BTS | Barcelona Technology School

Advanced Data Analysis

Master in Big Data Solutions 2020-2021

Ankit Tewari
Course Instructor, Real Time Data Analysis (2020-21)
ankit.tewari@bts.tech

Contents

Today's topics at a glance

1. Introduction to Image Processing
2. Image Histograms
3. Histogram Equalizations
4. Image Smoothening
5. Template Matching
6. Image Filtering
7. Morphological Analysis
8. Feature Detection: Corner Detection
9. Feature Description: Feature Matching

Introduction to Image Processing

What is Image Processing?

According to the Oxford's dictionary, image processing is defined as "the analysis and manipulation of a digitized image, especially in order to improve its quality".

Image processing basically includes the following three steps:

1. Importing the image via image acquisition tools;
2. Analysing and manipulating the image;
3. Output in which result can be altered image or report that is based on image analysis.

When we use the term "image manipulation", we may be referring to one (or more) of these-

1. Image resizing
2. Image enhancement
3. Image Filtering
4. Image smoothing .....

Introduction to Image Processing

Example of Image Processing

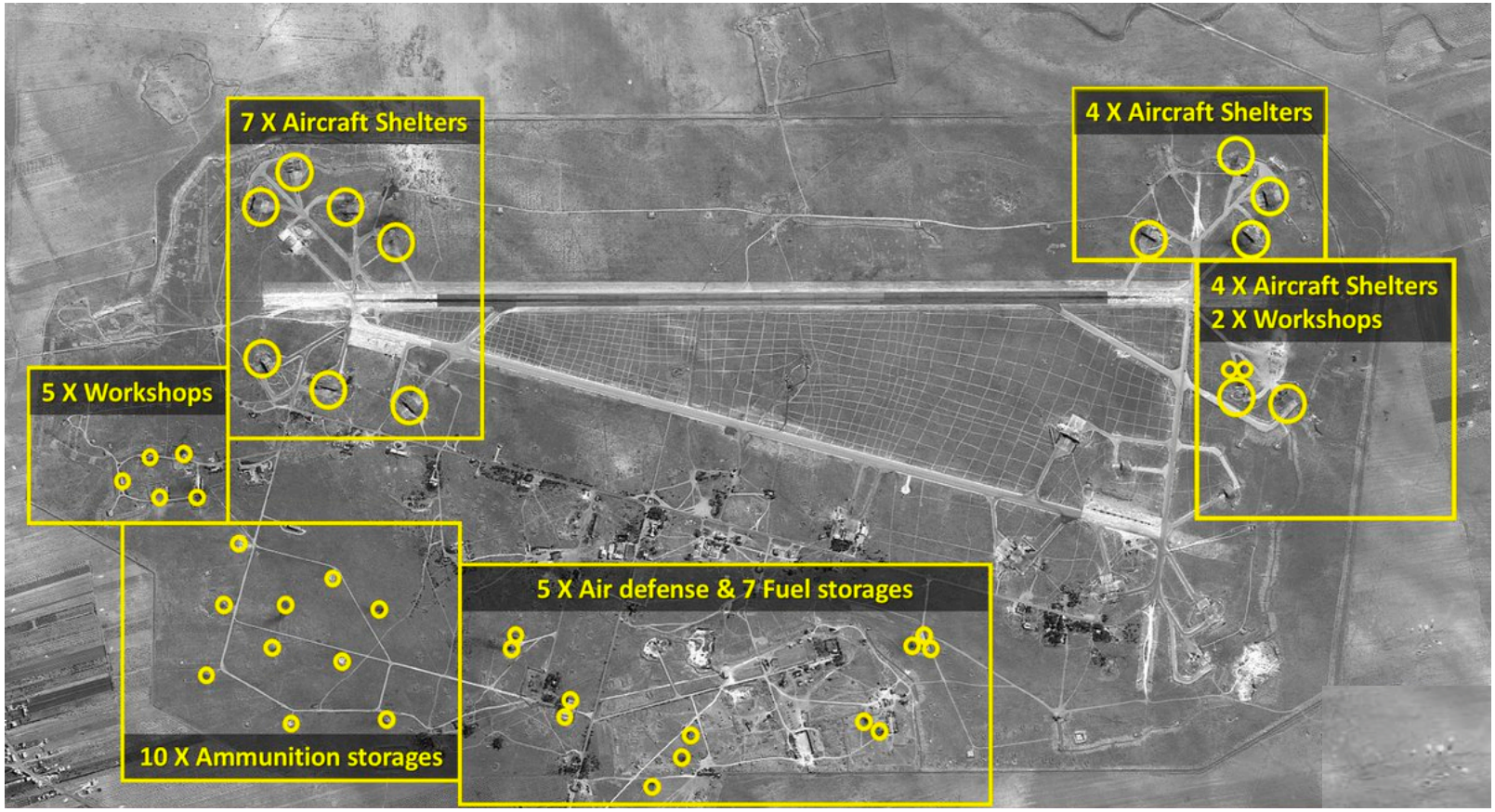


Image Processing

Image Histograms

Consider histogram as a graph or plot, which gives you an overall idea about the intensity distribution of an image. It is a plot with pixel values (ranging from 0 to 255, not always) in X-axis and corresponding number of pixels in the image on Y-axis.

It is just another way of understanding the image.

By looking at the histogram of an image, one may get intuition about contrast, brightness, intensity distribution etc of that image.

Almost all image processing tools today, provides features on histogram.

```
img = cv.imread('../Images/british_passport.png',0)
plt.imshow(img,cmap = 'gray'), plt.title('Original')
```

```
(<matplotlib.image.AxesImage at 0x145a4e610>, Text(0.5, 1.0, 'Original'))
```

Image Processing

Image Histograms

Consider histogram as a graph or plot, which gives you an overall idea about the intensity distribution of an image. It is a plot with pixel values (ranging from 0 to 255, not always) in X-axis and corresponding number of pixels in the image on Y-axis.

It is just another way of understanding the image.

By looking at the histogram of an image, one may get intuition about contrast, brightness, intensity distribution etc of that image.

Almost all image processing tools today, provides features on histogram.

```
plt.hist(img.ravel(),256,[0,256]); plt.show()
```

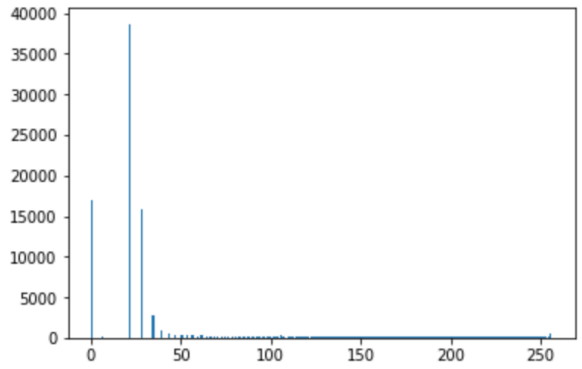


Image Processing

Image Histograms

Consider histogram as a graph or plot, which gives you an overall idea about the intensity distribution of an image. It is a plot with pixel values (ranging from 0 to 255, not always) in X-axis and corresponding number of pixels in the image on Y-axis.

It is just another way of understanding the image.

By looking at the histogram of an image, one may get intuition about contrast, brightness, intensity distribution etc of that image.

