Introduction to Computer Vision Systems

1. Read, save, and display images with Matplotlib

Importing required libraries

In [1]:

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

Reading Colored Image

In [3]:

```
img1 = plt.imread('Archive/lena_color.tiff')

plt.imshow(img1)
#plt.title('Lena')
plt.axis('off')
plt.show()
```

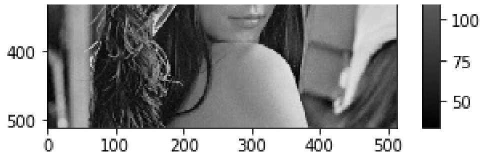
In [233]:

```
plt.imsave('lena1.tiff',img1)
```

Reading Grayscale Image

In [6]:

```
img2 = plt.imread('Archive/lena_gray.bmp')

plt.imshow(img2, cmap='gray')
plt.colorbar()
#plt.title('Lena')
#plt.axis('off')
plt.show()
```



In [235]:

```
plt.imsave('lena2.tiff',img2)
```

2. NumPy for Images

Images as NumPy arrays

- Matplotlib imread under Python returns a NumPy array
- The shape atribute keeps the array's dimensions

In the Matplotlib's Python wrapper, the imread function returns an image as a NumPy array. The array dimensions can be read from the shape attribute:

- A RGB color image can be represented by a M×N×3 uint8 array

Color images in RGB are commonly represented by 24 bits, 8 bits for each one of the three channels (red, green and blue). In NumPy, a M×N color image can be represented by a M×N×3 uint8 array.

In [236]:

```
#print(img1)
print(type(img1))
print(img1.shape)
print(img1.ndim)
print(img1.size)
print(img1.dtype)
print(img1.nbytes)
```

```
<class 'numpy.ndarray'>
(512, 512, 3)
3
786432
uint8
786432
```

- 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.

In [237]:

```
#print(img2)
print(type(img2))
print(img2.shape)
print(img2.ndim)
print(img2.size)
print(img2.dtype)
print(img2.nbytes)
```

```
<class 'numpy.ndarray'>
(512, 512)
2
262144
uint8
262144
```

- The color triplet can be retrived indexing the pixel position
- Indexing can be used to retrieve a specific channel

- Indexing can be used to retrieve a specifig channel
- Matplotlib imread returns color images in RGB order

The output of the next command shows the value at pixel (2,3) is 223, 136, 128. The image was loaded by Matplotlib using the imread procedure. Matplotlib loads color images in RGB order, so 223, 136 and 128 correspond to the values of red, green and blue respectively. The triplet is returned as a 3-d vector (a unidimensional array). Direct access to the color value can be done indexing the color dimension - img1[2,3,1] returns the green channel value, 136.

In [239]:

```
img1[2,3]
```

Out[239]:

```
array([223, 136, 128], dtype=uint8)
```

In [241]:

```
img1[2,3,1]
```

Out[241]:

```
136
```

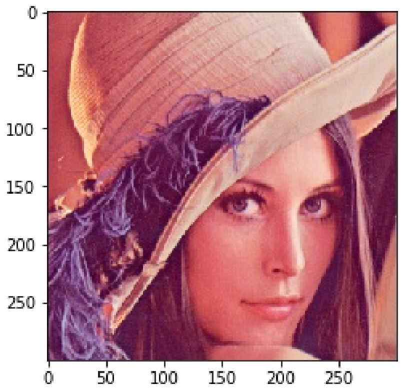
Slicing

- Slicing can retrieve parts of an array
- Employs the convention start:stop:step

As seen previously in the img1 example, array's elements can be indexed using the [] operator. Standard Python slicing can also be employed to retrieve parts of an array. Slicing employs the convention start:stop:step. For example, to retrieve the rows 3 to 9 of an bi-dimensional array A, the code A[3:10,:] is used (note stop is non-inclusive). In a similar way, to also limit the rows & columns to the range 100 to 400, A[100:400,100:400,:] is employed. Consider the user is only interested in hight from 100 to 400 and width from 100 to 400 from all channels.

