East West University

Department of CSE

Lab Report 02

CSE 438

Digital Image Processing

Submitted To:

Md. Mahir Ashhab

Lecturer

Department of Computer Science and Engineering

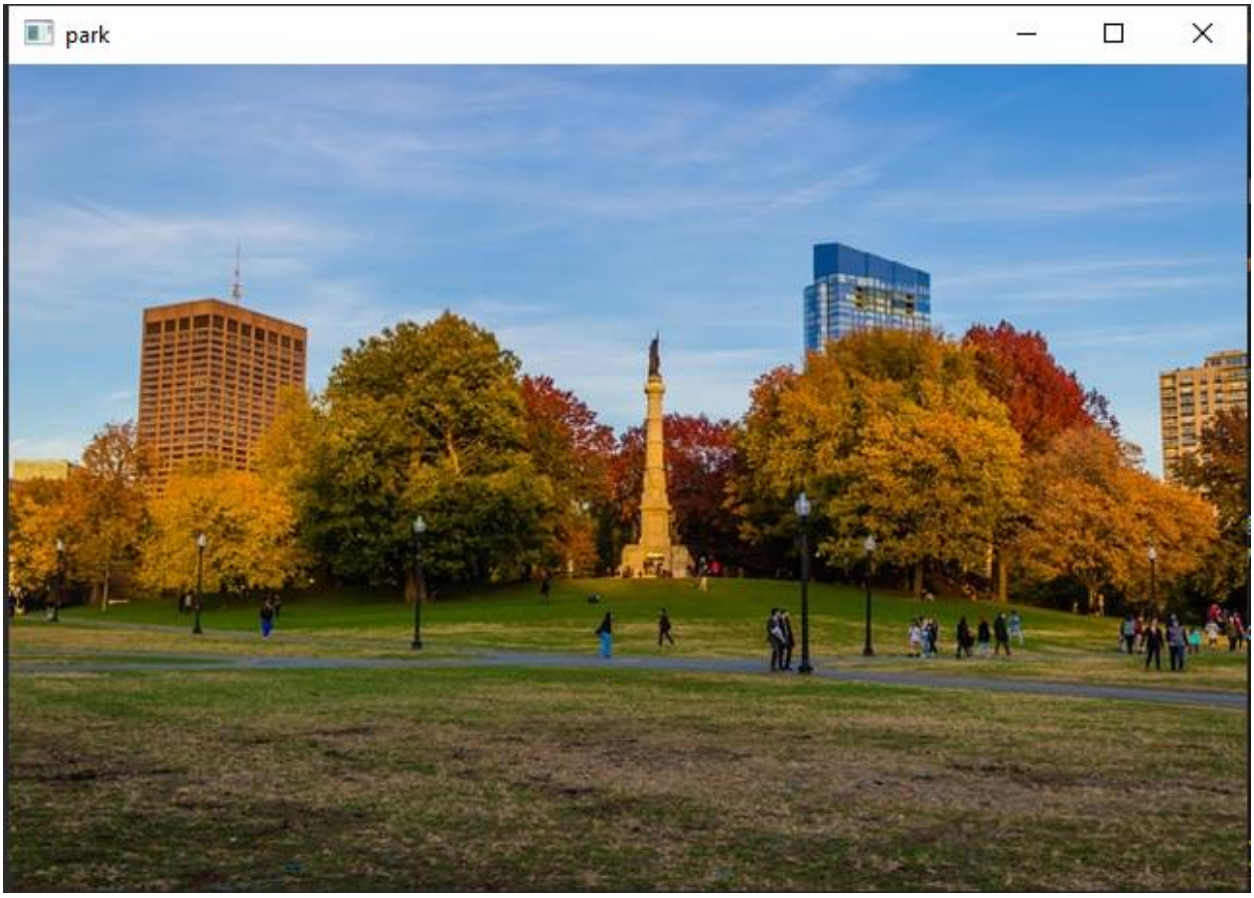
Submitted By:

