Basic Image Processing Functions

In image processing using Matlab We'll use the following basic image processing functions:

1. imread()

The **imread()** command will read an image into a matrix:

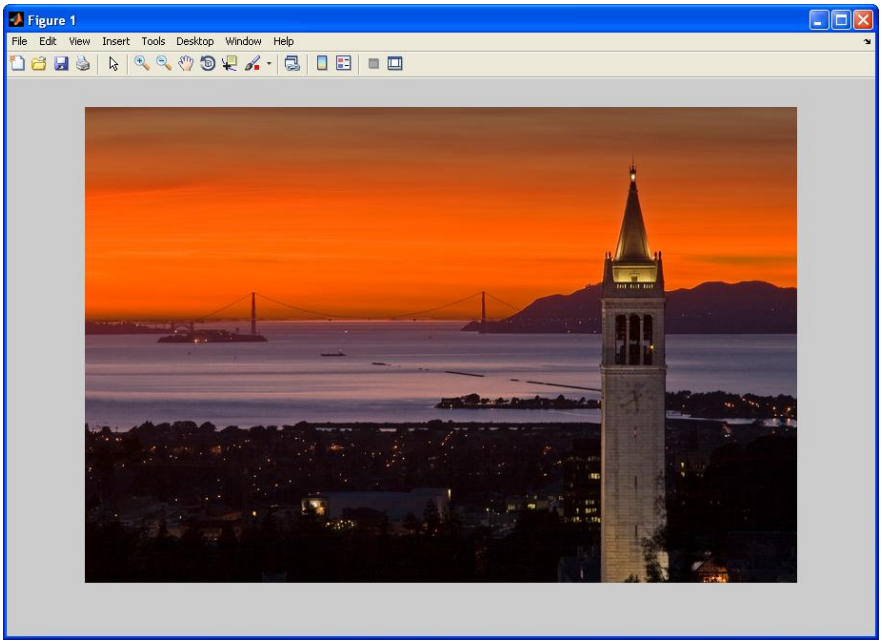
```
img = imread('ImageProcessing_1/BerkeleyTower.png');
>> size(img)

ans =
   499   748     3
```

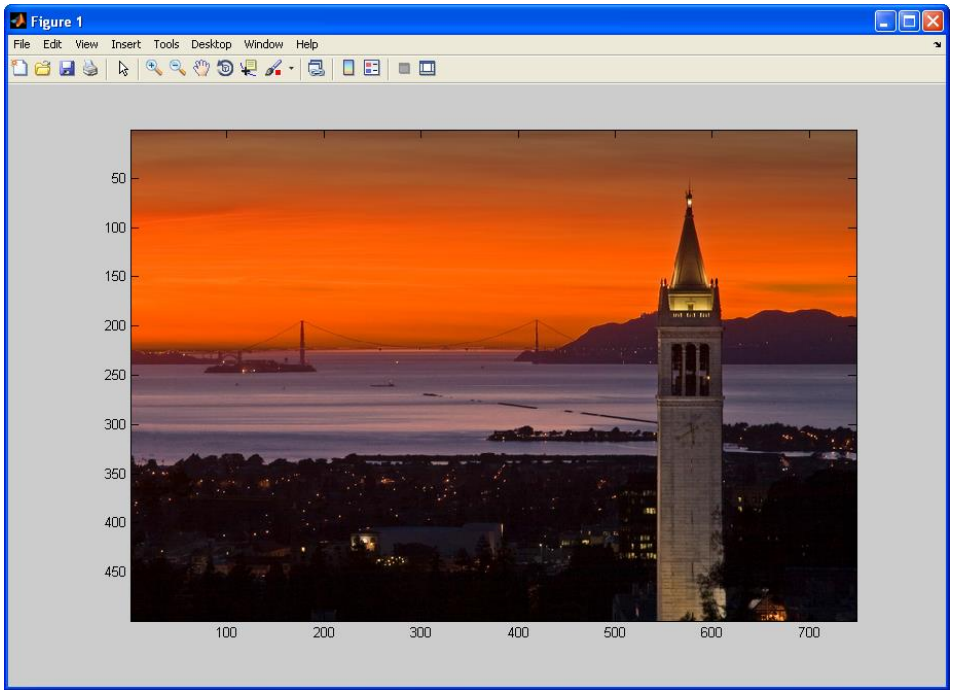
2. imshow()

