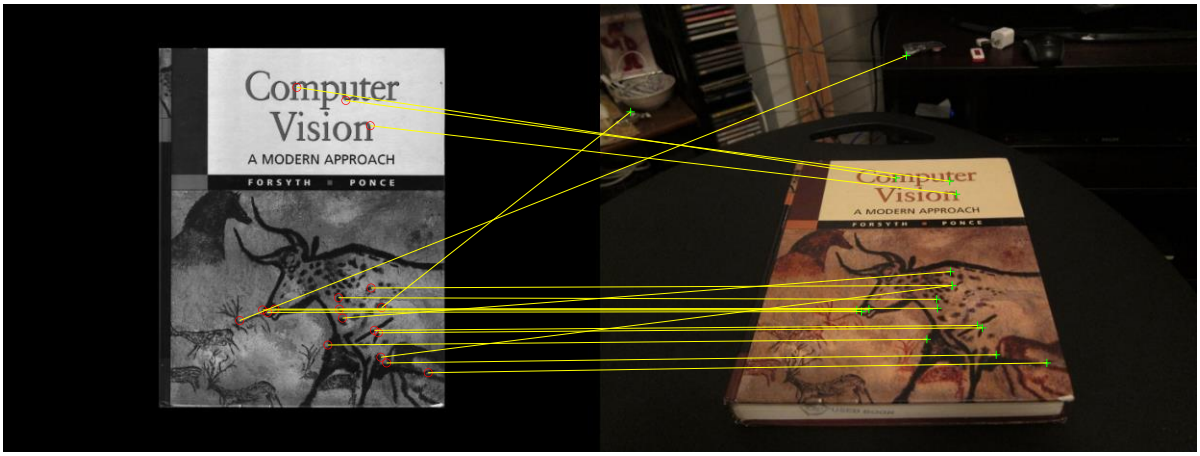
Project 4

Huiyi Zou
301355563

4.1. Feature Detection, Description, and Matching

```
function [locs1, locs2] = matchPics( I1, I2 )
%MATCHPICS Extract features, obtain their descriptors, and match them!
%% Convert images to grayscale, if necessary
gray_I1 = I1;
gray_I2 = I2;
if ndims(gray_I1) == 3
    gray_I1 = rgb2gray(I1);
end
if ndims(gray_I2) == 3
    gray_I2 = rgb2gray(I2);
end
%% Detect features in both images
points1 = detectFASTFeatures(gray_I1);
points2 = detectFASTFeatures(gray_I2);
%% Obtain descriptors for the computed feature locations
[desc1, locs1] = computeBrief(gray_I1, points1.Location);
[desc2, locs2] = computeBrief(gray_I2, points2.Location);
%% Match features using the descriptors
pairs = matchFeatures(desc1, desc2, 'MatchThreshold', 10.0, 'MaxRatio',0.7);
locs1 = locs1(pairs(:,1),:);
locs2 = locs2(pairs(:,2),:);
% figure;
% showMatchedFeatures(I1, I2, locs1, locs2, 'montage');
end
```

I used *ndims* to get the images' dimensions and used $rgb2gray$ function for grayscale conversion.
The $MatchThreshold$ parameter on $matchFeatures$ was set to be 10.0 for our binary BRIEF descriptor. I increased the $MaxRatio$ parameter to 0.7.



4.2. BRIEF and Rotations

The *briefRotTest.m* file:

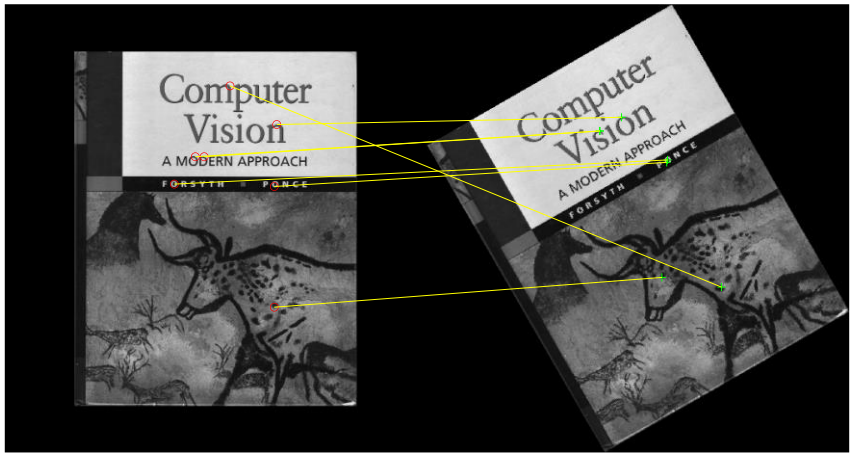
```
%% Read the image and convert to grayscale, if necessary
cv_img = imread('../data/cv_cover.jpg');
if ndims(cv_img) == 3
    cv_img = rgb2gray(cv_img);
end
%% get matched result
matches1 = [];
for i = 0:36
    %% Rotate image
    rotated_img = imrotate(cv_img, 10*i);
    %% Extract features and match
    [locs1, locs2] = matchPics(cv_img, rotated_img);
    if i==3 || i==6 || i==9
        figure;
        showMatchedFeatures(cv_img,rotated_img, locs1, locs2, 'montage');
    end
    %% Update histogram
    matches1 = [matches1 size(locs1,1)];
end
%% Display histogram
figure;
bar(0:10:360, matches1);
xlabel('Rotation of Degree')
ylabel('Number of Matched Features')
title('The Plot of Histogram with BRIEF Descriptor')
```

```
%% Using detectSURFFeatures and extractFeatures
points = detectSURFFeatures(cv_img);
[desc1, locs] = extractFeatures(cv_img, points.Location, 'Method', 'SURF');
matches2 = [];
for i = 0:36
    rotated_img = imrotate(cv_img, 10*i);
    rotated_points = detectSURFFeatures(rotated_img);
    [desc2, locs2] = extractFeatures(rotated_img, rotated_points.Location, 'Method', 'SURF');
    pairs = matchFeatures(desc1, desc2, 'MatchThreshold', 10.0, 'MaxRatio',0.7);
    locs1 = locs(pairs(:,1),:);
    locs2 = locs2(pairs(:,2),:);
    if i==3 || i==6 || i==9
        figure;
        showMatchedFeatures(cv_img,rotated_img,locs1, locs2, 'montage');
    end
    matches2 = [matches2 size(locs1,1)];
end
%%  Display histogram
figure;
bar(0:10:360, matches2);
xlabel('Rotation of Degree')
ylabel('Number of Matched Features')
title('The Plot of Histogram with SURF Detector')
```

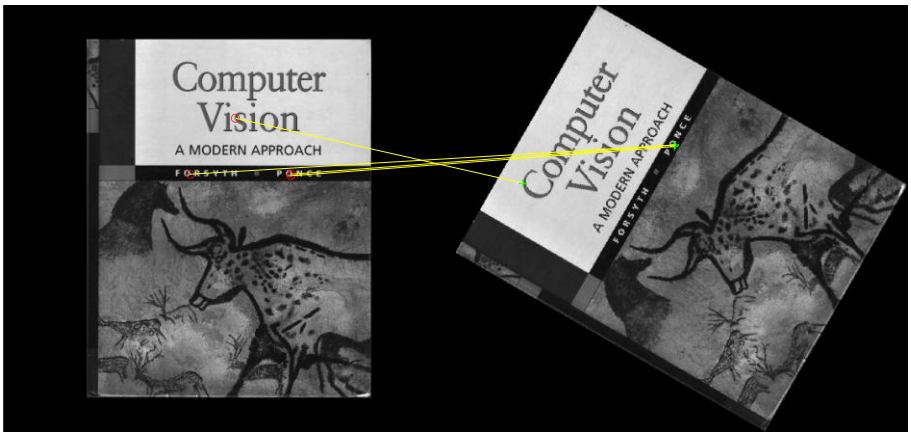
I kept the same parameters on *matchFeatures* functions when using *detectSURFFeatures* and *extractFeatures*. And I visualized the feature matching results at rotation degree 30, 60, and 90.