Adri Saha

ID: 2019-1-60-024

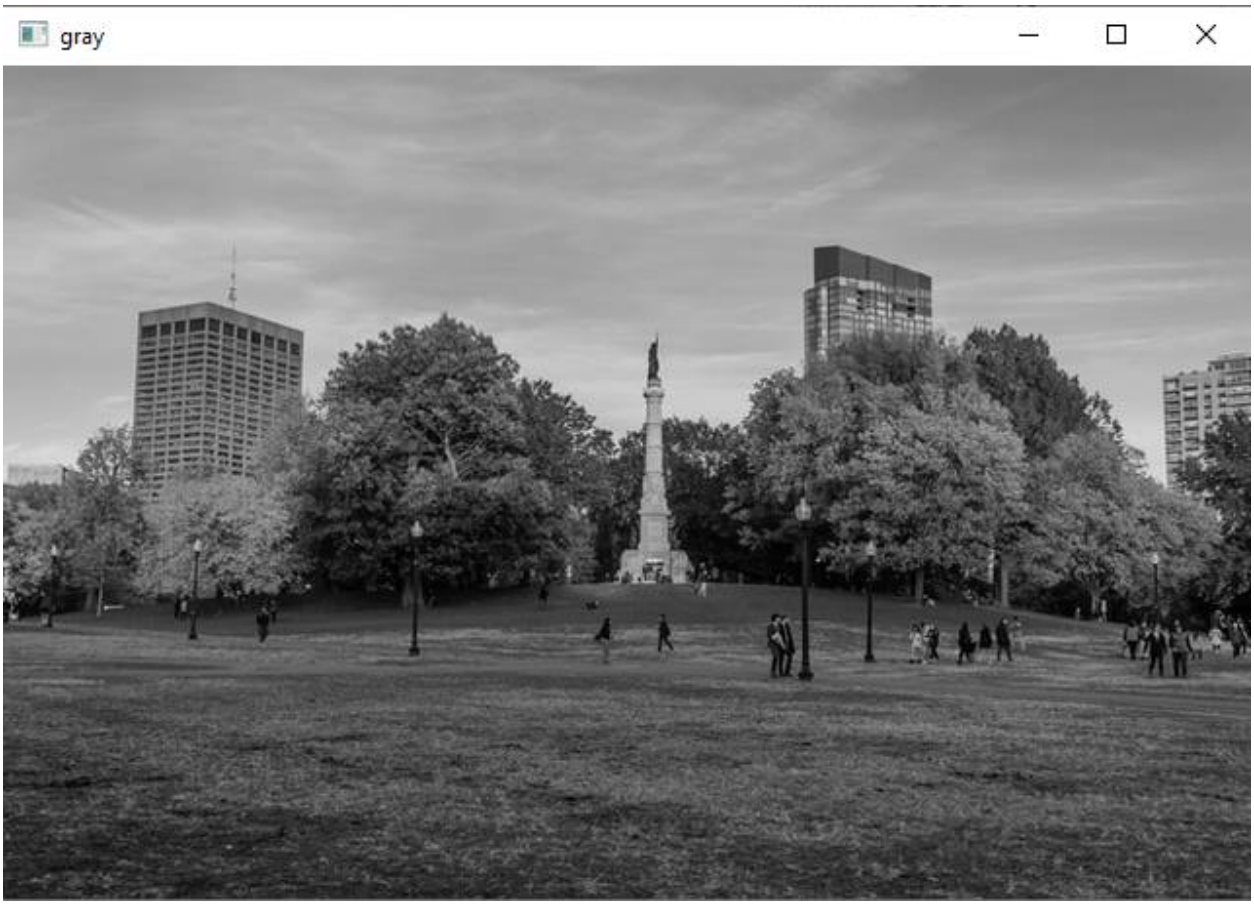
Submission Date: 01 August 2022

Actual image of Boston Park



Grayscale conversion:

```
gray=cv.cvtColor(img,cv.COLOR_BGR2GRAY)
cv.imshow('gray',gray)
```



Blurring effect

We know, there are 5 types of blurring effects:

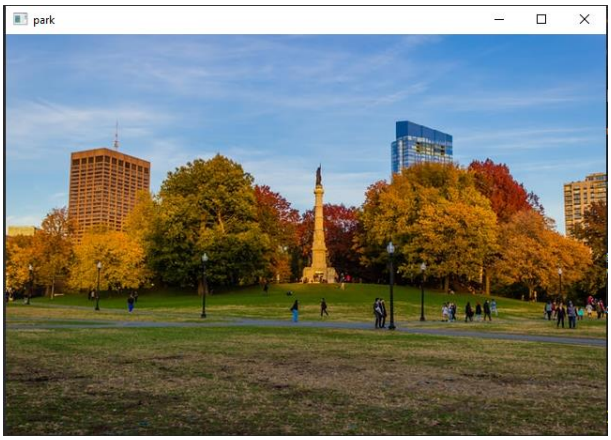
1. Blur with a box filter
2. Gaussian Blurring.
3. Average/Mean Blurring.
4. Bilateral Blurring.
5. Median Blurring.