To show our image, we use the **imshow()** or **imagesc()** command. The **imshow()** command shows an image in standard 8-bit format, like it would appear in a web browser. The **imagesc()**command displays the image on scaled axes with the min value as black and the max value as white.

Using **imshow()**:



Using **imagesc()**:



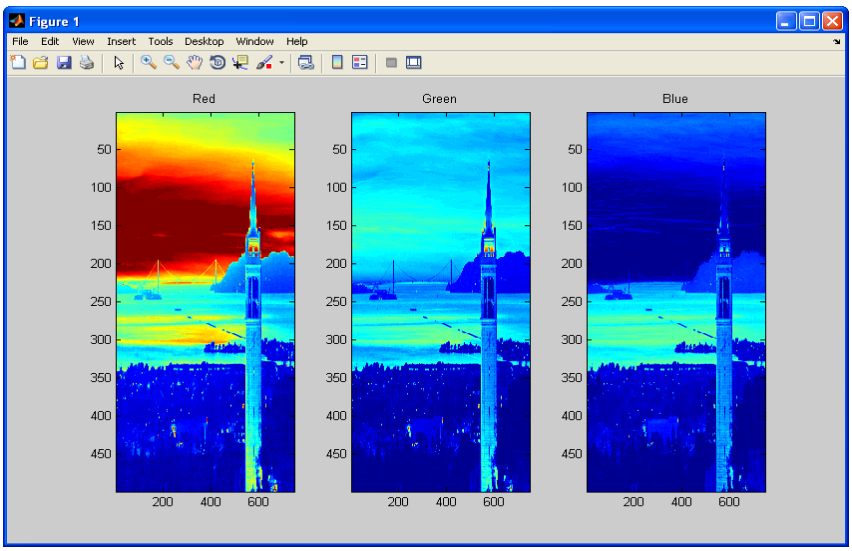
Notice each pixel is a 3-dimensional vector with values in the range [0,255]. The 3 numbers displayed is the amount of RGB.

Actually, a color image is a combined image of 3 grayscale images. So, we can display the individual RGB components of the image using the following script:

```
subplot(131);
imagesc(img(:,:,1));
title('Red');

subplot(132);
imagesc(img(:,:,2));
title('Green');

subplot(133);
imagesc(img(:,:,3));
title('Blue');
```



3. imwrite()

To save your new image, use the **imwrite()** command:

```
imwrite(blue_img, 'Blue4_BerkeleyTower.png', 'png');
```

4. rgb2gray()

To conver RGB image into grayscal this command is used.