- **With BRIEF descriptor**

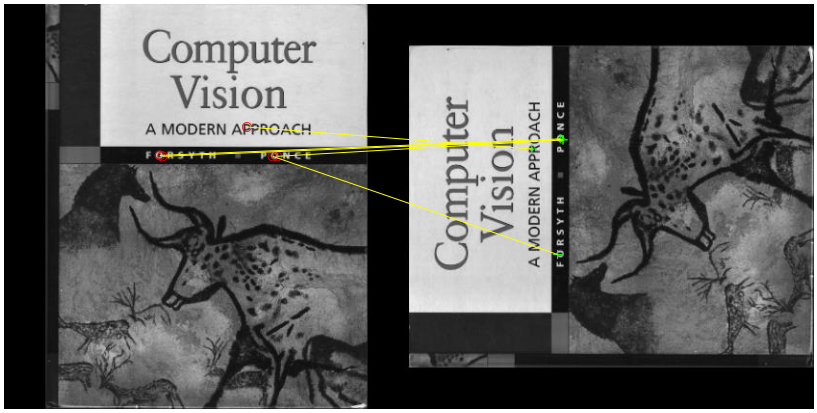
  Rotation of 30 degree:



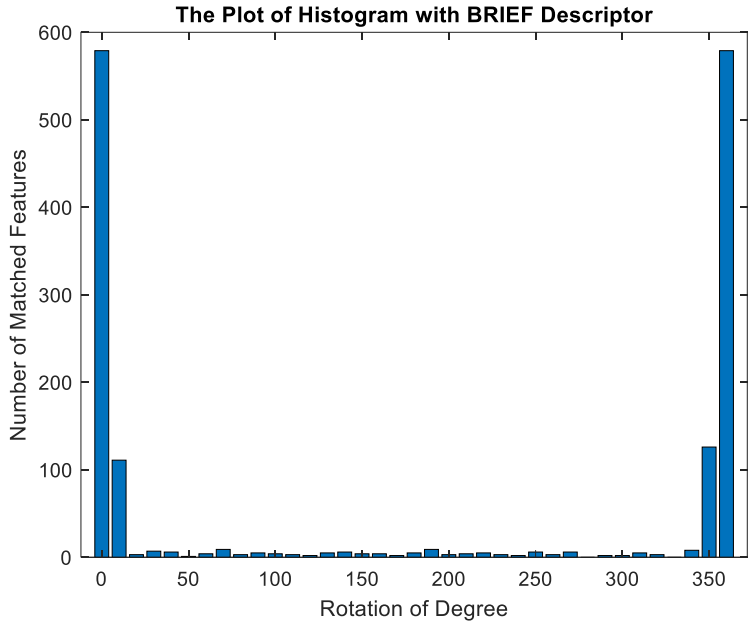
  Rotation of 60 degree



  Rotation of 90 degree:



The histogram:



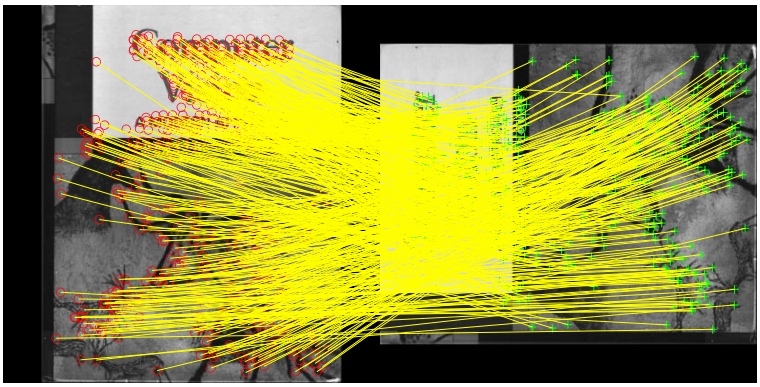
The number of matched features between the original image and itself is almost 600. The number of matched features between the original image and the image rotated is much less than the result from the image without rotation. It is because BRIEF is not invariant to rotation. The rotation will affect the matching result.