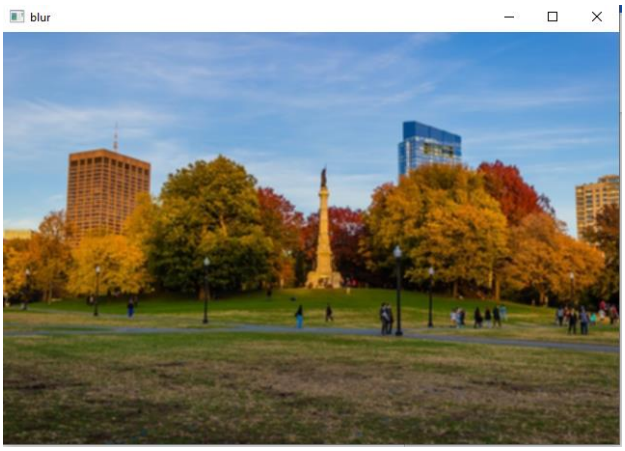
1. **Box Blur:** A box blur, often referred to as a box linear filter, is a spatial domain linear filter in which the average value of each pixel in the output picture is used to determine the value of every pixel in the input image. It is a kind of low pass filter that blurs images.

   Code:

```
blur = cv.boxFilter(img,ksize=(3,3),ddepth=-1,anchor=(-1,-1))

cv.imshow('blur',blur)
```





Actual image Blurred image

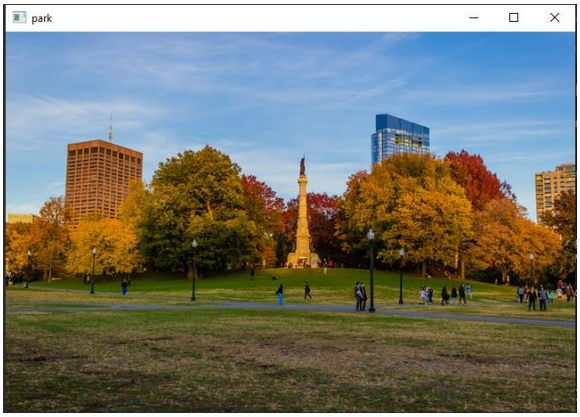
2. **Gaussian blur:** A Gaussian filter is a low pass filter used to blur certain areas of a picture and reduce noise (high frequency components).

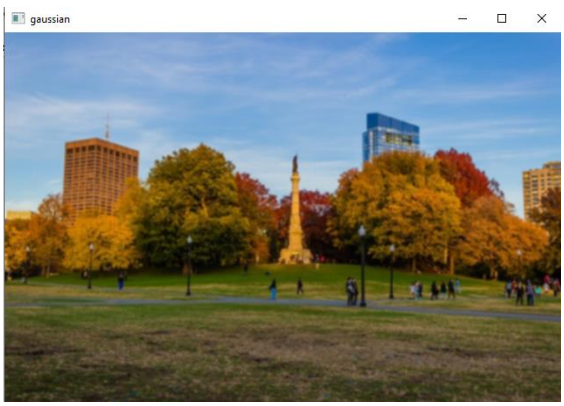
Gaussian Blurring: It is easy to implement. It is implemented using a convolution kernel. A weighted kernel can also be used which will give a different blur effect on the image. This algorithm can be slow as its processing time is dependent on the size of the image and the size of the kernel that is to be selected.

Code:

```
gaussian = cv.GaussianBlur(img,ksize=(5,5),sigmaX=0)
cv.imshow('gaussian',gaussian)
```

The built-in function cv.GaussianBlur takes 3 parameters- the source image, a 5*5 sized kernel and sigmaX represents standard deviation in X-direction.