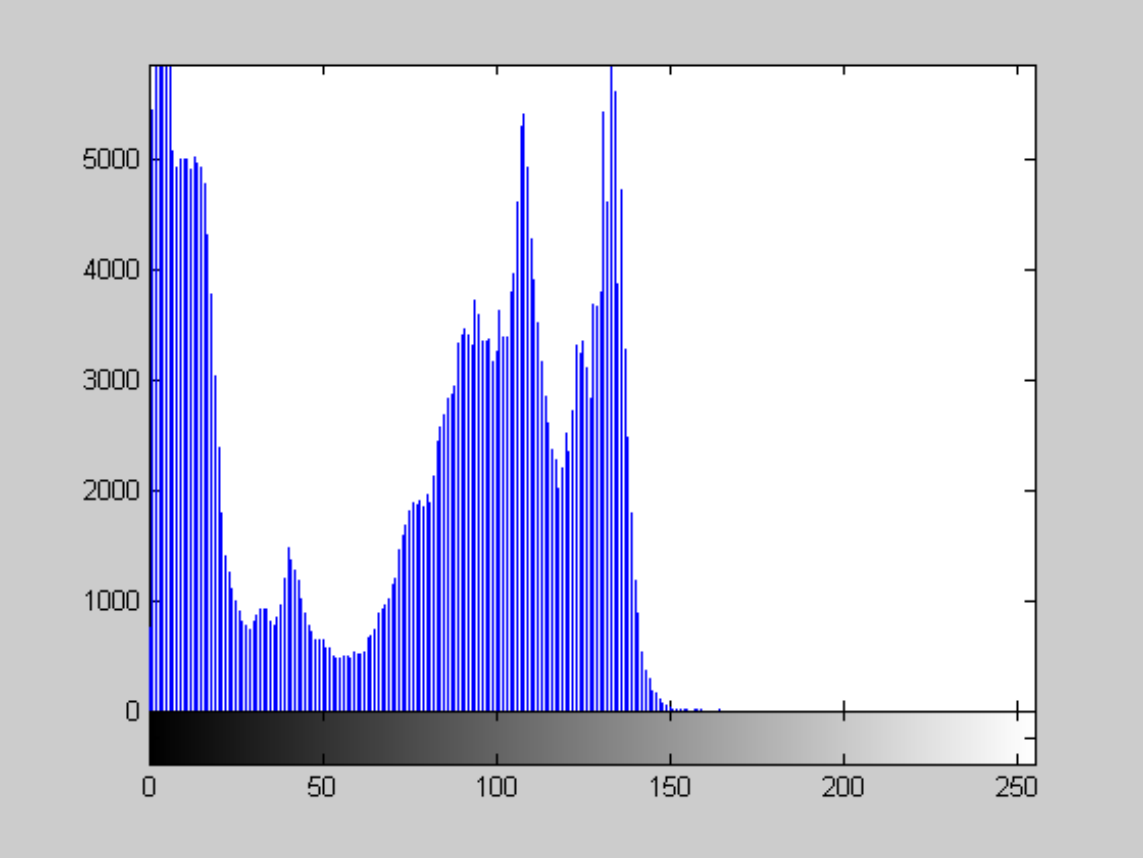
```
img = imread('ImageProcessing_1/BerkeleyTower.png');
gray = rgb2gray(img);
imshow(gray);
```

Now, we have only one value for each pixel not 3 values. In other words, the **rgb2gray()**converted RGB images to grayscale by eliminating the hue and saturation information while retaining the luminance.

5. imhist()

Display a histogram of image data. **imhist(img,n)** displays a histogram with n bins for the intensity image above a grayscale colorbar of length n. If we omit the argument, **imhist()** uses a default value of n = 256 if I is a grayscale image, or n = 2 if I is a binary image.

```
img = imread('ImageProcessing_1/BerkeleyTower.png');
gray = rgb2gray(img);
imhist(gray);
```