- **With SURF**

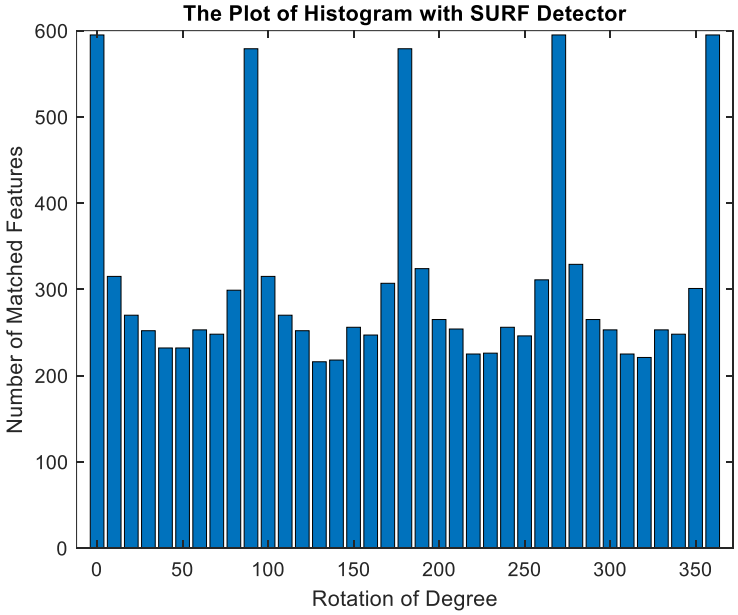
  Rotation of 30 degree:

Rotation of 60 degree

Rotation of 90 degree:



The histogram:



When the image is rotated, the SURF can get more feature matched than using the BRIEF. There are lots of matched features between the original image and the image rotated by the multiple of 90 degrees. It has a better behavior than the BRIEF descriptor, though the rotation of other degrees will reduce the number of matched features.

4.3. Homography Computation

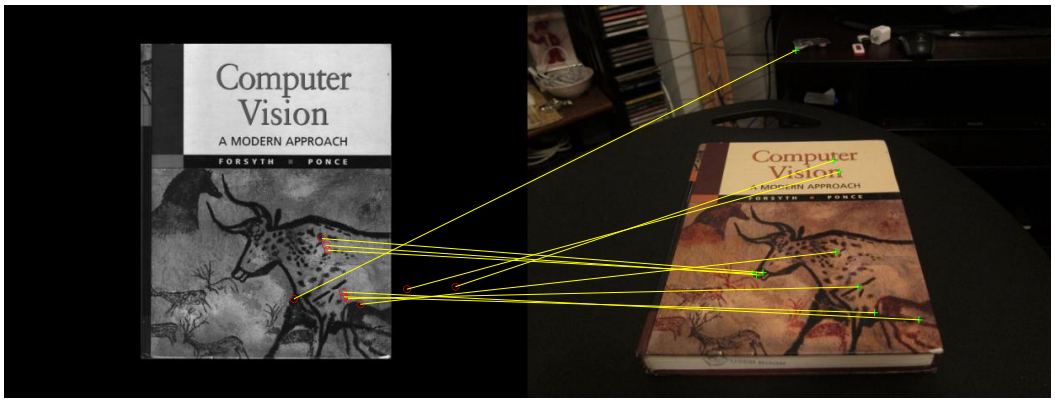
```
function [ H2to1 ] = computeH( x1, x2 )
%COMPUTEH Computes the homography between two sets of points
A = [];
for i = 1:size(x1,1)
    temp = [-x2(i,1),-x2(i,2),-1,0,0,0,x2(i,1)*x1(i,1),x2(i,2)*x1(i,1),x1(i,1);
        0,0,0,-x2(i,1),-x2(i,2),-1,x2(i,1)*x1(i,2),x2(i,2)*x1(i,2),x1(i,2)];
    A = [A; temp];
end

[~,~,V] = svd(A);
H2to1 = reshape(V(:,9),3,3).';
end
```

I used svd to get the best homography $H2to1$ for mapping the x2 to the x1. The $reshape$ function and the transpose operator reshaped a column vector form of $H2to1$ to a 3×3 matrix and made the matrix be the correct orientation.