Actual image Gaussian Blurred image

Canny edge detection:

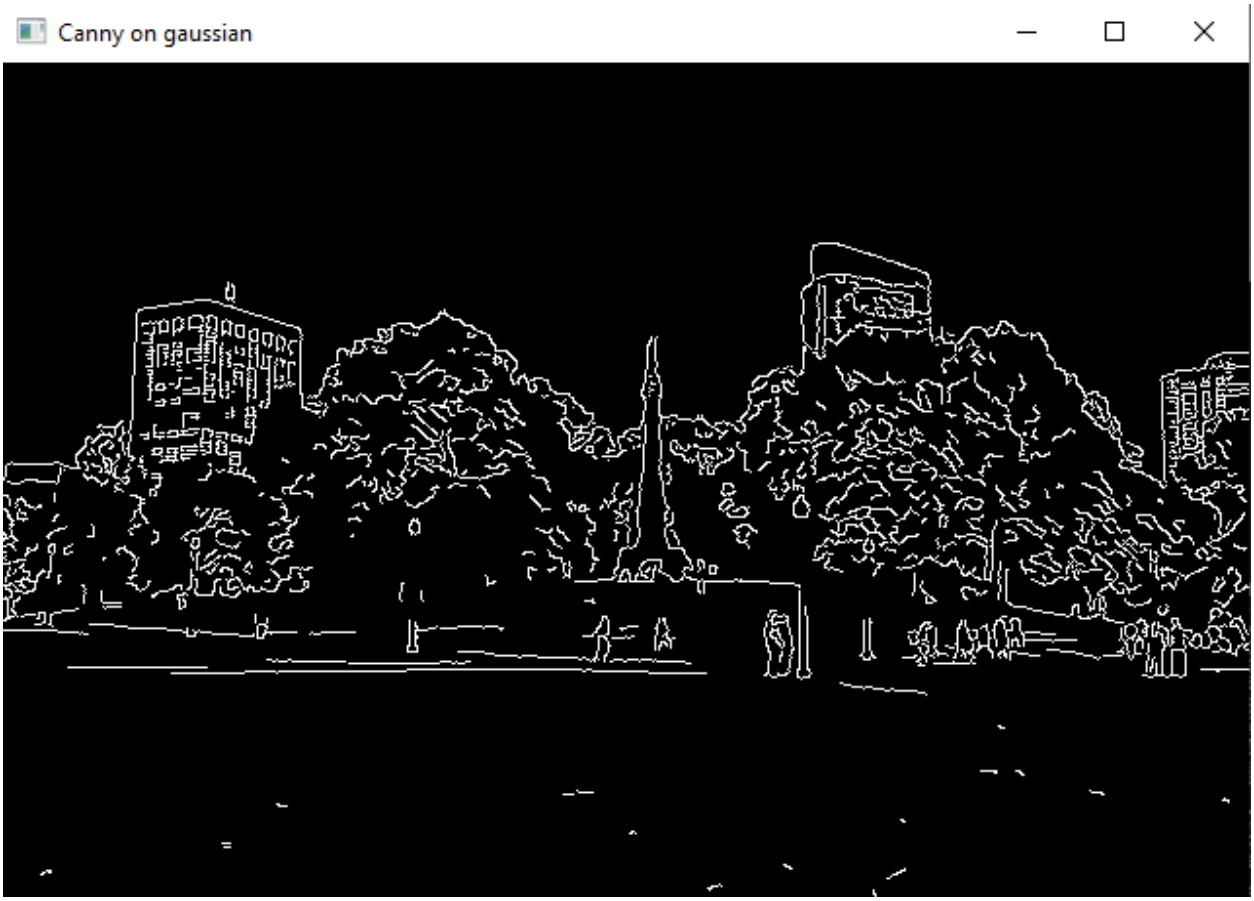
A multi-stage method is used by the Canny edge detector, an edge detection operator, to identify a variety of edges in pictures. The smart edge detection first smooths the image to reduce noise. Then it locates the gradient in the picture to highlight the areas with high spatial derivatives.

```
canny = cv.Canny(img,125,175)
cv.imshow('canny',canny)
```

Canny on gaussian:

Canny edge detection evaluates the edge strength and direction for each pixel in the noise-smoothed picture using linear filtering with a Gaussian kernel. The pixels that endure a process of thinning known as non-maximal suppression are those that are used to identify candidate edge pixels.

```
canny_gaussian = cv.Canny(gaussian,125,175)
cv.imshow('Canny on gaussian', canny_gaussian)
```



3. Average/Mean blur:

It takes a region of pixels around a center pixel, averages all of these pixels, and then replaces the core pixel with the average.