Almost all image processing tools today, provides features on histogram.

```
img = cv.imread('../Images/university_barcelona.jpeg',0)
plt.imshow(img,cmap = 'gray'), plt.title('Original')
```

```
(<matplotlib.image.AxesImage at 0x1465d2c50>, Text(0.5, 1.0, 'Original'))
```

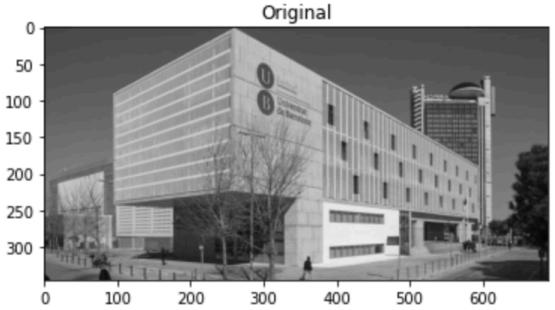
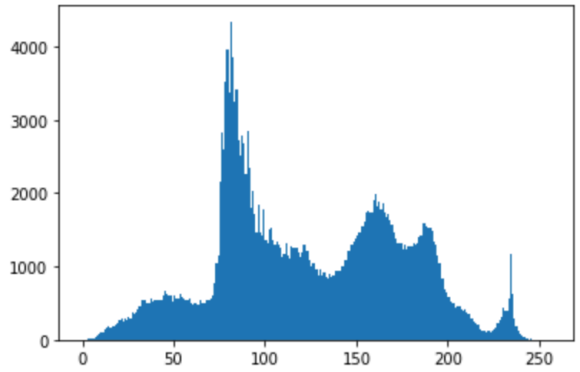


Image Processing

Image Histograms

```
plt.hist(img.ravel(),256,[0,256]); plt.show()
```




```
img = cv.imread('../Images/university_barcelona.jpeg')
color = ('b','g','r')
for i,col in enumerate(color):
    histr = cv.calcHist([img],[i],None,[256],[0,256])
    plt.plot(histr,color = col)
    plt.xlim([0,256])
plt.show()
```

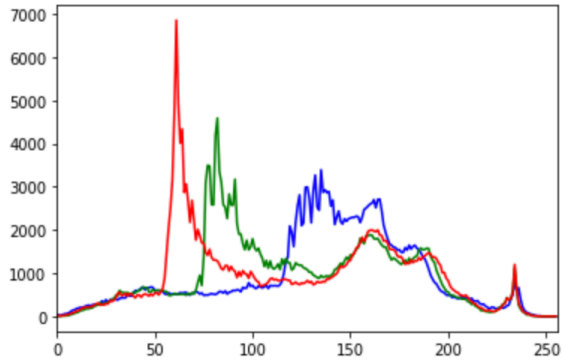


Image Processing

Histograms Equalization

Consider histogram as a graph or plot, which gives you an overall idea about the intensity distribution of an image. It is a plot with pixel values (ranging from 0 to 255, not always) in X-axis and corresponding number of pixels in the image on Y-axis.

Now, Consider an image whose pixel values are confined to some specific range of values only. For example, brighter image will have all pixels confined to high values. But a good image will have pixels from all regions of the image. So we need to stretch this histogram to either ends and that is what Histogram Equalization does (in simple words). This normally improves the contrast of the image.

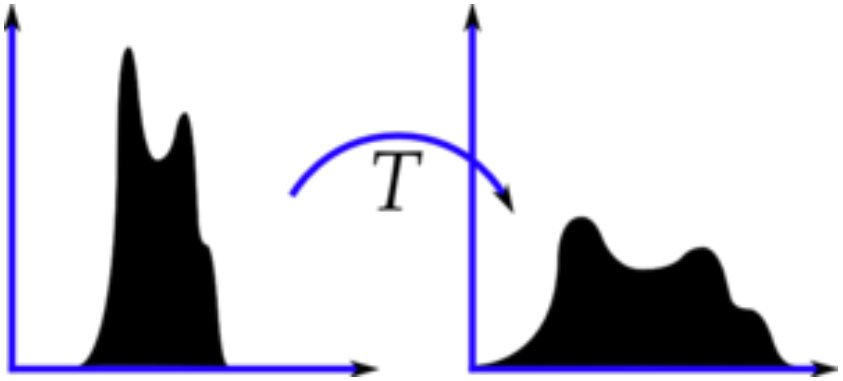


Image Processing

Histograms Equalization

Consider histogram as a graph or plot, which gives you an overall idea about the intensity distribution of an image. It is a plot with pixel values (ranging from 0 to 255, not always) in X-axis and corresponding number of pixels in the image on Y-axis.

```
img = cv.imread('../Images/hills_greyscale.jpg',0)
equ = cv.equalizeHist(img)
res = np.hstack((img,equ))

fig = plt.figure(figsize=(15, 15))
plt.imshow(res,cmap = 'gray'), plt.title('Original')
```

```
(<matplotlib.image.AxesImage at 0x145ae61d0>, Text(0.5, 1.0, 'Original'))
```

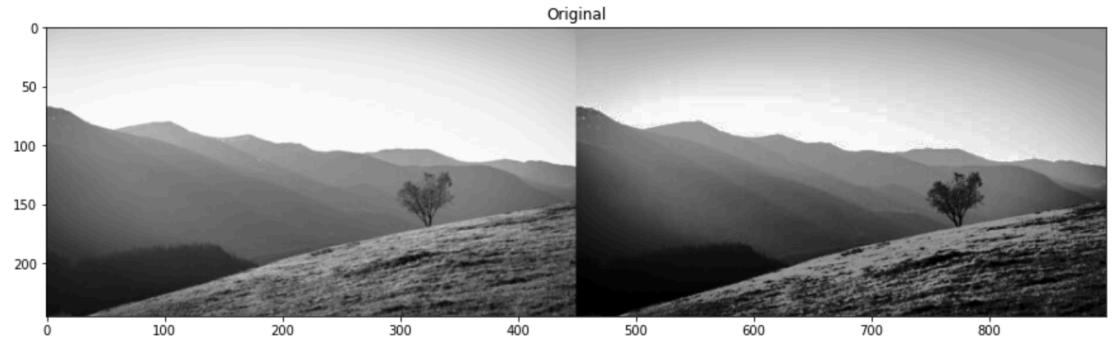


Image Processing

Histograms Equalization

Consider histogram as a graph or plot, which gives you an overall idea about the intensity distribution of an image. It is a plot with pixel values (ranging from 0 to 255, not always) in X-axis and corresponding number of pixels in the image on Y-axis.

```
img = cv.imread('../Images/hills_greyscale.jpg',0)
equ = cv.equalizeHist(img)
res = np.hstack((img,equ))

fig = plt.figure(figsize=(15, 15))
plt.imshow(res,cmap = 'gray'), plt.title('Original')
```

```
(<matplotlib.image.AxesImage at 0x145ae61d0>, Text(0.5, 1.0, 'Original'))
```

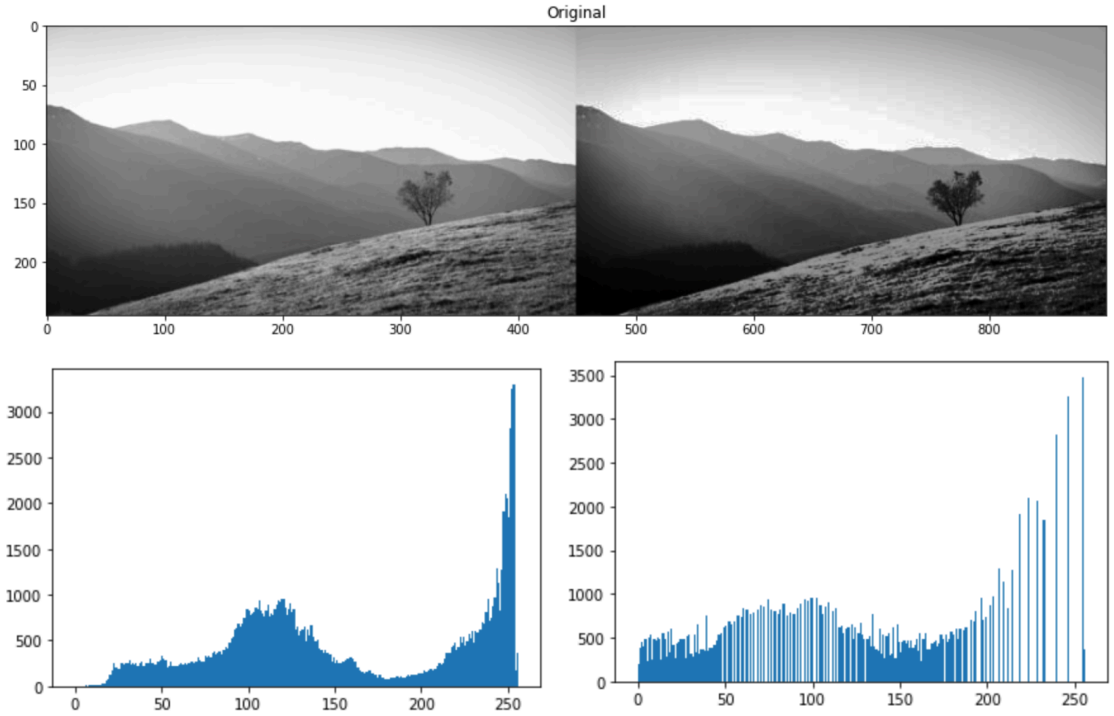


Image Smoothing

How to obtain smoothen images?