In [12]:

```
plt.imshow(img1[100:400:,100:400,:])
plt.show()
```



Color Intensity

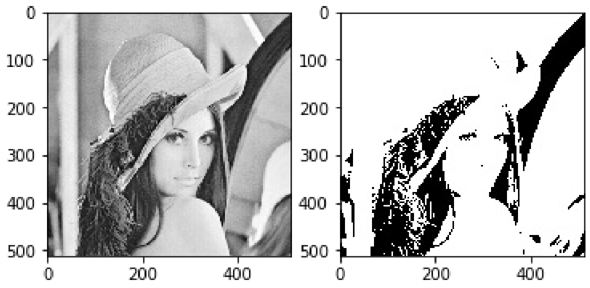
In [269]:

```
res = np.zeros_like(img1)
res[img1 > 128] = 255

plt.subplot(1,2,1)
plt.imshow(img1[:, :, 0], cmap='gray')
plt.subplot(1,2,2)
plt.imshow(res[:, :, 0], cmap='gray')
```

Out[269]:

Out[269]:

```
<matplotlib.image.AxesImage at 0x17903f59208>
```



3. Image Channels

Red Channel

In [45]:

```
img1[:, :, 0]
```

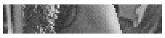
Out[45]:

```
array([[226, 226, 223, ..., 230, 221, 200],
       [226, 226, 223, ..., 230, 221, 200],
       [226, 226, 223, ..., 230, 221, 200],
       ...,
       [ 84,  84,  92, ..., 173, 172, 177],
       [ 82,  82,  96, ..., 179, 181, 185],
       [ 82,  82,  96, ..., 179, 181, 185]], dtype=uint8)
```

RGB Channels

In [52]:

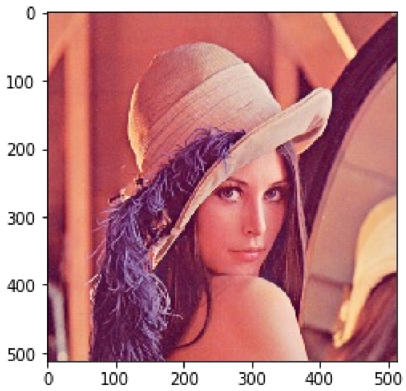
```
R = img1[:, :, 0]
G = img1[:, :, 1]
B = img1[:, :, 2]

output = [img1, R, G, B]
titles = ['Original Image', 'R Channel', 'G Channel', 'B Channel']

for i in range(4):
    plt.subplot(2, 2, i+1)
    plt.axis('off')
    plt.title(titles[i])
    if i == 0:
        plt.imshow(output[i])
    else:
        plt.imshow(output[i], cmap='gray')

plt.show()
```



Stacking them together

In [53]:

```
Output = np.dstack((R, G, B))
plt.imshow(Output)
plt.show()
```



4. Arithmetic Operations

Reading two images

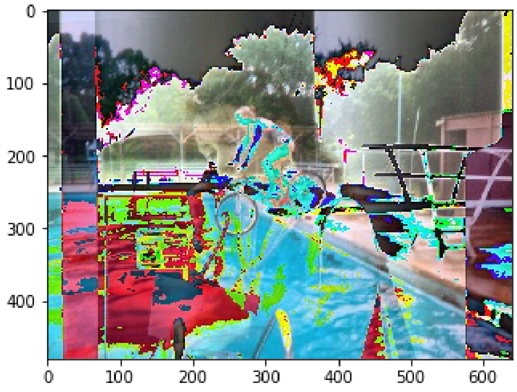
In [14]:

```
img1 = plt.imread('Archive/bmx2.tiff')
img2 = plt.imread('Archive/column.tiff')

plt.imshow(img1)
plt.show()
plt.imshow(img2)
plt.show()
```

Addition

In [110]:

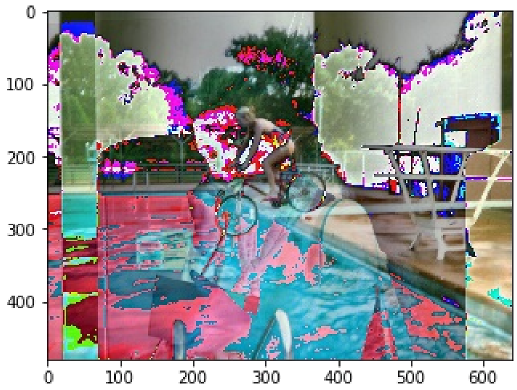
```
plt.imshow(img1 + img2)
plt.show()
```