Let us consider an image-

2	4	5
1	2	3
6	2	2

And a kernel of 3*3 size as below-

1/9

1	1	1
1	1	1
1	1	1

1/9*[(2*1) +(4*1) +(5*1) +(1*1) +(2*1) +(3*1) +(6*1) +(2*1) +(2*1)] = 3

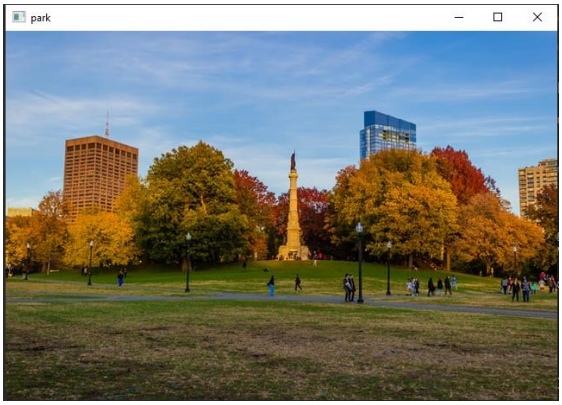
So, the resultant image will be-

2	4	5
1	3	3
6	2	2

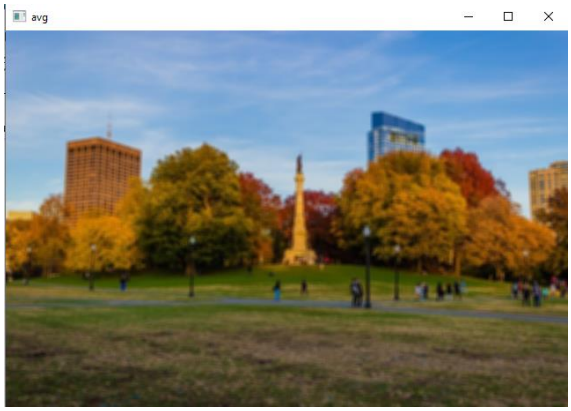
Code:

```
avg_blur= cv.blur(img,ksize=(5,5))
cv.imshow('avg',avg_blur)
```

In this case, the average blurred image was found using the built-in cv.blur technique. To find the blurred picture, the function requires two parameters: the original image and a 5x5 kernel.



Actual image



Average Blurred image

Median Blur: The median blur function averages data, just like other averaging methods. In this example, the new center component of the image is the median of all the pixels in the kernel region. While processing the edges, this technique removes the noise.

Let's consider the image below-

1	4	2
6	3	5
8	9	7

Here, we must first find the median and then replace the pixel value 4 with the median value.

After sorting- 1, 2, 3, 4, 5, 6, 7, 8, 9

Median value- 5. So, 4 will be replaced by 5.

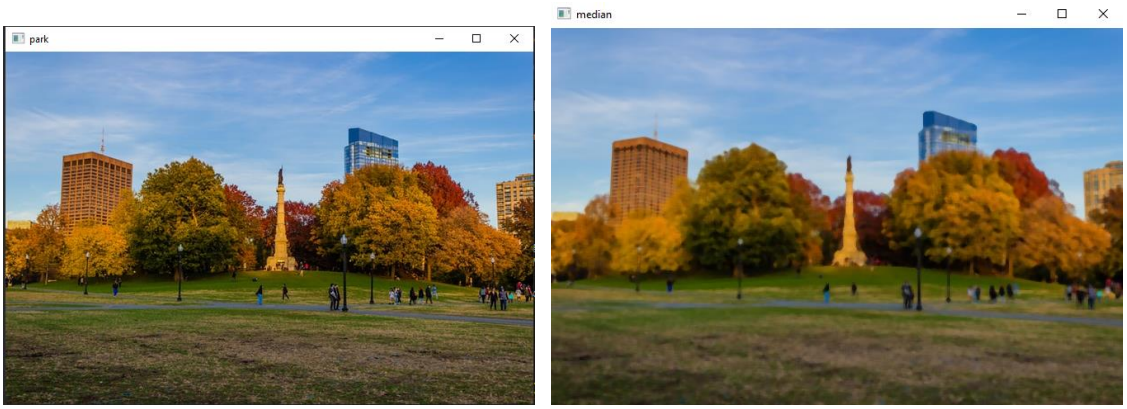
Resultant image-

1	4	2
6	5	5
8	9	7

Code:

```
median_blur= cv.medianBlur(img,ksize=5)
cv.imshow('median',median_blur)
```

To find the median blurred image, used the built-in function cv.medainBlur(). It generates the output using the kernel size and source image as inputs.

