

**California State University, Long Beach**  
**Department of Computer Engineering and Computer Science**  
**CECS 553 Sec 02 11792 (Machine Vision) – Fall 2022**  
**Assignment 03 – Tuesday, 09/20/2022**

## Manipulating Images

Estimated time needed: **30** minutes

### Objectives

In this lab, you will learn how to manipulate images, OpenCV image Arrays. You will learn how to copy an image to avoid aliasing. We will cover flipping images and cropping images. You will also learn to change pixel images; this will allow you to draw shapes, write text and superimpose images over other images.

- [Manipulating Images](#)
  - Copying Images
  - Flipping Images
  - Cropping an Image
  - Changing Specific Image Pixels

---

Download the images for the lab

```
In [ ]: !wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DEVELOPER/Python%20Developer%20Toolbox/Week%203/03-01-Images/03-01-Images-Data/03-01-Images-Data.zip
!wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DEVELOPER/Python%20Developer%20Toolbox/Week%203/03-01-Images/03-01-Images-Data/03-01-Images-Data.zip
!wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DEVELOPER/Python%20Developer%20Toolbox/Week%203/03-01-Images/03-01-Images-Data/03-01-Images-Data.zip
```

We will be using these imported functions in the lab

```
In [ ]: import matplotlib.pyplot as plt
import cv2
import numpy as np
```

# Copying Images

If you want to reassign an array to another variable, you should use the `copy` method. If we do not apply the method `copy()`, the variable will point to the same location in memory. Consider the following array:

```
In [ ]: baboon = cv2.imread("baboon.png")
plt.figure(figsize=(10,10))
plt.imshow(cv2.cvtColor(baboon, cv2.COLOR_BGR2RGB))
plt.show()
```

If we do not apply the method `copy()`, the new variable will point to the same location in memory:

```
In [ ]: A = baboon
```

we use the `id` function to find the object's memory address; we see it is the same as the original array.

```
In [ ]: id(A)==id(baboon)
id(A)
```

If we apply the method `copy()`, the memory address is different

```
In [ ]: B = baboon.copy()
id(B)==id(baboon)
```

When we do not apply the method `copy()`, the variable will point to the same location in memory. Consider the array `baboon`, if we set all its values to zero, then all the values in `A` will be zero. This is because `baboon` and `A` point to the same place in memory, but `B` will not be affected.

```
In [ ]: baboon[:, :, ] = 0
```

```
In [ ]: plt.figure(figsize=(10,10))
plt.subplot(121)
plt.imshow(cv2.cvtColor(baboon, cv2.COLOR_BGR2RGB))
plt.title("baboon")
plt.subplot(122)
plt.imshow(cv2.cvtColor(A, cv2.COLOR_BGR2RGB))
plt.title("array A")
plt.show()
```

We see they are the same, this is called aliasing. Aliasing happens whenever one variable's value is assigned to another variable because variables are just names that store references to values. We can also compare `baboon` and array `B` :

```
In [ ]: plt.figure(figsize=(10,10))
plt.subplot(121)
plt.imshow(cv2.cvtColor(baboon, cv2.COLOR_BGR2RGB))
plt.title("baboon")
plt.subplot(122)
plt.imshow(cv2.cvtColor(B, cv2.COLOR_BGR2RGB))
plt.title("array B")
plt.show()
```

They are different because they used the method `copy`.

## Flipping Images

Flipping images involves reordering the index of the pixels such that it changes the orientation of the image. Consider the following image:

```
In [ ]: image = cv2.imread("cat.png")
plt.figure(figsize=(10,10))
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.show()
```

We can cast it to an array and find the shape:

```
In [ ]: width, height, C = image.shape
print('width, height, C', width, height, C)
```

Let's Flip i.e rotate it vertically. First, we create an array of equal size of type `np.uint8` bit image.

```
In [ ]: array_flip = np.zeros((width, height, C), dtype=np.uint8)
```

We assign the first row of pixels of the original array to the new array's last row. We repeat the process for every row, incrementing the row number for the original array and decreasing the new array's row index assigning the pixels accordingly.

```
In [ ]: for i, row in enumerate(image):
        array_flip[width-1-i, :, :] = row
```

We plot the results

```
In [ ]: plt.figure(figsize=(5,5))
plt.imshow(cv2.cvtColor(array_flip, cv2.COLOR_BGR2RGB))
plt.show()
```

OpenCV has several ways to flip an image, we can use the `flip()` function; we have the input image array. The parameter is the `flipCode`

is the value indicating what kind of flip we would like to perform;

- `flipcode` = 0: flip vertically around the x-axis
- `flipcode` > 0: flip horizontally around y-axis positive value
- `flipcode` < 0: flip vertically and horizontally, flipping around both axes negative value