Subtraction

In [111]:

```
plt.imshow(img1 - img2)
plt.show()
```



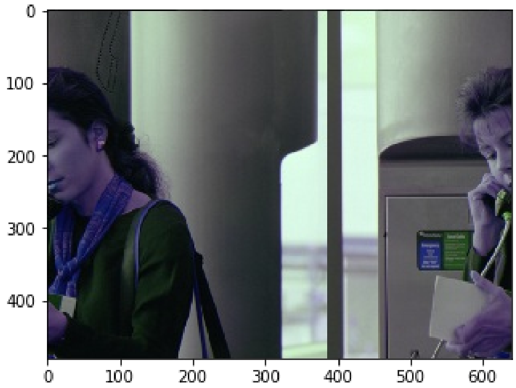
Flipping, Rolling and Rotating

In [15]:

```
plt.imshow(np.flip(img2, -3))
plt.show()
```

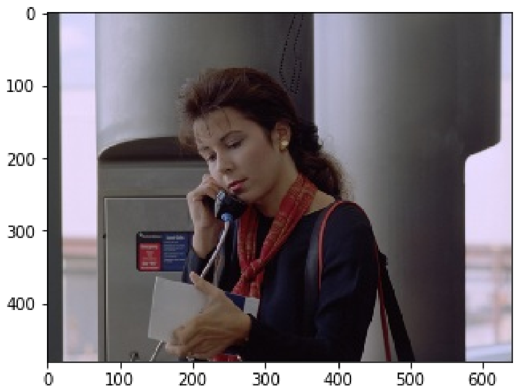
In [16]:

```
plt.imshow(np.roll(img2, 5000))
plt.show()
```



In [147]:

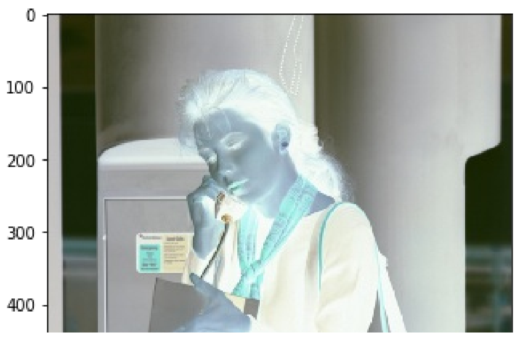
```
plt.imshow(img2)
#plt.imshow(np.fliplr(img2))
#plt.imshow(np.flipud(img2))
#plt.imshow(np.rot90(img2))
plt.show()
```



Bitwise Operations

In [22]:

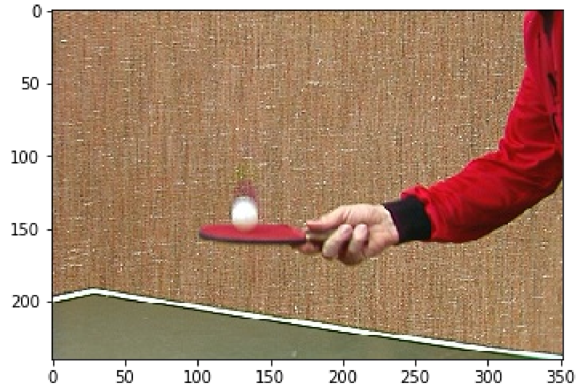
```
#lt.imshow(np.bitwise_and(img1,img2))
#plt.imshow(np.bitwise_or(img1,img2))
#plt.imshow(np.bitwise_xor(img1,img2))
plt.imshow(np.bitwise_not(img2))
plt.show()
```

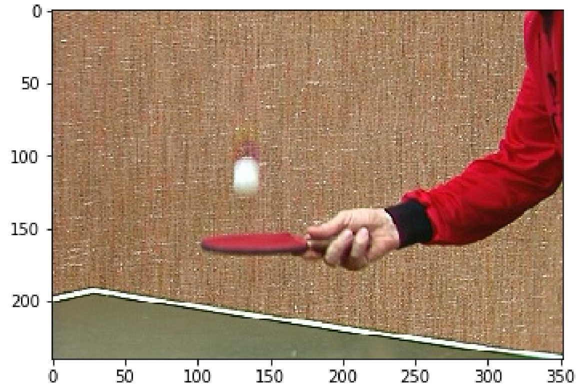


Motion Difference

In [270]:

```
img1 = plt.imread('Archive/t4.png')
img2 = plt.imread('Archive/t5.png')

plt.imshow(img1)
plt.show()
plt.imshow(img2)
plt.show()
```