Randomly 10 points from x2 and the corresponding locations in the x1 after the homography transformation.

```
%% visualize computeH
H2to1 = computeH(locs1, locs2);
n = randperm(size(locs2,1),10);
pre_x1 = [];
for i = 1:10
    temp = H2to1*[locs2(n(i),:)'; 1];
    pre_x1 = [pre_x1;(temp/temp(3))'];
end
figure;
showMatchedFeatures(cv_img, desk_img, pre_x1(:,1:2), locs2(n, :), 'montage');
```



4.4. Homography Normalization

```
function [H2to1] = computeH_norm(x1, x2)
%% Compute centroids of the points
centroid1 = mean(x1);
centroid2 = mean(x2);
%% Shift the origin of the points to the centroid
shifted_x1 = x1 - centroid1;
shifted_x2 = x2 - centroid2;
%% Normalize the points so that the average distance from the origin is equal to sqrt(2).
avg_distance1 = mean(hypot(shifted_x1(:,1),shifted_x1(:,2)));
avg_distance2 = mean(hypot(shifted_x2(:,1),shifted_x2(:,2)));
s1 = sqrt(2) / avg_distance1;
s2 = sqrt(2) / avg_distance2;
Normalized_x1 = shifted_x1 * s1;
Normalized_x2 = shifted_x2 * s2;
%% similarity transform 1
T1 = [s1, 0, -centroid1(1)*s1; 0, s1, -centroid1(2)*s1; 0, 0, 1];
%% similarity transform 2
T2 = [s2, 0, -centroid2(1)*s2; 0, s2, -centroid2(2)*s2; 0, 0, 1];
%% Compute Homography
H = computeH(Normalized_x1, Normalized_x2);
%% Denormalization
H2to1 = T1\H*T2;
```

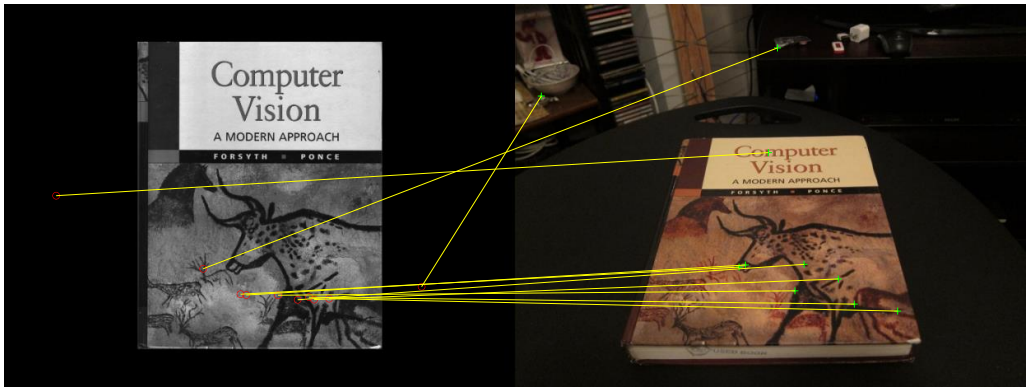
I referred to this MATLAB answers to compute the average distance:
https://www.mathworks.com/matlabcentral/answers/215062-average-distance-from-the-origin
Since the average distance should be $\sqrt{2}$ after scaling, for a point (x, y), we have:

$$\sqrt{(sx)^2 + (sy)^2} = \sqrt{2}$$
$$s^2x^2 + s^2y^2 = 2$$
$$s = \pm\sqrt{\frac{2}{x^2 + y^2}}$$

Thus, I let $s = \sqrt{2}/avg\_distance$.

Randomly 10 points from x2 and the corresponding locations in the x1 after the homography transformation.

```
%% visualize computeH_norm
H2to1 = computeH_norm(locs1, locs2);
n = randperm(size(locs2,1),10);
pre_x1 = [];
for i = 1:10
    temp = H2to1*[locs2(n(i),:)'; 1];
    pre_x1 = [pre_x1;(temp/temp(3))'];
end
figure;
showMatchedFeatures(cv_img, desk_img, pre_x1(:,1:2), locs2(n, :), 'montage');
```