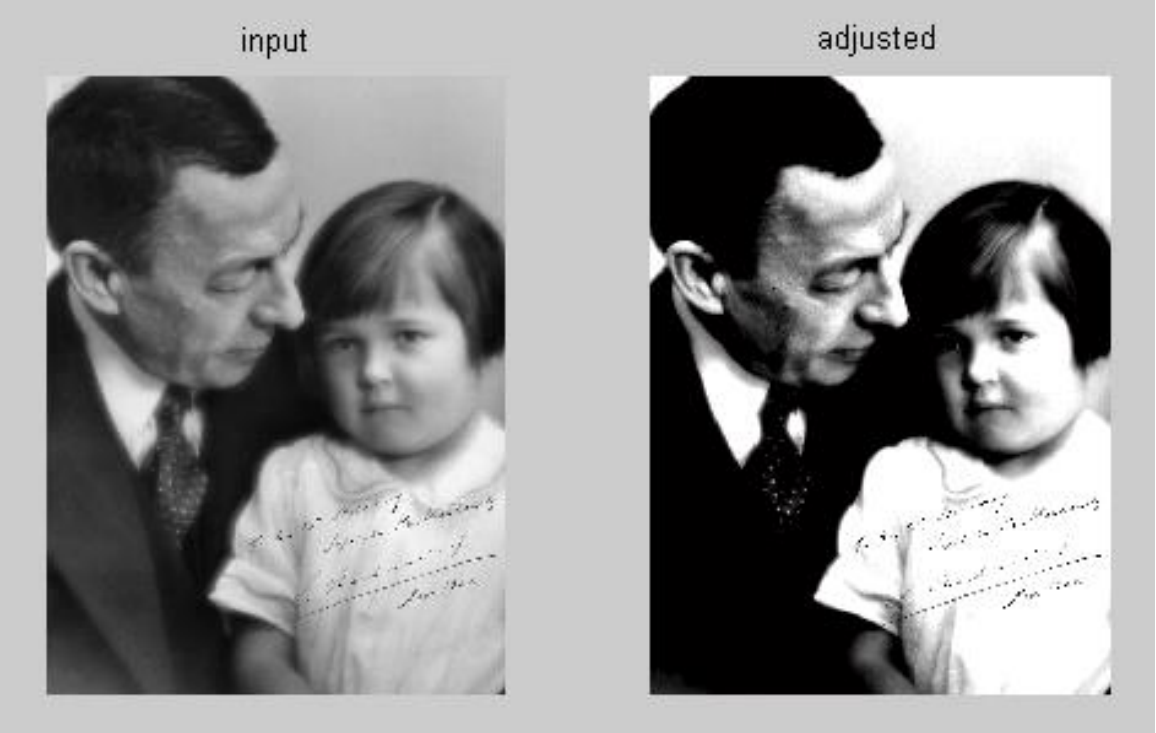
6. imadjust()

imadjust() adjust image intensity values. It maps the values in intensity image of an input to new values in output image. This increases the contrast of the output image.

```
img = imread('ImageProcessing_1/Rachmaninoff.jpg');
gray = rgb2gray(img);
adj_img = imadjust(gray, [0.3,0.7],[]);

subplot(121);
imshow(gray);
title('input');

subplot(122);
imshow(adj_img);
title('adjusted');
```



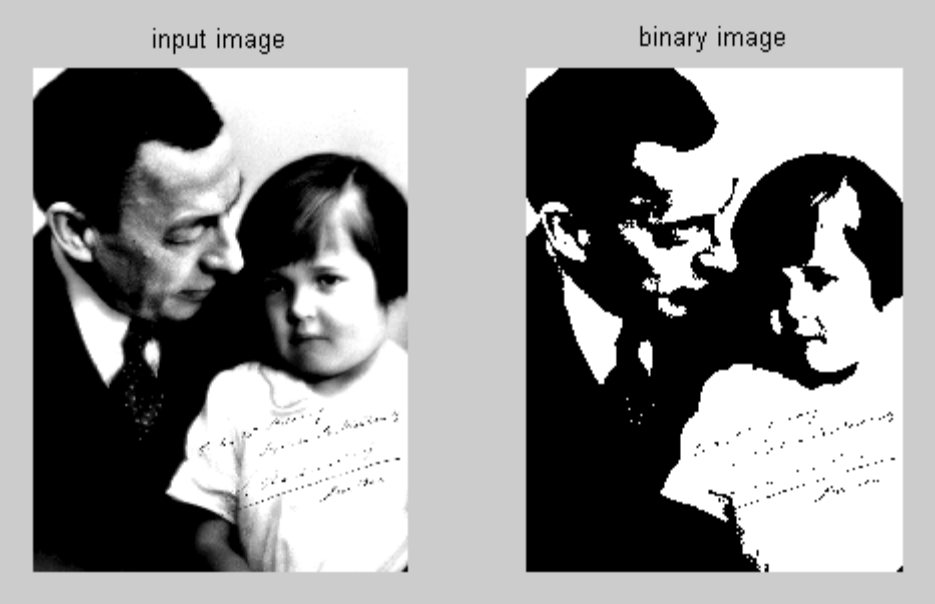
7. im2bw()