The objective of image smoothing is to exploit the low pass filters to perform-

- Image blurring: Image blurring is achieved by convolving the image with a low-pass filter kernel. It is useful for removing noise. It actually removes high frequency content (for example, noise, edges etc) from the image. So edges are blurred a little bit in this operation (there are also blurring techniques which don't blur the edges). OpenCV provides four main types of blurring techniques.
- Feature extraction: Apply custom-made filters to images (2D convolution)

But before looking at those techniques, let's look how does a typical blurring looks like.

```
img = cv.imread('../Images/2021.jpg')
kernel = np.ones((5,5),np.float32)/25
dst = cv.filter2D(img,-1,kernel)

fig = plt.figure(figsize=(15, 15))
plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(dst),plt.title('Smoothen')
plt.xticks([]), plt.yticks([])
plt.show()
```

Image Smoothing

How to obtain smoothen images?

The objective of image smoothing is to exploit the low pass filters to perform-

- Image blurring: Image blurring is achieved by convolving the image with a low-pass filter kernel. It is useful for removing noise. It actually removes high frequency content (for example, noise, edges etc) from the image. So edges are blurred a little bit in this operation (there are also blurring techniques which don't blur the edges). OpenCV provides four main types of blurring techniques.
- Feature extraction: Apply custom-made filters to images (2D convolution)

But before looking at those techniques, let's look how does a typical blurring looks like.

```
img = cv.imread('../Images/2021.jpg')
kernel = np.ones((5,5),np.float32)/25
dst = cv.filter2D(img,-1,kernel)

fig = plt.figure(figsize=(15, 15))
plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(dst),plt.title('Smoothen')
plt.xticks([]), plt.yticks([])
plt.show()
```

Image Smoothing

Smoothening by Averaging

It is done by convolving an image with a normalized box filter. It simply takes the average of all the pixels under the kernel area and replaces the central element.

We need to specify the width and height of the kernel. For example, for a 3x3 normalized box filter would look like the below:

$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Image Smoothing

Smoothening by Averaging

It is done by convolving an image with a normalized box filter. It simply takes the average of all the pixels under the kernel area and replaces the central element.

We need to specify the width and height of the kernel. For example, for a 3x3 normalized box filter would look like the below:

$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

```
img = cv.imread('../Images/tensorflow_image.png', 0)
kernel = np.ones((5,5),np.float32)/25
dst = cv.filter2D(img,-1,kernel)

fig = plt.figure(figsize=(15, 15))
plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(dst),plt.title('Smoothen')
plt.xticks([]), plt.yticks([])
plt.show()
```

Image Smoothing

Smoothening by Gaussian Blurring

In this method, instead of a box filter, a Gaussian kernel is used.

It is performed using the function provided for this purpose and we need to specify the width and height of the kernel which should be positive and odd.

Apart from that, we also need to specify the standard deviation in the X and Y directions, denoted by sigmaX and sigmaY respectively. If only sigmaX is specified, sigmaY is taken as the same as sigmaX.

If both are given as zeros, they are calculated from the kernel size. Gaussian blurring is highly effective in removing Gaussian noise from an image.

Image Smoothing

Smoothening by Gaussian Blurring

In this method, instead of a box filter, a Gaussian kernel is used.

It is performed using the function provided for this purpose and we need to specify the width and height of the kernel which should be positive and odd.

Apart from that, we also need to specify the standard deviation in the X and Y directions, denoted by sigmaX and sigmaY respectively. If only sigmaX is specified, sigmaY is taken as the same as sigmaX.

If both are given as zeros, they are calculated from the kernel size. Gaussian blurring is highly effective in removing Gaussian noise from an image.

```
blur = cv.GaussianBlur(img,(5,5),0)

fig = plt.figure(figsize=(15, 15))
plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(blur),plt.title('Smoothen')
plt.xticks([]), plt.yticks([])
plt.show()
```

Image Smoothing

Smoothening by Median Blurring

In this method, the function takes the median of all the pixels under the kernel area and the central element is replaced with this median value. This is highly effective against salt-and-pepper noise in an image.

Interestingly, in the previous filters, the central element is a newly calculated value which may be a pixel value in the image or a new value. But in median blurring, the central element is always replaced by some pixel value in the image. It reduces the noise effectively. Its kernel size should be a positive odd integer.

Image Smoothing

Smoothening by Median Blurring

In this method, the function takes the median of all the pixels under the kernel area and the central element is replaced with this median value. This is highly effective against salt-and-pepper noise in an image.

Interestingly, in the previous filters, the central element is a newly calculated value which may be a pixel value in the image or a new value. But in median blurring, the central element is always replaced by some pixel value in the image. It reduces the noise effectively. Its kernel size should be a positive odd integer.

```
median = cv.medianBlur(img,5)

fig = plt.figure(figsize=(15, 15))
plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(median),plt.title('Smoothen')
plt.xticks([]), plt.yticks([])
plt.show()
```

Image Smoothing

Smoothening by Bilateral Filtering

It is highly effective in noise removal while keeping edges sharp. However, the operation is slower compared to other filters.

The manner in which Gaussian filter takes the neighbourhood around the pixel and finds its Gaussian weighted average, we conclude that Gaussian filter is a function of space alone, that is, nearby pixels are considered while filtering. It doesn't consider whether pixels have almost the same intensity. It doesn't consider whether a pixel is an edge pixel or not. So it blurs the edges also, which we don't want to do.

Bilateral filtering considers two Gaussian filters- one that performs filtering in space, and another filter which is a function of pixel difference. The Gaussian function of space makes sure that only nearby pixels are considered for blurring, while the Gaussian function of intensity difference makes sure that only those pixels with similar intensities to the central pixel are considered for blurring. Thus, Bilateral filtering preserves the edges since pixels at edges will have large intensity variation.

Image Smoothing

Smoothening by Bilateral Filtering

It is highly effective in noise removal while keeping edges sharp. However, the operation is slower compared to other filters.

Bilateral filtering considers two Gaussian filters- one that performs filtering in space, and another filter which is a function of pixel difference. The Gaussian function of space makes sure that only nearby pixels are considered for blurring, while the Gaussian function of intensity difference makes sure that only those pixels with similar intensities to the central pixel are considered for blurring. Thus, Bilateral filtering preserves the edges since pixels at edges will have large intensity variation.

```
blur = cv.bilateralFilter(img,9,75,75)

fig = plt.figure(figsize=(15, 15))
plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(blur),plt.title('Smoothen')
plt.xticks([]), plt.yticks([])
plt.show()
```

Image Smoothing

Motion Blurring

```
img = cv.imread('../Images/motion_train.jpg')
size = 11

# generating the kernel
kernel_motion_blur = np.zeros((size, size))
kernel_motion_blur[int((size-1)/2), :] = np.ones(size)
kernel_motion_blur = kernel_motion_blur / size

# applying the kernel to the input image
output = cv2.filter2D(img, -1, kernel_motion_blur)

plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(output),plt.title('Motion Blurred')
plt.xticks([]), plt.yticks([])
plt.show()
```

Image Gradients

What are Image Gradients?

Image gradients are the operators that allow examination of an image in terms of its edges by means of high pass filters. There are three common types of gradient filters or High-pass filters, known as, Sobel, Scharr and Laplacian. We will see each one of them.

1. **Sobel and Scharr Derivatives:** Sobel operators is a joint Gaussian smoothing plus differentiation operation, so it is more resistant to noise.

   We may specify the direction of derivatives to be taken, vertical or horizontal (by the arguments, yorder and xorder respectively) apart from specifying the size of kernel by the argument. If kernel size = -1, a 3x3 Scharr filter is used which gives better results than 3x3 Sobel filter.