4.5. RANSAC

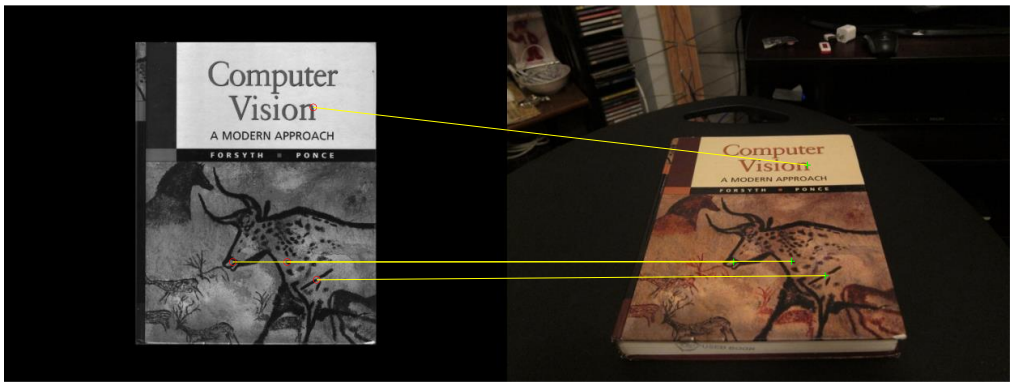
```
function [ bestH2to1, inliers, best_n] = computeH_ransac( locs1, locs2)
%COMPUTEH_RANSAC A method to compute the best fitting homography given a
%list of matching points.
%Q2.2.3
threshold = 1;
max_count = 0;
inliers = [];
bestH2to1 = [];
best_n = [];
for iteration = 1:2000
    n = randperm(size(locs1,1),4);
    H = computeH_norm(locs1(n, :), locs2(n, :));
    temp_inliers = zeros(size(locs1,1),1);
    for i = 1:size(locs2,1)
        temp = H*[locs2(i,:)'; 1];
        temp = temp/temp(3);
        distance = sqrt((temp(1)-locs1(i,1))^2+(temp(2)-locs1(i,2))^2);
        if distance<=threshold
            temp_inliers(i) = 1;
        end
    end
    total = length(find(temp_inliers==1));
    if total>=max_count
        best_n = n;
        max_count = total;
        inliers = temp_inliers;
        bestH2to1 = H;
    end
end
end
```

I set 2000 iterations for the RANSAC loop, and let the distance threshold be 1, which means when the distance between an actual matched point and a computed point using $H2to1$ in the dest_img is less than 1, the point is inlier.

To visualize the 4 point-pairs that produced the most number of inliers, I also returned the variable $best\_n$, which stored the index of these 4 point-pairs

```
%% visualize computeH_ransac
[bestH2to1, inliers, best_n] = computeH_ransac(locs1, locs2);
figure;
showMatchedFeatures(cv_img, desk_img, locs1(best_n, :), locs2(best_n, :), 'montage');
pre_x1 = [];
n = find(inliers==1);
x2_inliers = locs2(n, :);
for i = 1:length(x2_inliers)
    temp = bestH2to1*[x2_inliers(i,:)'; 1];
    pre_x1 = [pre_x1;(temp/temp(3))'];
end
```

The 4 point-pairs:



The inlier matches that were selected by RANSAC algorithm:

4.6. HarryPotterizing a Book

The *HarryPotterize.m* file: (modified based on *HarryPotterize_auto.m*)

```
%Q2.2.4
clear all;
close all;

cv_img = imread('../data/cv_cover.jpg');
desk_img = imread('../data/cv_desk.png');
hp_img = imread('../data/hp_cover.jpg');

%% Extract features and match
[locs1, locs2] = matchPics(cv_img, desk_img);
%% Compute homography using RANSAC
[bestH2to1, inliers, best_n] = computeH_ransac(locs1, locs2);
%% Scale harry potter image to template size
% Why is this is important?
scaled_hp_img = imresize(hp_img, [size(cv_img,1) size(cv_img,2)]);
%% Display warped image.
imshow(warpH(scaled_hp_img, inv(bestH2to1), size(desk_img)));
%% Display composite image
imshow(compositeH(bestH2to1, scaled_hp_img, desk_img));
```

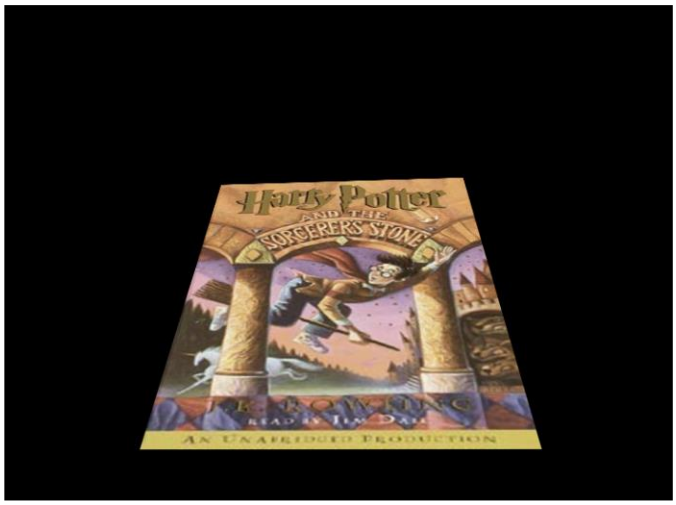
Since the homography computed was from the image to the template, the homography should be inverted for warping the template to the image.