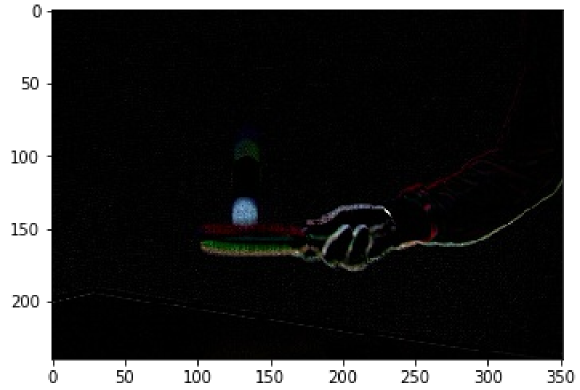




In [275]:

```
plt.imshow(img1 - img2)
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



5. Image Histogram with NumPy and Matplotlib

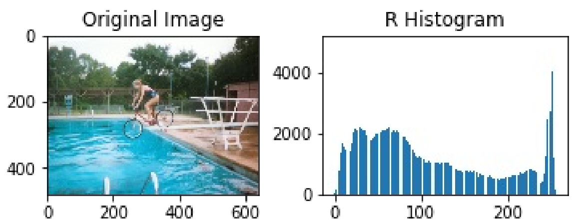
Histogram of Red Channel Intensity

In [186]:

```
R = img1[:, :, 0]
G = img1[:, :, 1]
B = img1[:, :, 2]

plt.subplot(2, 2, 1)
plt.title('Original Image')
plt.imshow(img1)

hist, bins = np.histogram(R.ravel(), bins=256, range = (0,256))
plt.subplot(2, 2, 2)
plt.title('R Histogram')
plt.bar(bins[:-1],hist)
```

Out[186]:

```
<BarContainer object of 256 artists>
```



Histogram of RGB Channels Intensity

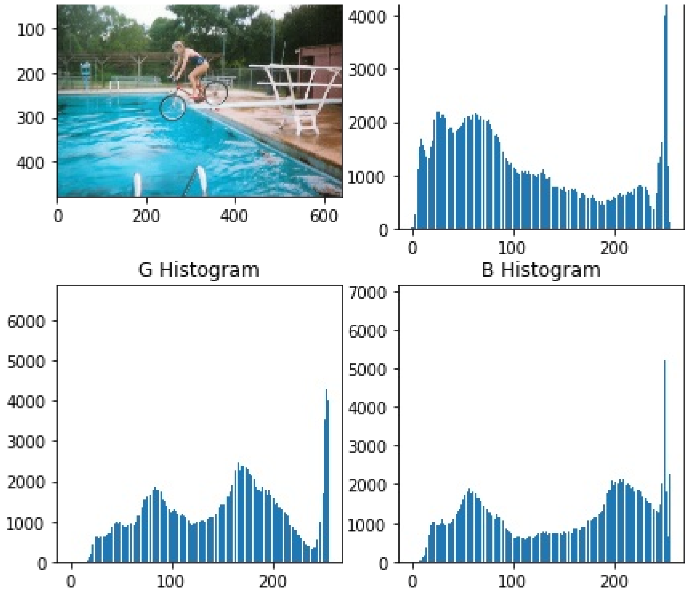
In [187]:

```
plt.subplots_adjust(hspace=0.5, wspace=0.5)

plt.figure(figsize=(7,7))
plt.subplot(2, 2, 1)
plt.title('Original Image')
plt.imshow(img1)

hist, bins = np.histogram(R.ravel(), bins=256, range = (0,256))
plt.subplot(2, 2, 2)
plt.title('R Histogram')
plt.bar(bins[:-1],hist)

hist, bins = np.histogram(G.ravel(), bins=256, range = (0,256))
plt.subplot(2, 2, 3)
plt.title('G Histogram')
plt.bar(bins[:-1],hist)

hist, bins = np.histogram(B.ravel(), bins=256, range = (0,256))
plt.subplot(2, 2, 4)
plt.title('B Histogram')
plt.bar(bins[:-1],hist)
```

Out[187]:

```
<BarContainer object of 256 artists>
```

```
<Figure size 432x288 with 0 Axes>
```


G Histogram
B Histogram
100
200
300
400
0
200
400
600
4000
3000
2000
1000
0
100
200
6000
5000
7000