2. **Laplacian Derivatives:**
   It calculates the Laplacian of the image given by the relation:

$$\Delta src = \frac{\partial^2 src}{\partial x^2} + \frac{\partial^2 src}{\partial y^2}$$

   where each derivative is found using Sobel derivatives. If kernel size = 1, then following kernel is used for filtering:

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

Image Gradients

High Pass Filters

Let us understand the concept we just discussed by means of a code-

```
# kernel size must be odd numbers
img = cv.imread('../Images/newspaper_sudoku.jpeg',0)
laplacian = cv.Laplacian(img,cv.CV_64F)
sobelx = cv.Sobel(img,cv.CV_64F,1,0,ksize=1)
sobely = cv.Sobel(img,cv.CV_64F,0,1,ksize=5)
plt.rcParams['figure.figsize'] = (18.0, 8.0)
plt.subplot(2,2,1),plt.imshow(img,cmap = 'gray')
plt.title('Original'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,2),plt.imshow(laplacian,cmap = 'gray')
plt.title('Laplacian'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,3),plt.imshow(sobelx,cmap = 'gray')
plt.title('Sobel X'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,4),plt.imshow(sobely,cmap = 'gray')
plt.title('Sobel Y'), plt.xticks([]), plt.yticks([])
plt.show()
```

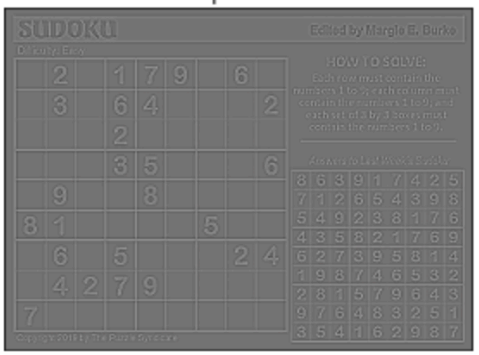
Image Gradients

High Pass Filters

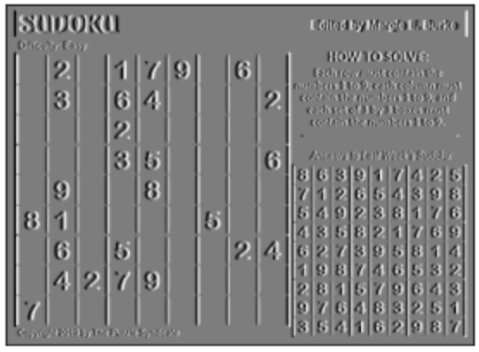
Let us understand the concept we just discussed by means of a code-

Original

Laplacian



Sobel X



Sobel Y

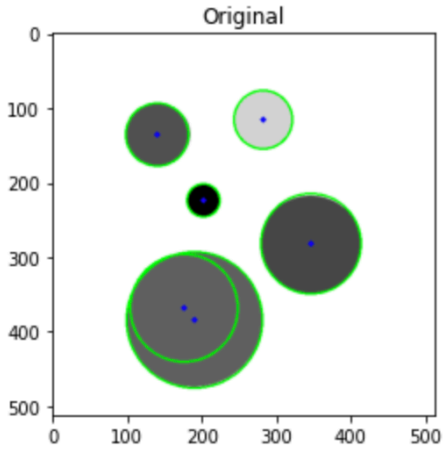
Geometry Detection

Detecting Circles

The Hough Transform is a popular technique to detect any shape, if we can represent that shape in a mathematical form. It can detect the shape even if it is broken or distorted a little bit.

```
img = cv.imread('../Images/circles.jpeg',0)
img = cv.medianBlur(img,5)
cimg = cv.cvtColor(img,cv.COLOR_GRAY2BGR)
circles = cv.HoughCircles(img,cv.HOUGH_GRADIENT,1,20,
                          param1=50,param2=30,minRadius=0,maxRadius=0)
circles = np.uint16(np.around(circles))
for i in circles[0,:]:
    # draw the outer circle
    cv.circle(cimg,(i[0],i[1]),i[2],(0,255,0),2)
    # draw the center of the circle
    cv.circle(cimg,(i[0],i[1]),2,(0,0,255),3)
plt.plot(), plt.imshow(cimg),plt.title('Original')
```

```
([], <matplotlib.image.AxesImage at 0x1479f55d0>, Text(0.5, 1.0, 'Original'))
```



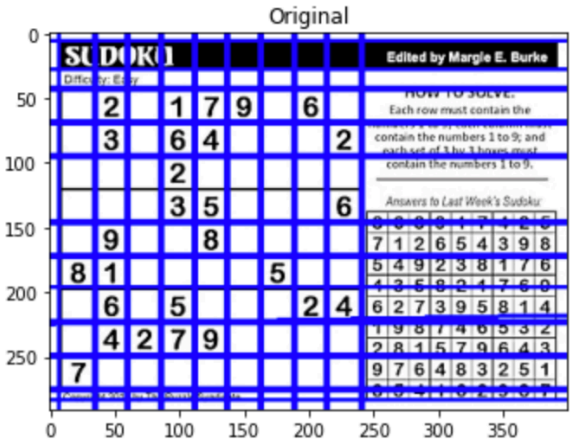
Geometry Detection

Detecting Lines

The Hough Transform is a popular technique to detect any shape, if we can represent that shape in a mathematical form. It can detect the shape even if it is broken or distorted a little bit.

```
img = cv.imread(cv.samples.findFile('../Images/newspaper_sudoku.jpeg'))
gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)
edges = cv.Canny(gray,50,150,apertureSize = 3)
lines = cv.HoughLines(edges,1,np.pi/180,200)
for line in lines:
    rho,theta = line[0]
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a*rho
    y0 = b*rho
    x1 = int(x0 + 1000*(-b))
    y1 = int(y0 + 1000*(a))
    x2 = int(x0 - 1000*(-b))
    y2 = int(y0 - 1000*(a))
    cv.line(img,(x1,y1),(x2,y2),(0,0,255),2)
plt.plot(), plt.imshow(img),plt.title('Original')
```

```
([], <matplotlib.image.AxesImage at 0x147f39f50>, Text(0.5, 1.0, 'Original'))
```



Template Matching

Detecting Mario Coins

Template Matching is a method for searching and finding the location of a template image in a larger image. It simply slides the template image over the input image (as in 2D convolution) and compares the template and patch of input image under the template image.

It returns a grayscale image, where each pixel denotes how much does the neighbourhood of that pixel match with template.

If input image is of size (WxH) and template image is of size (wxh), output image will have a size of (W-w+1, H-h+1). Once we get the result, we can proceed to find where is the maximum/minimum value. Take it as the top-left corner of rectangle and take (w,h) as width and height of the rectangle. That rectangle is then our region of template.

```
img_rgb = cv.imread('../Images/mario_snapshot.jpeg')
img_gray = cv.cvtColor(img_rgb, cv.COLOR_BGR2GRAY)
template = cv.imread('../Images/mario_coin.jpeg',0)
w, h = template.shape[::-1]
res = cv.matchTemplate(img_gray,template,cv.TM_CCOEFF_NORMED)
threshold = 0.8
loc = np.where( res >= threshold)
for pt in zip(*loc[::-1]):
    cv.rectangle(img_rgb, pt, (pt[0] + w, pt[1] + h), (0,0,255), 2)

fig = plt.figure(figsize=(15, 15))
plt.plot(),plt.imshow(img_rgb),plt.title('Original')
```

Template Matching

Detecting Mario Coins

Morphological Analysis

Image Dilation vs Image Erosion

Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images.
It needs two inputs, one is our original image, second one is called structuring element or kernel which decides the nature of operation. Two basic morphological operators are Erosion and Dilation. Then its variant forms like Opening, Closing, Gradient etc also comes into play.

Let us first understand what the concepts of Erosion and Dilation mean by means of an example and then we proceed to understand them deeper.

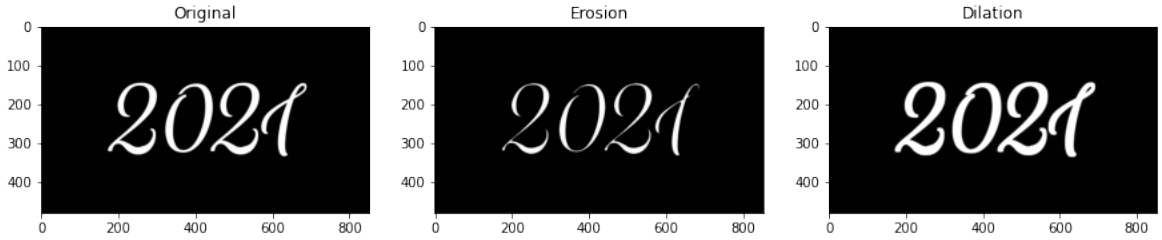
```
img = cv.imread('../Images/2021.jpg')
kernel = np.ones((5,5), np.uint8)
img_erosion = cv.erode(img, kernel, iterations=1)
img_dilation = cv.dilate(img, kernel, iterations=1)

fig = plt.figure(figsize=(10, 12))

plt.subplot(3, 1, 1), plt.imshow(img),plt.title('Original')
plt.subplot(3, 1, 2),plt.imshow(img_erosion),plt.title('Erosion')
plt.subplot(3, 1, 3),plt.imshow(img_dilation),plt.title('Dilation')
plt.show()
```

Morphological Analysis

Image Dilation vs Image Erosion

Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images.
It needs two inputs, one is our original image, second one is called structuring element or kernel which decides the nature of operation. Two basic morphological operators are Erosion and Dilation. Then its variant forms like Opening, Closing, Gradient etc also comes into play.

```
img = cv.imread('../Images/2021.jpg')
kernel = np.ones((5,5), np.uint8)
img_erosion = cv.erode(img, kernel, iterations=1)
img_dilation = cv.dilate(img, kernel, iterations=1)

fig = plt.figure(figsize=(10, 12))

plt.subplot(3, 1, 1), plt.imshow(img),plt.title('Original')
plt.subplot(3, 1, 2),plt.imshow(img_erosion),plt.title('Erosion')
plt.subplot(3, 1, 3),plt.imshow(img_dilation),plt.title('Dilation')
plt.show()
```



Morphological Analysis

Image Dilation vs Image Erosion

Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images.
It needs two inputs, one is our original image, second one is called structuring element or kernel which decides the nature of operation. Two basic morphological operators are Erosion and Dilation. Then its variant forms like Opening, Closing, Gradient etc also comes into play.

