# Introduction to Computer Vision Systems

- Second LAP

1. Color images as a functions

### Importing required libraries

In [1]:

```
import numpy as np
from copy import deepcopy
from matplotlib import pyplot as plt
```

2. Addition from another perspective

In [9]:

```
bicycle = plt.imread('bicycle.jpg')
dolphin = plt.imread('dolphin.jpg')

plt.imshow(bicycle, cmap='gray')
plt.axis('off')
plt.show()
plt.imshow(dolphin, cmap='gray')
plt.axis('off')
plt.show()
```

### Addition

In [3]:

```
Sum =  dolphin + bicycle
plt.imshow(Sum, cmap='gray')
plt.axis('off')
plt.show()
```

In [83]:

```
Sum_alt =  (dolphin/2) + (bicycle/2)
plt.imshow(Sum_alt, cmap='gray')
plt.axis('off')
plt.show()
```

In [5]:

```
Avg =  (dolphin + bicycle) / 2
plt.imshow(Avg, cmap='gray')
plt.axis('off')
plt.show()
```

## What?!!

**wait a second..**

$$\frac{183}{} \quad \frac{152}{} \qquad \frac{183 + 152}{}$$

is.. $(2) + (2)$ is equal to.. $(2)$

..

3. Bliding images

In [38]:

```
result =  dolphin * 0.5 + bicycle * 0.8
plt.imshow(result, cmap='gray')
plt.axis('off')
plt.show()
```

## Building a function for blinding

In [48]:

```
def blind(img1, img2, alpha):
    result = img1 * alpha + img2 * (1-alpha)
    return result
```

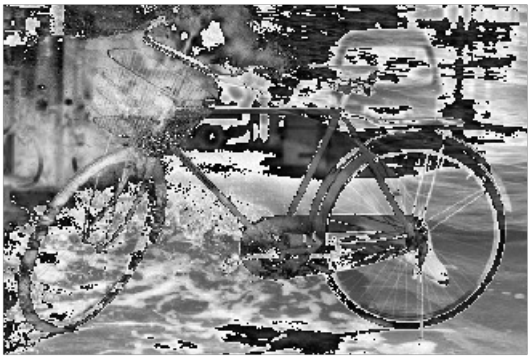
In [50]:

```
result = blind(bicycle,dolphin,0.8)
plt.imshow(result, cmap='gray')
plt.axis('off')
plt.show()
```

4. Diffrence

In [53]:

```
plt.imshow(dolphin-bicycle, cmap='gray')
plt.axis('off')
plt.show()
```



In [60]:

```
plt.imshow(abs(dolphin-bicycle), cmap='gray')
plt.axis('off')
plt.show()
```

In [90]:

```
plt.imshow((dolphin.astype('float')-bicycle.astype('float')), cmap='gray')
plt.axis('off')
plt.show()
```

5. Generate Gaussian Noise

### rand() functions

In [110]:

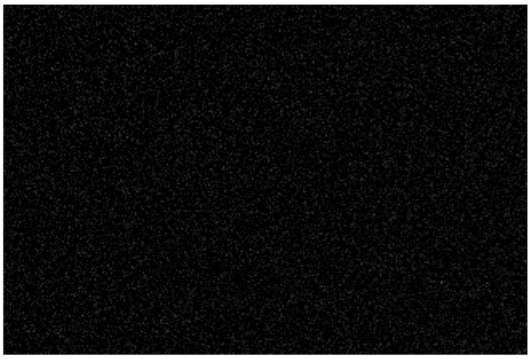
```
#rand for random numbers
print(np.random.rand(5))
#randint for random integers
print(np.random.randint(5, size=10))
#randn for random gaussian
print(np.random.randn(5))
```

```
[0.67554694 0.61403949 0.6160472  0.84480451 0.49640922]
[2 3 0 1 0 3 1 0 3 0]
[ 1.32852707  0.16809302  1.07495479 -0.35379172  1.18838478]
```

### Creating Gaussian Noise

In [15]:

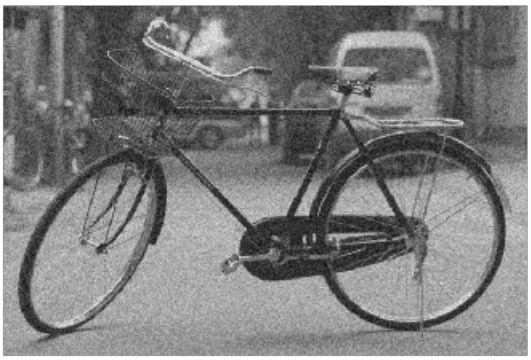
```
Sigma = 20
Gaussian = np.random.randn(400,600)*Sigma
plt.imshow(Gaussian, cmap='gray', vmin=0, vmax=255)
plt.axis('off')
plt.show()
```