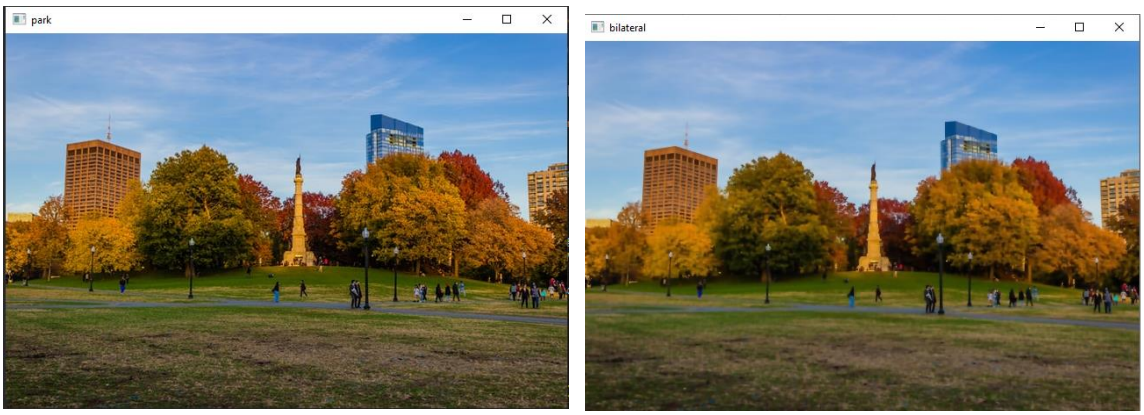


Actual image Median Blurred image

Bilateral blur: This filter does both smooths and preserves the edges of image.

```
bilateral= cv.bilateralFilter(img,d=5, sigmaColor=100, sigmaSpace=100)
cv.imshow('bilateral',bilateral)
```

4 parameters are passed bilateral filter in function cv.bilateralFilter (), including the source picture, d: the distance we are prepared to consider, sigmacolor: the color standard deviation. Sigmaspace: it symbolizes the filter sigma in the coordinate space. The more it increases, the more the surrounding colors will blend to form one color.



Actual image Bilateral image

From the above analysis,

1. We can say **Bilateral blurred image seems natural**. A non-linear, edge-preserving, and noise-reduction smoothing filter for images is known as a bilateral filter. It switches out each pixel's intensity for a weighted average of intensity values from adjacent pixels. This weight may have a Gaussian distribution as its basis. Above all the filters, the bilateral blurred image looks more natural.
2. **Average Filter is not good for real-life use**. The median filter is better than an average filter. Because in averaging filter for one specific large or small, huge, differenced value, whole mean value changed dramatically. But in the median, we must sort all values, then must select a median value. So, there is no huge change for a specific value.
3. **For heavy noise cancellation median blur is used:** Impulse noise, sometimes known as "salt and pepper" noise, is easily eliminated from images using a median blur. The median filter works by replacing each pixel's gray level with the median of the gray levels in its immediate vicinity rather than by averaging them.

Dilated:

```
dilated = cv.dilate(canny_gaussian,(7,7))

cv.imshow('dilated',dilated)
```

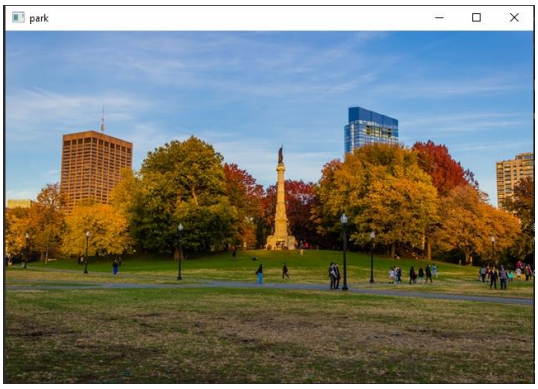
Eroded:

```
eroded = cv.erode(dilated,(7,7))
cv.imshow('eroded',eroded)
```

Resizing

1. **Cropped:**

```
cropped = img[50:400, 100:400]
cv.imshow('cropped', cropped)
```