Erosion: The basic idea of erosion is just like soil erosion only, it erodes away the boundaries of foreground object (Always try to keep foreground in white). So what it does?

The kernel slides through the image (as in 2D convolution). A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero).

So what happens is that, all the pixels near boundary will be discarded depending upon the size of kernel. So the thickness or size of the foreground object decreases or simply white region decreases in the image. It is useful for removing small white noises (as we have seen in colorspace chapter), detach two connected objects etc.

Dilation: It is just opposite of erosion. Here, a pixel element is '1' if at least one pixel under the kernel is '1'. So it increases the white region in the image or size of foreground object increases. Normally, in cases like noise removal, erosion is followed by dilation. Because, erosion removes white noises, but it also shrinks our object. So we dilate it. Since noise is gone, they won't come back, but our object area increases. It is also useful in joining broken parts of an object.

Image Segmentation

The Watershed Algorithm

Any grayscale image can be viewed as a topographic surface where high intensity denotes peaks and hills while low intensity denotes valleys. You start filling every isolated valleys (local minima) with different coloured water (labels). As the water rises, depending on the peaks (gradients) nearby, water from different valleys, obviously with different colours will start to merge. To avoid that, you build barriers in the locations where water merges. We continue the work of filling water and building barriers until all the peaks are under water. Then the barriers we have created gives us the segmentation result. This is the "philosophy" behind the watershed.

Image Segmentation

The Watershed Algorithm

```
img = cv.imread('../Images/coins_segmentation.jpg')
gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)
ret, thresh = cv.threshold(gray,0,255,cv.THRESH_BINARY_INV+cv.THRESH_OTSU)

fig = plt.figure(figsize=(15, 15))
plt.subplot(1,2,1),plt.imshow(img,cmap = 'gray'), plt.title('Original')
plt.subplot(1,2,2),plt.imshow(gray,cmap = 'gray'), plt.title('Gray')
plt.show()
```

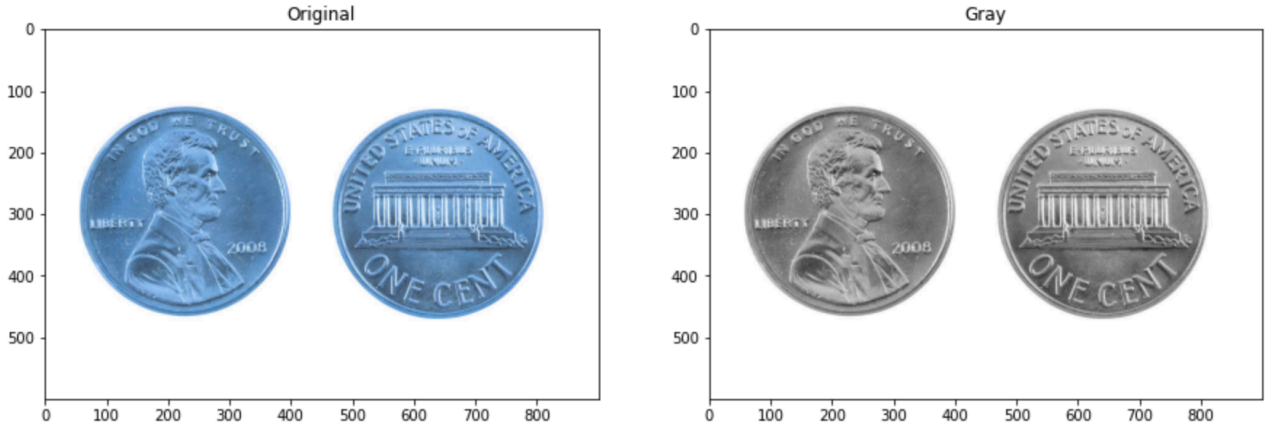


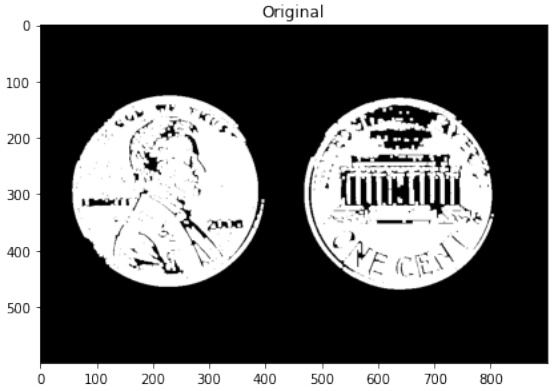
Image Segmentation

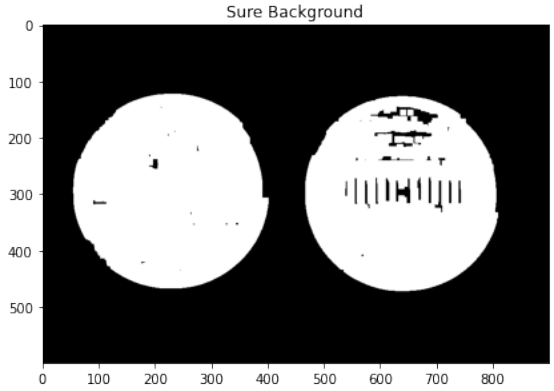
The Watershed Algorithm

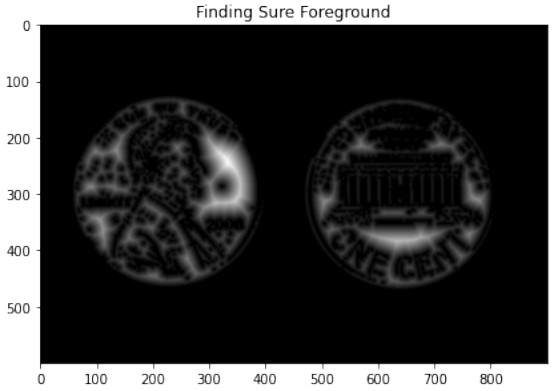
```
# noise removal
kernel = np.ones((3,3),np.uint8)
opening = cv.morphologyEx(thresh,cv.MORPH_OPEN,kernel, iterations = 2)

# sure background area
sure_bg = cv.dilate(opening,kernel,iterations=3)

# Finding sure foreground area
dist_transform = cv.distanceTransform(opening,cv.DIST_L2,5)
ret, sure_fg = cv.threshold(dist_transform,0.7*dist_transform.max(),255,0)

# Finding unknown region
sure_fg = np.uint8(sure_fg)
unknown = cv.subtract(sure_bg,sure_fg)

fig = plt.figure(figsize=(15, 15))
plt.subplot(2,2,1),plt.imshow(opening,cmap = 'gray'), plt.title('Original')
plt.subplot(2,2,2),plt.imshow(sure_bg,cmap = 'gray'), plt.title('Sure Background')
plt.subplot(2,2,3),plt.imshow(dist_transform,cmap = 'gray'), plt.title('Finding Sure Foreground')
plt.subplot(2,2,4),plt.imshow(unknown,cmap = 'gray'), plt.title('Finding Unknown Region')
```

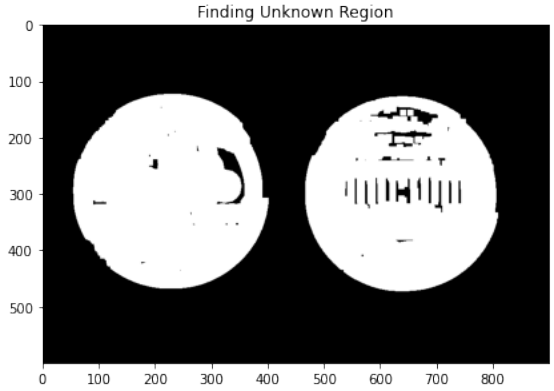
Image Segmentation

The Watershed Algorithm









Feature Detection and Description

Corner Detection

One early attempt to find the corners was done by **Chris Harris & Mike Stephens** in their paper "**A Combined Corner and Edge Detector"** in 1988, so now it is called the Harris Corner Detector. He took this simple idea to a mathematical form. It basically finds the difference in intensity for a displacement of (u,v) in all directions. This is expressed as below:

$$E(u,v) = \sum_{x,y} \underbrace{w(x,y)}_{\text{window function}} \underbrace{[I(x+u,y+v)}_{\text{shifted intensity}} - \underbrace{I(x,y)}_{\text{intensity}}]^2$$

The window function is either a rectangular window or a Gaussian window which gives weights to pixels underneath.