### Adding Gaussian Noise to image

In [16]:

```
plt.imshow(bicycle + Gaussian, cmap='gray')
plt.axis('off')
plt.show()
```

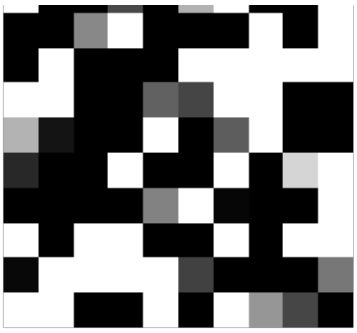


6. Smoothing the images

### Creating Noise

In [575]:

```
Sigma = 1000
Gaussian = np.random.randn(10,10)*Sigma
plt.imshow(Gaussian, cmap='gray', vmin=-255, vmax=255)
plt.axis('off')
plt.show()
```

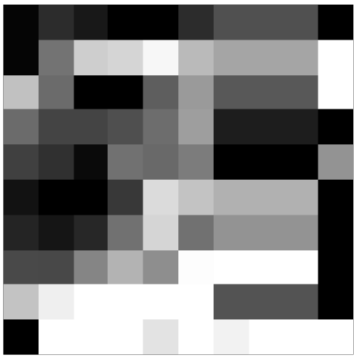


### 1. Using Moving Average Filter

In [570]:

```
MAF = deepcopy(Gaussian)
for i in range(0,7,1):
    for j in range(0,7, 1):
        Img = Gaussian[i:i+3,j:j+3]
        Filter =  (1/9)*np.matrix('1 1 1; 1 1 1; 1 1 1')
        MAF[i:i+3,j:j+3] = Img * Filter

plt.imshow(MAF, cmap='gray', vmin=-225, vmax=255)
plt.axis('off')
plt.show()
```

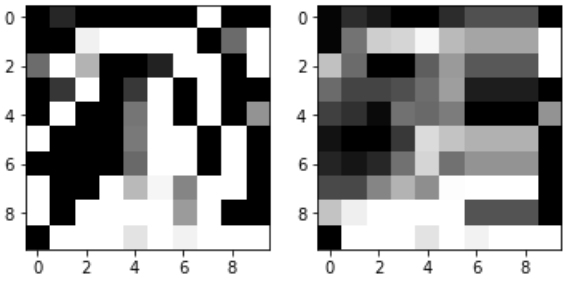


In [571]:

```
fig, (ax1, ax2) = plt.subplots(1, 2)
ax1.imshow(Gaussian, cmap='gray', vmin=-225, vmax=255)
ax2.imshow(MAF, cmap='gray', vmin=-225, vmax=255)
```

Out[571]:

```
<matplotlib.image.AxesImage at 0x24923e76a08>
```



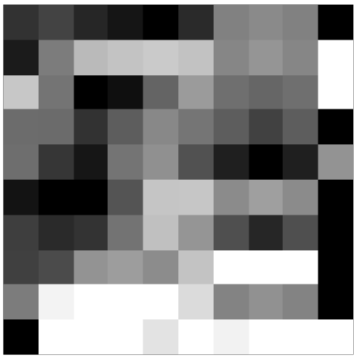

### 2. Using Weighted Average Filter

In [572]:

```
WAF = deepcopy(Gaussian)
for i in range(0,7,1):
```

```
for i in range(0,7,1):
    for j in range(0,7, 1):
        Img = Gaussian[i:i+3,j:j+3]
        Filter =  (1/16)* np.matrix('1 2 1; 2 4 2; 1 2 1')
        WAF[i:i+3,j:j+3] = Img * Filter

plt.imshow(WAF, cmap='gray', vmin=-225, vmax=255)
plt.axis('off')
plt.show()
```