im2bw() converts the grayscale image to a binary image. We'll use the adjusted image.

```
img = imread('ImageProcessing_1/Rachmaninoff.jpg');
gray = rgb2gray(img);
adj_img = imadjust(gray, [0.3,0.7],[]);
bw_img = im2bw(adj_img);

subplot(121);
imshow(adj_img);
title('input image');

subplot(122);
imshow(bw_img);
title('binary image');
```



Local Features

Local features refer to a pattern or distinct structure found in an image, such as a point, edge, or small image patch. They are usually associated with an image patch that differs from its immediate surroundings by texture, color, or intensity. What the feature actually represents does not matter, just that it is distinct from its surroundings. Examples of local features are blobs, corners, and edge pixels.

8. Edge Detection

In an image, an edge is a curve that follows a path of rapid change in image intensity. Edges are often associated with the boundaries of objects in a scene. Edge detection is used to identify the edges in an image. Common edge detection algorithms include Sobel, Canny, Prewitt, Roberts, and fuzzy logic methods.

The most powerful edge-detection method that edge provides is the Canny method. The Canny method differs from the other edge-detection methods in that it uses two different thresholds (to detect strong and weak edges), and includes the weak edges in the output only if they are connected to strong edges. This method is therefore less likely than the others to be affected by noise, and more likely to detect true weak edges.

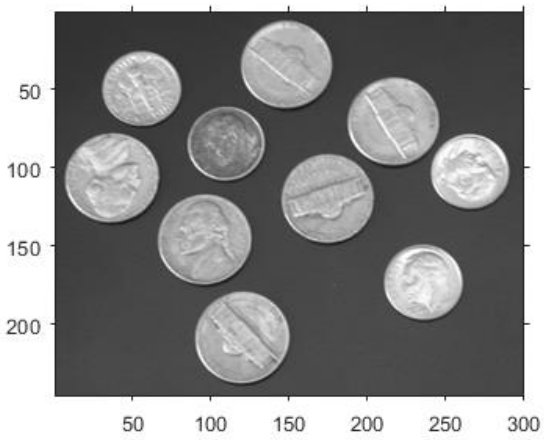
8.1. Detect Edges in Images

This example shows how to detect edges in an image using both the Canny edge detector and the Sobel edge detector.

Step#1

Read image and display it.

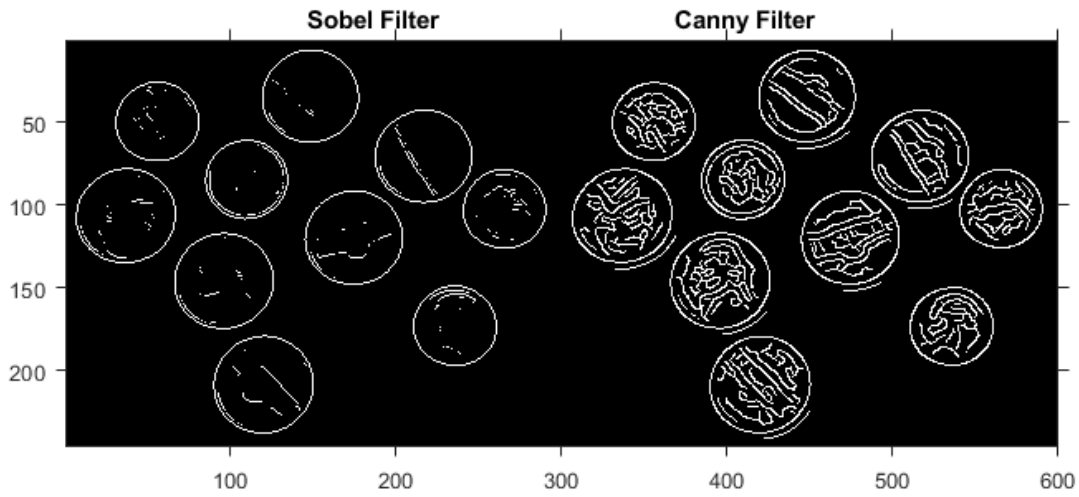
```
I = imread('coins.png');
imshow(I)
```