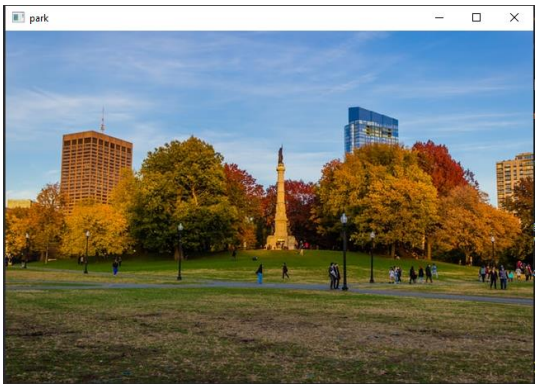


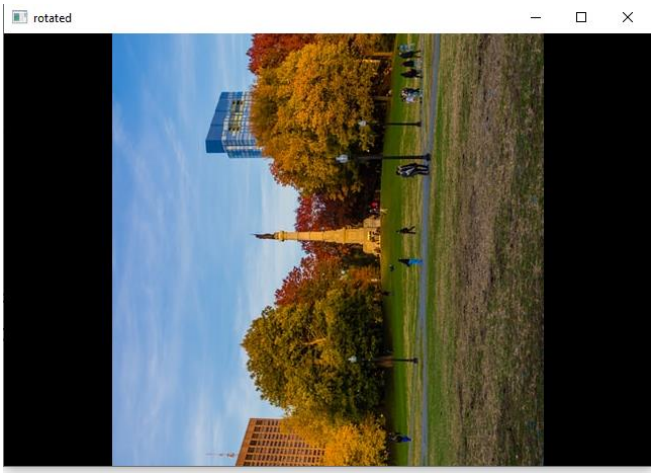
Actual image

Cropped image

2. **Rotation:**
   All parallel lines from the source image are preserved as parallel lines in the output image by WarpAffine.
   Code:

```
def rotate(img,angle,center=None):
    width = img.shape[1]
    height = img.shape[0]
    if center is None:
        center = (width//2,height//2)
    rotMatrix = cv.getRotationMatrix2D(center,angle,scale=1)
    dim = (width,height)
    return cv.warpAffine(img,rotMatrix,dim)
rotated = rotate(img,90)
cv.imshow('rotated', rotated)
```



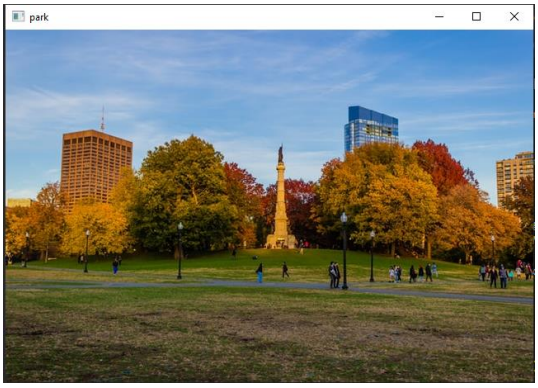


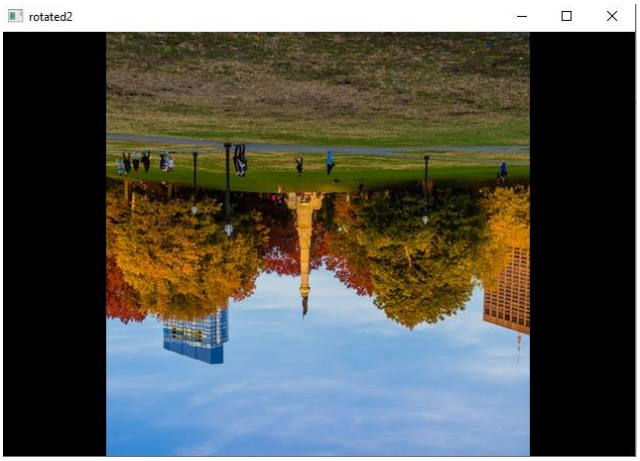
Actual image Rotated image

3. **Rotation 2:**

rotated2 = rotate(rotated,90)

cv.imshow('rotated2', rotated2)



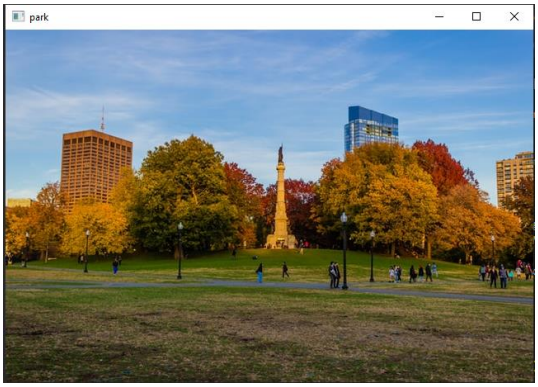


Actual image 2nd rotated image

4. **Flip:**

flip = cv.flip(img,-1)

cv.imshow('flipped',flip)