We have to maximize this function E(u,v) for corner detection. That means we have to maximize the second term. Applying Taylor Expansion to the above equation and using some mathematical steps (please refer to any standard text books you like for full derivation

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$$

Here, I_x and I_y are image derivatives in x and y directions respectively. (These can be easily found using **cv.Sobel()**).

Then comes the main part. After this, they created a score, basically an equation, which determines if a window can contain a corner or not.

$$R = \det(M) - k(\text{trace}(M))^2$$

Feature Detection and Description

Corner Detection

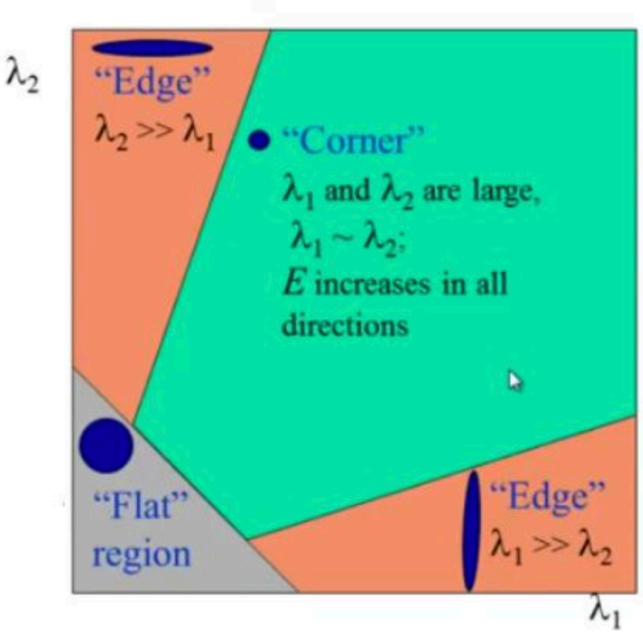
where

- $\det(M) = \lambda_1 \lambda_2$
- $\text{trace}(M) = \lambda_1 + \lambda_2$
- λ_1 and λ_2 are the eigenvalues of M

So the magnitudes of these eigenvalues decide whether a region is a corner, an edge, or flat.

- When $|R|$ is small, which happens when λ_1 and λ_2 are small, the region is flat.
- When $R < 0$, which happens when $\lambda_1 >> \lambda_2$ or vice versa, the region is edge.
- When R is large, which happens when λ_1 and λ_2 are large and $\lambda_1 \sim \lambda_2$, the region is a corner.

It can be represented in a nice picture as follows:

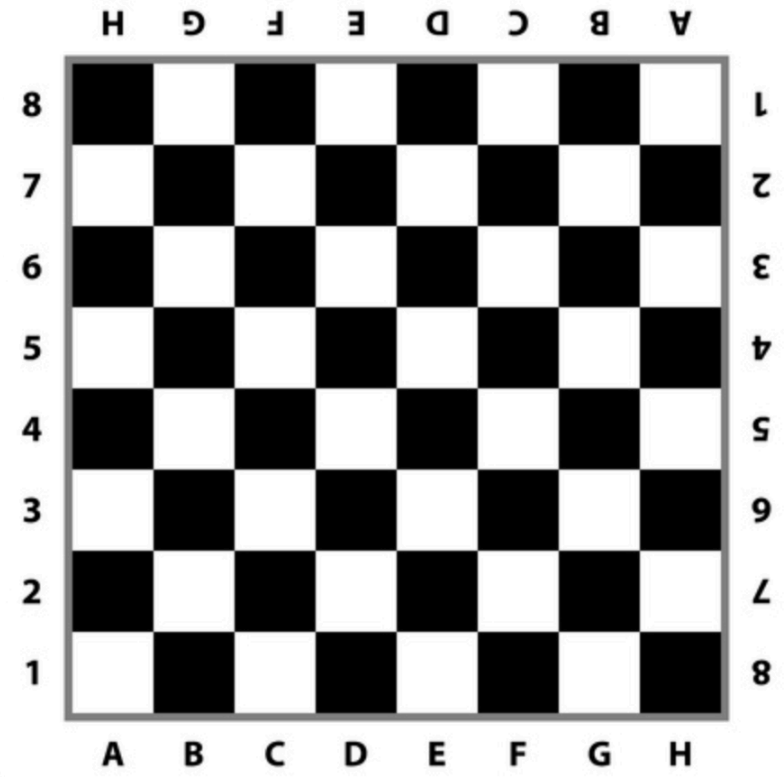


image

Feature Detection and Description

Corner Detection

One early attempt to find the corners was done by **Chris Harris & Mike Stephens** in their paper "**A Combined Corner and Edge Detector"** in 1988, so now it is called the Harris Corner Detector. He took this simple idea to a mathematical form. It basically finds the difference in intensity for a displacement of (u,v) in all directions.



Feature Detection and Description

Corner Detection

One early attempt to find the corners was done by **Chris Harris & Mike Stephens** in their paper "**A Combined Corner and Edge Detector"** in 1988, so now it is called the Harris Corner Detector. He took this simple idea to a mathematical form. It basically finds the difference in intensity for a displacement of (u,v) in all directions

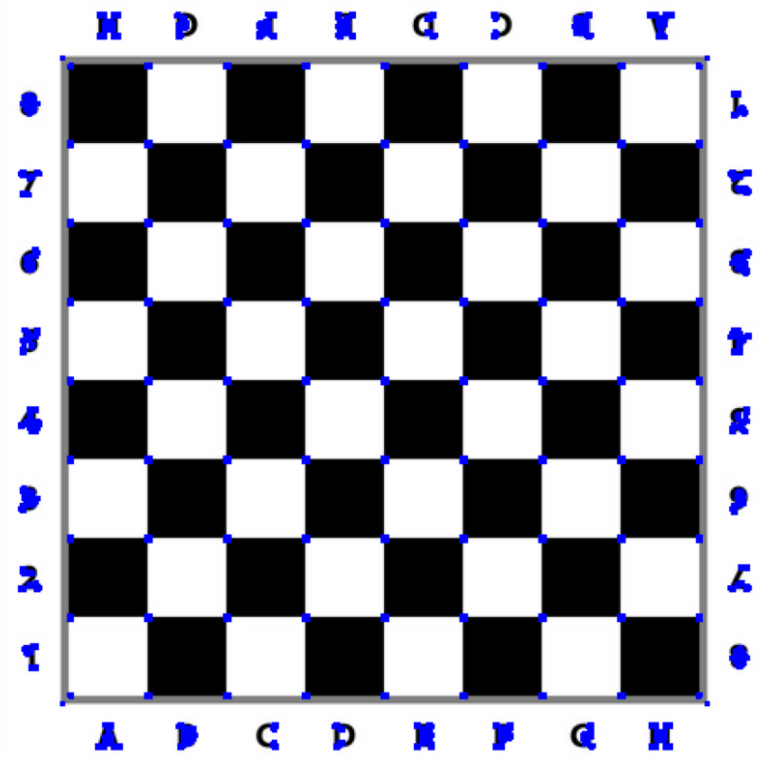
```
import numpy as np
import cv2 as cv
filename = '../Images/chessboard_corners.jpg'
img = cv.imread(filename)

fig = plt.figure(figsize=(6, 6))
plt.plot(), plt.imshow(img),plt.title('Original')
```

```
gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)
gray = np.float32(gray)
dst = cv.cornerHarris(gray,2,3,0.04)
#result is dilated for marking the corners, not important
dst = cv.dilate(dst,None)
# Threshold for an optimal value, it may vary depending on the image.
img[dst>0.01*dst.max()]=[0,0,255]

fig = plt.figure(figsize=(6,6))
plt.plot(), plt.imshow(img),plt.title('Corner Detection')
```