Step#2

Apply both the Sobel and Canny edge detectors to the image and display them for comparison.

```
BW1 = edge(I,'sobel');
BW2 = edge(I,'canny');
figure;
imshowpair(BW1,BW2,'montage')
title('Sobel Filter                                    Canny Filter');
```

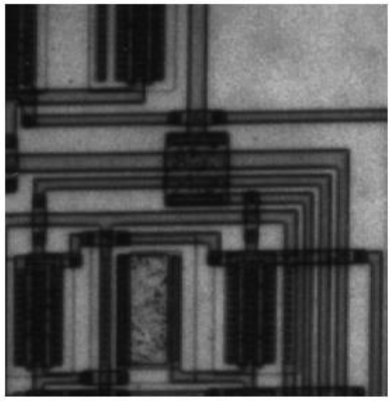


8.2. Compare Edge Detection Using Canny and Prewitt Methods:

Step#1

Read a grayscale image into the workspace and display it.

```
I = imread('circuit.tif');
imshow(I)
```



Step#2

Find edges using the Canny method.

```
BW1 = edge(I,'Canny');
```

Step#3

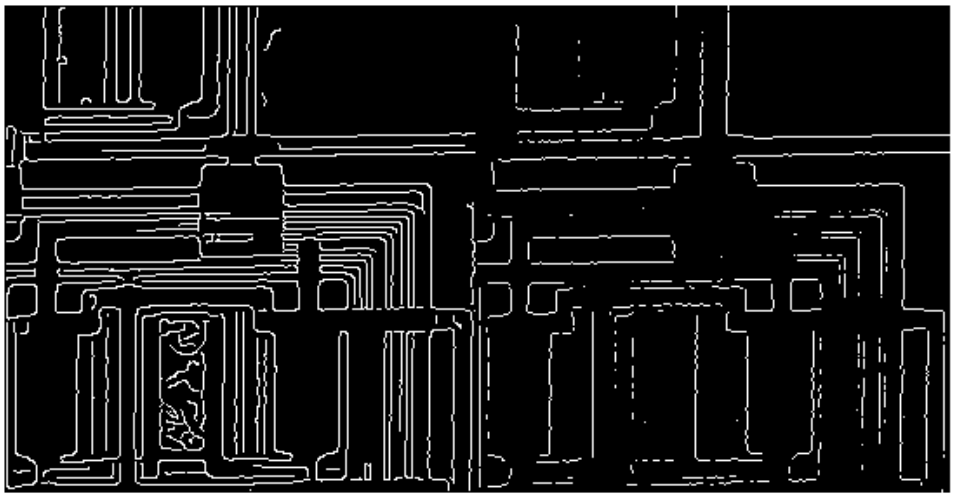
Find edges using the Prewitt method.

```
BW2 = edge(I,'Prewitt');
```

Step#4

Display both results side-by-side.

```
imshowpair(BW1,BW2,'montage')
```



9. Local Feature Extraction and detection

Computer Vision System Toolbox™ algorithms include the FAST, Harris, and Shi & Tomasi corner detectors, and the SURF, KAZE, and MSER blob detectors. The toolbox includes the SURF, FREAK, BRISK, LBP, and HOG descriptors. You can mix and match the detectors and the descriptors depending on the requirements of your application.