Let apply different flipcode's in a loop:

```
In [ ]: for flipcode in [0,1,-1]:
        im_flip = cv2.flip(image,flipcode )
        plt.imshow(cv2.cvtColor(im_flip,cv2.COLOR_BGR2RGB))
        plt.title("flipcode: "+str(flipcode))
        plt.show()
```

We can also use the `rotate()` function. The parameter is an integer indicating what kind of flip we would like to perform.

```
In [ ]: im_flip = cv2.rotate(image,0)
plt.imshow(cv2.cvtColor(im_flip,cv2.COLOR_BGR2RGB))
plt.show()
```

OpenCV module has built-in attributes the describe the type of flip, the values are just integers. Several are shown in the following `dict` :

```
In [ ]: flip = {"ROTATE_90_CLOCKWISE":cv2.ROTATE_90_CLOCKWISE,"ROTATE_90_COUNTERCLOCKWISE":cv2.ROTATE_90_COUNTERCLOCKWISE}
```

We see the keys are just an integer

```
In [ ]: flip["ROTATE_90_CLOCKWISE"]
```

We can plot each of the outputs using the different parameter values

```
In [ ]: for key, value in flip.items():
        plt.subplot(1,2,1)
        plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
        plt.title("original")
        plt.subplot(1,2,2)
        plt.imshow(cv2.cvtColor(cv2.rotate(image,value), cv2.COLOR_BGR2RGB))
        plt.title(key)
        plt.show()
```

## Cropping an Image

Cropping is "cutting out" the part of the image and throwing out the rest; we can crop using arrays. Let start with a vertical crop; the variable `upper` is the first row that we would like to include in the image, the variable `lower` is the last row we would like to include. We then use slicing to obtain the new image.

```
In [ ]: upper = 150
        lower = 400
        crop_top = image[upper: lower, :, :]
        plt.figure(figsize=(10,10))
        plt.imshow(cv2.cvtColor(crop_top, cv2.COLOR_BGR2RGB))
        plt.show()
```

consider the array `crop_top` we can also crop horizontally the variable `right` is the first column that we would like to include in the image, the variable `left` is the last column we would like to include in the image.

```
In [ ]: left = 150
        right = 400
        crop_horizontal = crop_top[:, left:right, :]
        plt.figure(figsize=(5,5))
        plt.imshow(cv2.cvtColor(crop_horizontal, cv2.COLOR_BGR2RGB))
        plt.show()
```

## Changing Specific Image Pixels

We can change specific image pixels using array indexing; for example, we can set all the channels in the original image we cropped to zero :

```
In [ ]: array_sq = np.copy(image)
        array_sq[upper:lower, left:right, :] = 0
```

We can compare the results to the new image.

```
In [ ]: plt.figure(figsize=(10,10))
plt.subplot(1,2,1)
plt.imshow(cv2.cvtColor(image,cv2.COLOR_BGR2RGB))
plt.title("original")
plt.subplot(1,2,2)
plt.imshow(cv2.cvtColor(array_sq,cv2.COLOR_BGR2RGB))
plt.title("Altered Image")
plt.show()
```

We can also create shapes and `OpenCV`, we can use the method `rectangle`. The parameter `pt1` is the top-left coordinate of the rectangle: `(left,top)` or `$(x\_0,y\_0)$`, `pt2` is the bottom right coordinate `(right,lower)` or `$(x\_1,y\_1)$`. The parameter `color` is a tuple representing the intensity of each channel `( blue, green, red)`. Finally, we have the line thickness.

```
In [ ]: start_point, end_point = (left, upper),(right, lower)
image_draw = np.copy(image)
cv2.rectangle(image_draw, pt1=start_point, pt2=end_point, color=(0, 255, 0),
plt.figure(figsize=(5,5))
plt.imshow(cv2.cvtColor(image_draw, cv2.COLOR_BGR2RGB))
plt.show()
```

We can overlay text on an image using the function `putText` with the following parameter values:

- `img` : Image array
- `text` : Text string to be overlaid
- `org` : Bottom-left corner of the text string in the image
- `fontFace` : type of font
- `fontScale` : Font scale
- `color` : Text color
- `thickness` : Thickness of the lines used to draw a text
- `lineType` : Line type

```
In [ ]: image_draw=cv2.putText(img=image,text='Stuff',org=(10,500),color=(255,255,255)
plt.figure(figsize=(10,10))
plt.imshow(cv2.cvtColor(image_draw,cv2.COLOR_BGR2RGB))
plt.show()
```

## Question

Use the image baboon.png from this lab or take any image you like.

Open the image and create a OpenCV Image object called `im`, convert the image from BGR format to RGB format, flip `im` vertically around the x-axis and create an image called `im_flip`, mirror `im` by flipping it horizontally around the y-axis and create an image called `im_mirror`, finally plot both images