Feature Detection and Description

Corner Detection

One early attempt to find the corners was done by **Chris Harris & Mike Stephens** in their paper "**A Combined Corner and Edge Detector"** in 1988, so now it is called the Harris Corner Detector. He took this simple idea to a mathematical form. It basically finds the difference in intensity for a displacement of (u,v) in all directions



Feature Detection and Description

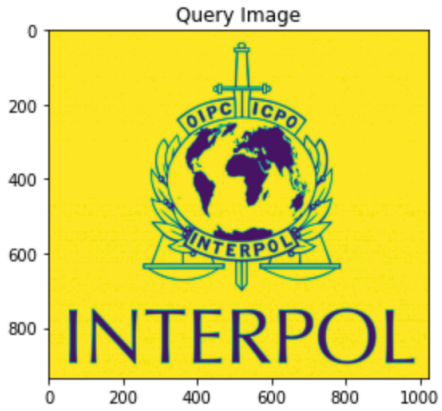
Feature Matching

The objective of feature mapping is to match features in one image with others. Brute-Force matcher is simple. It takes the descriptor of one feature in first set and is matched with all other features in second set using some distance calculation. And the closest one is returned. Let's look at an example-

```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
img1 = cv.imread('../Images/interpol-logo.jpg',cv.IMREAD_GRAYSCALE)
img2 = cv.imread('../Images/interpol_file.jpg',cv.IMREAD_GRAYSCALE)
# Initiate ORB detector
orb = cv.ORB_create()
# find the keypoints and descriptors with ORB
kp1, des1 = orb.detectAndCompute(img1,None)
kp2, des2 = orb.detectAndCompute(img2,None)
```

```
plt.plot(), plt.imshow(img1),plt.title('Query Image')
```

```
([],
 <matplotlib.image.AxesImage at 0x147587310>,
 Text(0.5, 1.0, 'Query Image'))
```



Feature Detection and Description

Feature Matching

The objective of feature mapping is to match features in one image with others. Brute-Force matcher is simple. It takes the descriptor of one feature in first set and is matched with all other features in second set using some distance calculation. And the closest one is returned. Let's look at an example-

```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
img1 = cv.imread('../Images/interpol-logo.jpg',cv.IMREAD_GRAYSCALE)
img2 = cv.imread('../Images/interpol_file.jpg',cv.IMREAD_GRAYSCALE)
# Initiate ORB detector
orb = cv.ORB_create()
# find the keypoints and descriptors with ORB
kp1, des1 = orb.detectAndCompute(img1,None)
kp2, des2 = orb.detectAndCompute(img2,None)
```

```
plt.plot(), plt.imshow(img2),plt.title('Search Image')
```

```
([],
 <matplotlib.image.AxesImage at 0x147637e50>,
 Text(0.5, 1.0, 'Search Image'))
```

Feature Detection and Description

Feature Matching

The objective of feature mapping is to match features in one image with others. Brute-Force matcher is simple. It takes the descriptor of one feature in first set and is matched with all other features in second set using some distance calculation. And the closest one is returned. Let's look at an example-

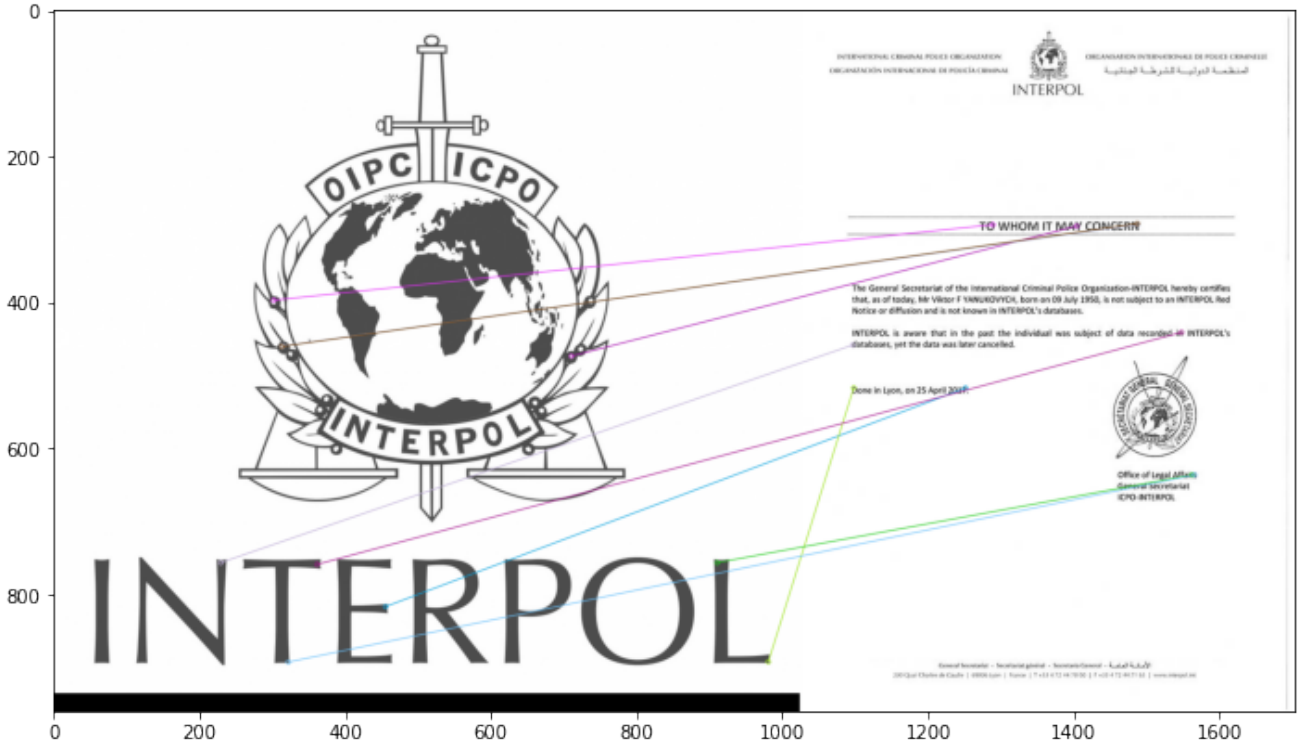
```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
img1 = cv.imread('../Images/interpol-logo.jpg',cv.IMREAD_GRAYSCALE)
img2 = cv.imread('../Images/interpol_file.jpg',cv.IMREAD_GRAYSCALE)
# Initiate ORB detector
orb = cv.ORB_create()
# find the keypoints and descriptors with ORB
kp1, des1 = orb.detectAndCompute(img1,None)
kp2, des2 = orb.detectAndCompute(img2,None)
```

```
# create BFMatcher object
bf = cv.BFMatcher(cv.NORM_HAMMING, crossCheck=True)
# Match descriptors.
matches = bf.match(des1,des2)
# Sort them in the order of their distance.
matches = sorted(matches, key = lambda x:x.distance)
# Draw first 10 matches.
fig = plt.figure(figsize=(12, 12))
img3 = cv.drawMatches(img1,kp1,img2,kp2,matches[:10],None,flags=cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
plt.imshow(img3),plt.show()
```

Feature Detection and Description

Feature Matching

The objective of feature mapping is to match features in one image with others. Brute-Force matcher is simple. It takes the descriptor of one feature in first set and is matched with all other features in second set using some distance calculation. And the closest one is returned. Let's look at an example-