9.1. Feature Detection:

There are various various feature which we detect by using matlab some of the function for features detection are given below

Detect Features

detectBRISKFeatures	Detect BRISK features and return BRISKPoints object
detectFASTFeatures	Detect corners using FAST algorithm and return cornerPoints object
detectHarrisFeatures	Detect corners using Harris–Stephens algorithm and return cornerPoints object
detectMinEigenFeatures	Detect corners using minimum eigenvalue algorithm and return cornerPoints object
detectMSERFeatures	Detect MSER features and return MSERRegions object
detectSURFFeatures	Detect SURF features and return SURFPoints object
detectKAZEFeatures	Detect KAZE features

9.1.1.detectFASTfeatures Function:

Detect corners using FAST algorithm and return cornerPoints object.

Step#1

Read the image

```
I = imread('cameraman.tif');
```

Step#2

Find the corners.

```
corners = detectFASTFeatures(I);
```

Step#3

Display the results.

```
imshow(I); hold on;
plot(corners.selectStrongest(50));
```

9.2. Extract Features:

This function extracts interest point descriptors from images

Extract Corner Features from an image example:

Step#1

Read the image

```
I = imread('cameraman.tif');
```

Step#2

Find and extract corners features.

```
corners = detectHarrisFeatures(I);
[features, valid_corners] = extractFeatures(I, corners);
```

Step#3

Display the image.

```
figure; imshow(I); hold on
```

Step#4

Plot valid corners points

```
plot(valid_corners);
```

.

9.2.1.Extract SURF Features from an image:

Step#1

Read the image

```
I = imread('cameraman.tif');
```

Step#2

Find and extract features

```
points = detectSURFFeatures(I);
[features, valid_points] = extractFeatures(I, points);
```

Step#3

Display and plot ten strongest SURF Features

```
figure; imshow(I); hold on;
plot(valid_points.selectStrongest(10),'showOrientation',true);
```

9.2.2.Match Features:

Display corresponding feature points between two images

Step#1

Read the images

```
I1 = rgb2gray(imread('parkinglot_left.png'));
I2 = rgb2gray(imread('parkinglot_right.png'));
```

Step#2

Detect SURF features

```
points1 = detectHarrisFeatures(I1);
points2 = detectHarrisFeatures(I2);
```

Step#3

Extract SURF features

```
[f1, vpts1] = extractFeatures(I1, points1);
[f2, vpts2] = extractFeatures(I2, points2);
```

Step#4

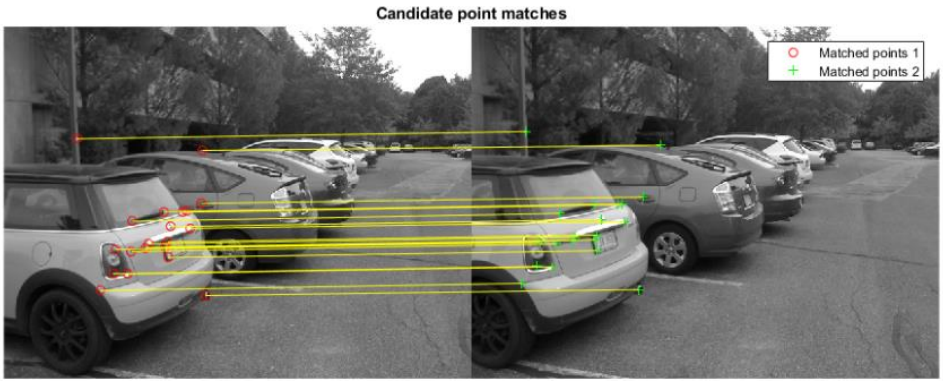
Match features

```
indexPairs = matchFeatures(f1, f2) ;
matchedPoints1 = vpts1(indexPairs(1:20, 1));
matchedPoints2 = vpts2(indexPairs(1:20, 2));
```

Step#4

Validate candidate Matches

```
figure; ax = axes;
showMatchedFeatures(I1,I2,matchedPoints1,matchedPoints2,'montage','Parent',ax);
title(ax, 'Candidate point matches');
legend(ax, 'Matched points 1','Matched points 2');
```