The function *compositeH*:

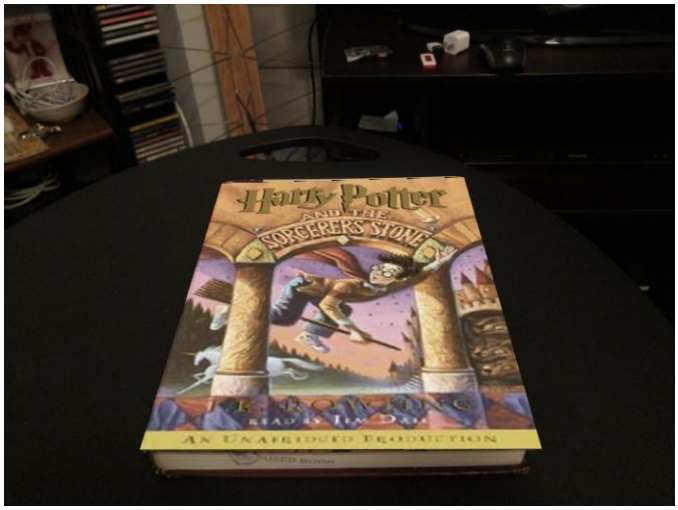
```
function [ composite_img ] = compositeH( H2to1, template, img )
%COMPOSITE Create a composite image after warping the template image on top
%of the image using the homography
% Note that the homography we compute is from the image to the template;
% x_template = H2to1*x_photo
% For warping the template to the image, we need to invert it.
H_template_to_img = inv(H2to1);
%% Create mask of same size as template
mask = ones(size(template,1),size(template,2),size(template,3));
%% Warp mask by appropriate homography
mask = warpH(mask, H_template_to_img, size(img));
%% Warp template by appropriate homography
template = warpH(template, H_template_to_img, size(img));
%% Use mask to combine the warped template and the image
composite_img = img;
composite_img(mask ~= 0) = template(mask ~= 0);
end
```

For the part for using mask to combine the warped template and the image, I referred to the answers from this link: https://www.mathworks.com/matlabcentral/answers/319311-how-to-combine-image-with-mask

The warped image



The composite image



5. Creating your Augmented Reality application

The $ar.m$ file:

```
% Q3.3.1
clear all;
close all;
cv_img = imread('../data/cv_cover.jpg');
book = loadVid('../data/book.mov');
source = loadVid('../data/ar_source.mov');
video = VideoWriter('../result/ar.avi');
open(video);
for f = 1:length(source)
    book_img = book(f).cdata;
    [locs1, locs2] = matchPics(cv_img, book_img);

    try
        [bestH2to1, ~, ~] = computeH_ransac(locs1, locs2);
        prev_H2to1 = bestH2to1;
    catch
        bestH2to1 = prev_H2to1;
    end

    source_img = source(f).cdata;
    source_img = source_img(50:310,210:430,:);
    scaled_cource_img = imresize(source_img, [size(cv_img,1) size(cv_img,2)]);
    img = compositeH(bestH2to1, scaled_cource_img, book_img);
    writeVideo(video, img);
end
close(video);
```

I extracted the sub-image from point (50, 210) to point (310, 430).

Since several frames which are blurred had no or few matched features for computing $bestH2to1$, there might be an error when called the $computeH\_ransac$ function, which means we could not get the $bestH2to1$ for that frame. If just increase the $MaxRatio$ parameter on $MatchPics$ function, there will be too many incorrect features matched. Therefore, I used the try and catch to call the $computeH\_ransac$ function and record the $bestH2to1$ in the try block. If an error occurs, the previous $bestH2to1$ will be used as the current $bestH2to1$.

I referred to this answer for saving the result video:
https://www.mathworks.com/matlabcentral/answers/280635-make-video-from-images