Feature Detection and Description

Feature Matching

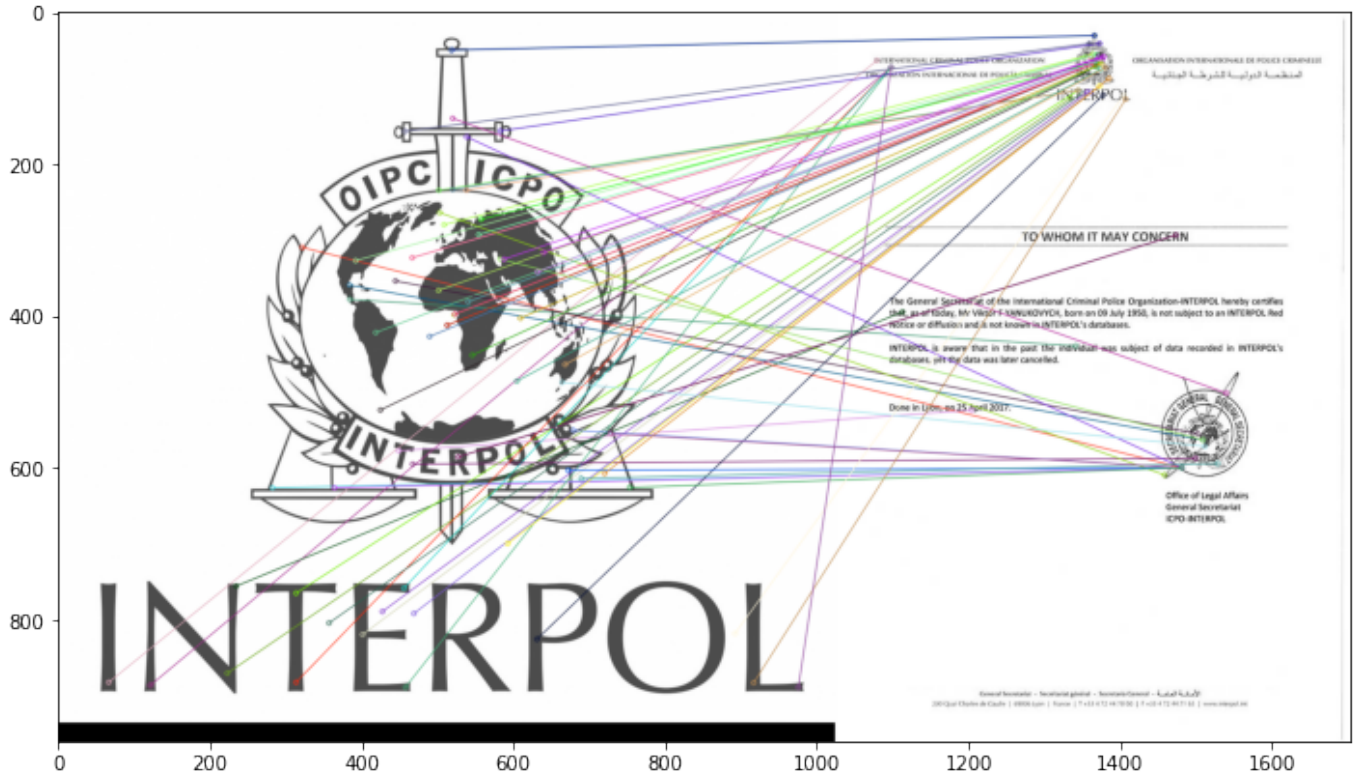
The objective of feature mapping is to match features in one image with others. Brute-Force matcher is simple. It takes the descriptor of one feature in first set and is matched with all other features in second set using some distance calculation. And the closest one is returned. Let's look at an example-

```
img1 = cv.imread('../Images/interpol-logo.jpg',cv.IMREAD_GRAYSCALE)
img2 = cv.imread('../Images/interpol_file.jpg',cv.IMREAD_GRAYSCALE)

# Initiate SIFT detector
sift = cv.SIFT_create()
# find the keypoints and descriptors with SIFT
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)
# BFMatcher with default params
bf = cv.BFMatcher()
matches = bf.knnMatch(des1,des2,k=2)

# Apply ratio test
good = []
for m,n in matches:
    if m.distance < 0.75*n.distance:
        good.append([m])
# cv.drawMatchesKnn expects list of lists as matches.
img3 = cv.drawMatchesKnn(img1,kp1,img2,kp2,good,None,flags=cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
fig = plt.figure(figsize=(12, 12))
plt.imshow(img3),plt.show()
```

Feature Detection and Description

Feature Matching

The objective of feature mapping is to match features in one image with others. Brute-Force matcher is simple. It takes the descriptor of one feature in first set and is matched with all other features in second set using some distance calculation. And the closest one is returned. Let's look at an example-



Feature Detection and Description

Computational Photography

Often, the need to remove noise created by some understandable patterns is felt. It can be easily done with the help of masking and then using INPAINT method.

```
img = cv.imread('messi.jpg')
mask = cv.imread('mask.png',0)
dst = cv.inpaint(img,mask,3,cv.INPAINT_TELEA)

plt.plot(), plt.imshow(img),plt.title('Original')
plt.plot(), plt.imshow(mask),plt.title('Masked')
plt.plot(), plt.imshow(dst),plt.title('Algorithm 1')
```

Feature Detection and Description

Computational Photography

Often, the need to remove noise created by some understandable patterns is felt. It can be easily done with the help of masking and then using INPAINT method.

```
img = cv.imread('messi.jpg')
mask = cv.imread('mask.png',0)
dst = cv.inpaint(img,mask,3,cv.INPAINT_TELEA)

plt.plot(), plt.imshow(img),plt.title('Original')
plt.plot(), plt.imshow(mask),plt.title('Masked')
plt.plot(), plt.imshow(dst),plt.title('Algorithm 1')
```

Feature Detection and Description

Computational Photography

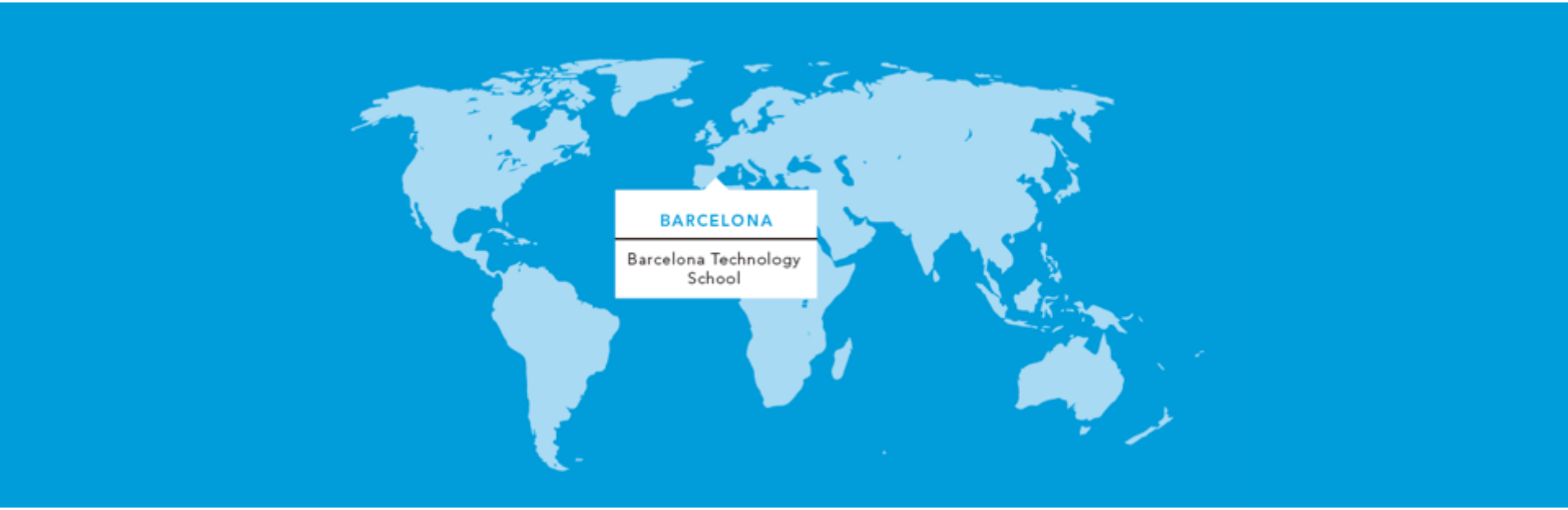
Often, the need to remove noise created by some understandable patterns is felt. It can be easily done with the help of masking and then using INPAINT method. Similar to INPAINT methods, we also have INPAINT_NS method.

```
img = cv.imread('messi.jpg')
mask = cv.imread('mask.png',0)
dst = cv.inpaint(img,mask,3,cv.INPAINT_NS)

plt.plot(), plt.imshow(img),plt.title('Original')
plt.plot(), plt.imshow(mask),plt.title('Masked')
plt.plot(), plt.imshow(dst),plt.title('Algorithm 2')
```

Thank you!

We appreciate your patience and interest..

Questions??


BARCELONA
Barcelona Technology School