10. Find Feature Matching using RANSAC

This example shows how to automatically align two images that differ by a rotation and a scale change. It closely parallels another example titled Find Image Rotation and Scale. Instead of using a manual approach to register the two images, it utilizes feature-based techniques found in the Computer Vision System Toolbox™ to automate the registration process.In this example, you will use detectSURFFeatures and vision.GeometricTransformEstimator System object to recover rotation angle and scale factor of a distorted image. You will then transform the distorted image to recover the original image

Step#1: Read the image

```
original = imread('cameraman.tif');
imshow(original);
text(size(original,2),size(original,1)+15, ...
    'Image courtesy of Massachusetts Institute of Technology', ...
    'FontSize',7,'HorizontalAlignment','right');
```

Image courtesy of Massachusetts Institute of Technology

Step#2: Resize and rotate the image.

```
scale = 0.7;
J = imresize(original, scale); % Try varying the scale factor.

theta = 30;
distorted = imrotate(J,theta); % Try varying the angle, theta.
figure, imshow(distorted)
```

Step#3: Find Matching features between images.
Detect feature in both images.

```
ptsOriginal  = detectSURFFeatures(original);
ptsDistorted = detectSURFFeatures(distorted);
```

Extract features descriptors.

```
[featuresOriginal,  validPtsOriginal]  = extractFeatures(original,  ptsOriginal);
[featuresDistorted, validPtsDistorted] = extractFeatures(distorted, ptsDistorted);
```

Match feature by using their descriptors.

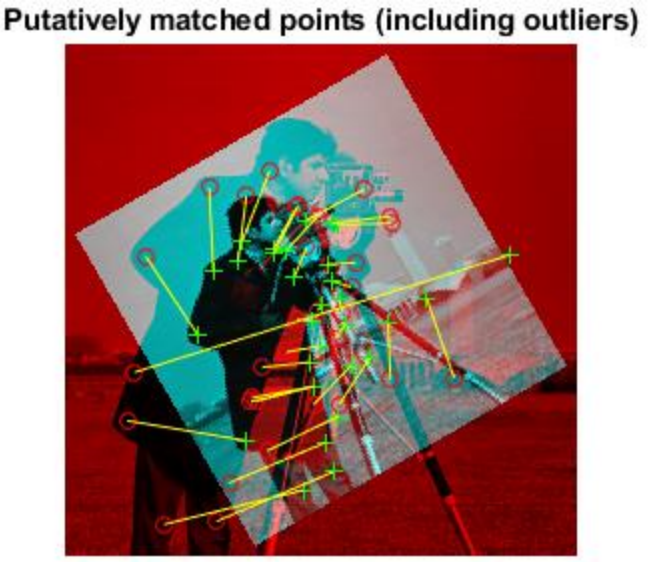
```
indexPairs = matchFeatures(featuresOriginal, featuresDistorted);
```

Retrieve location of corresponding point for each image.

```
matchedOriginal  = validPtsOriginal(indexPairs(:,1));
matchedDistorted = validPtsDistorted(indexPairs(:,2));
```

Show putative point matches

```
figure;
showMatchedFeatures(original,distorted,matchedOriginal,matchedDistorted);
title('Putatively matched points (including outliers)');
```



Step#4: Estimate Transformations.

```
[tform, inlierDistorted, inlierOriginal] = estimateGeometricTransform(...
    matchedDistorted, matchedOriginal, 'similarity');
```

Display matching point pairs used in the computation of transformation.

```
figure;
showMatchedFeatures(original,distorted,inlierOriginal,inlierDistorted);
title('Matching points (inliers only)');
legend('ptsOriginal','ptsDistorted');
```

Exercise#1

1. Read an image of your own choice?
2. Convert it into grayscale?
3. Adjust intensity values?
4. Convert to binary image?
5. Compute histogram?
6. Write all the results to a specific folder?

Note: Display all the result in subplots

Exercise#2

Find the corresponding point between any two images by using SURF feature extractor. Read First image, Rotate the first image by 20 degree and consider it as a second image. Help: u can use imrotate (image, -20) for rotation of first image.