### Comparison

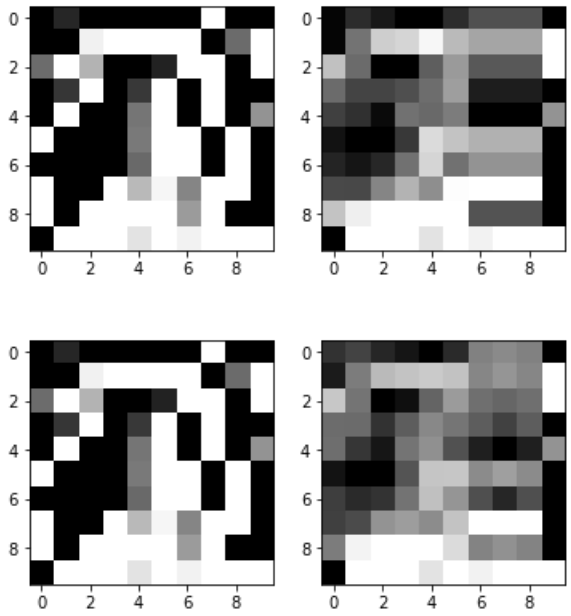
In [573]:

```
fig, (ax1, ax2) = plt.subplots(1, 2)
ax1.imshow(Gaussian, cmap='gray', vmin=-225, vmax=255)
ax2.imshow(MAF, cmap='gray', vmin=-225, vmax=255)

fig, (ax1, ax2) = plt.subplots(1, 2)
ax1.imshow(Gaussian, cmap='gray', vmin=-225, vmax=255)
ax2.imshow(WAF, cmap='gray', vmin=-225, vmax=255)
```

Out[573]:

```
<matplotlib.image.AxesImage at 0x24923db9b48>
```




### 3. Let's Make a Blurring Filter as Function

In [531]:

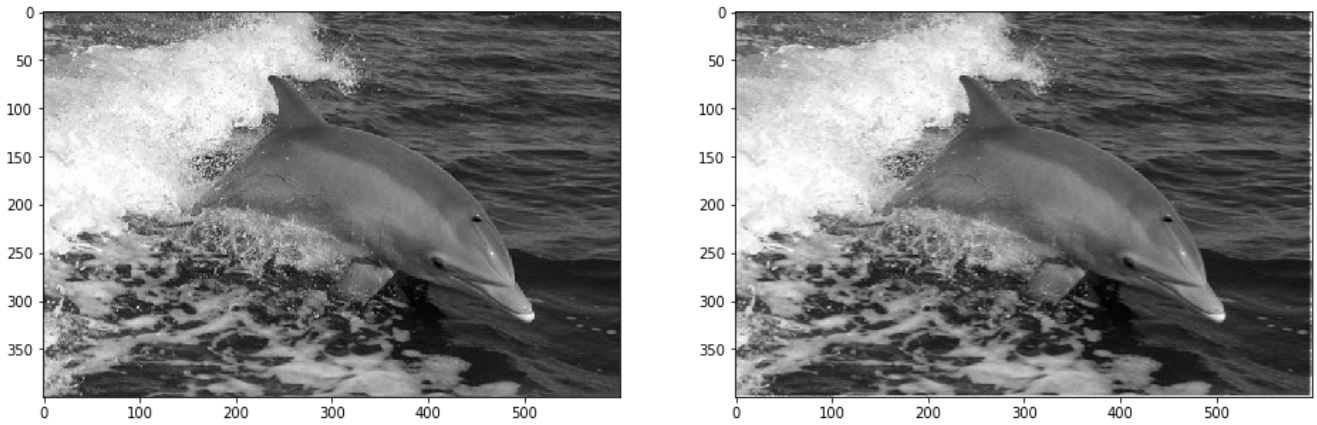
```
def Bluring(img, hight, width):
    Blurred_Image = deepcopy(img)
    for i in range(0,hight-3,1):
```

```
        for j in range(0,width-3, 1):
            Img = img[i:i+3,j:j+3]
            Filter = (1/16)* np.matrix('1 2 1; 2 4 2; 1 2 1')
            Blurred_Image[i:i+3,j:j+3] = Img * Filter
    return Blurred_Image

result = Bluring(dolphin,400,600)
```

In [535]:

```
plt.figure(figsize=(16,15))
plt.subplot(1, 2, 1)
plt.imshow(dolphin, cmap='gray', vmin=0, vmax=255)
plt.subplot(1, 2, 2)
plt.imshow(result, cmap='gray', vmin=0, vmax=63)
plt.show()
```




## 2. NumPy for Images

- Grayscale images can be represented as uint8 arrays

Grayscale images are commonly represented by 2D arrays of 8 bits unsigned integers, corresponding to values from 0 ("black") to 255 ("white"). In NumPy, this data type (dtype) is named uint8.