Actual image Flipped image

5. **Translated**

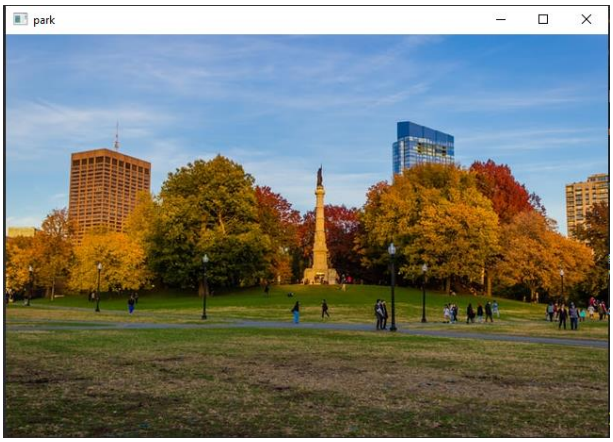
It usually refers to moving or changing the location of a picture.

Generally this matrix is used for translation:

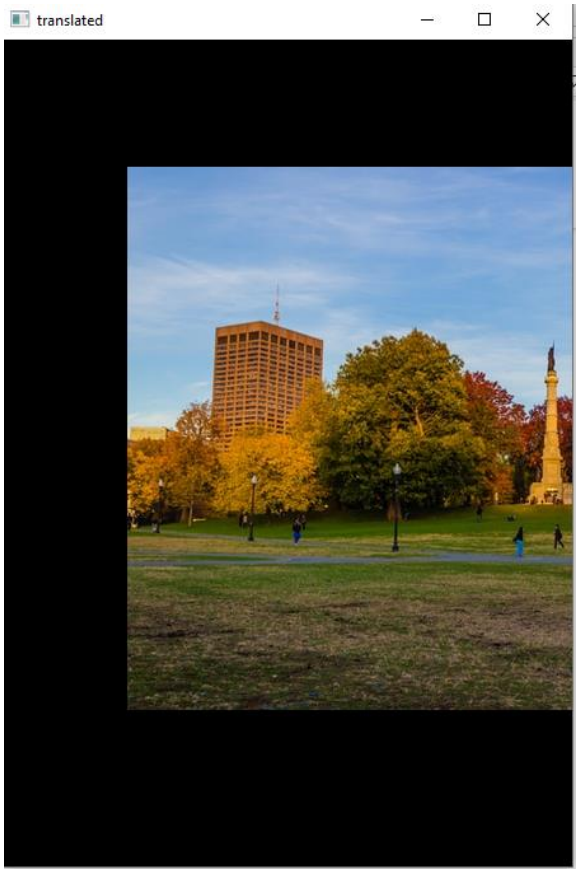
$$\begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \end{bmatrix}$$

tx – shift along the x-axis, ty- shift along the y-axis.

```
matrix=np.float32([[1,0,100],[0,1,100]])
translated = cv.warpAffine(img,matrix,(450,650))
cv.imshow('translated', translated)
```



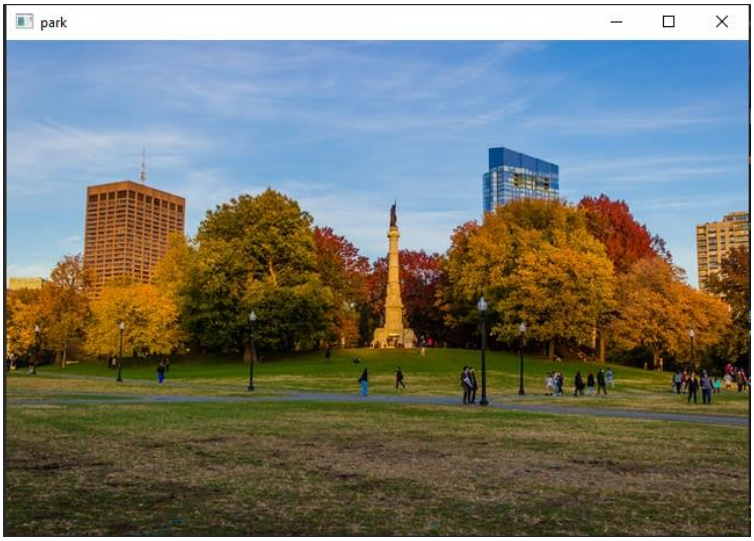
Actual image



Translated image

6. **Resizing:**

```
imageResize= cv.resize(img,(300,500))
cv.imshow('Resized Image', imageResize)
cv.waitKey(0)
```



Actual image

Resized image