

HƯỚNG DẪN THỰC HÀNH XỬ LÝ ẢNH Affine Transformation

Nguyễn Hải Triều

Khoa CNTT-Trường ĐH Nha Trang

`trieunh@ntu.edu.vn`

Ngày 25 tháng 11 năm 2023

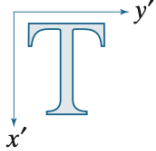
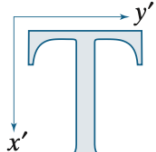
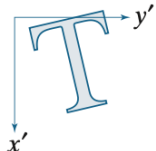
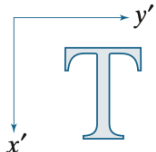
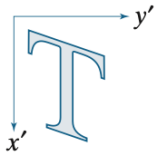
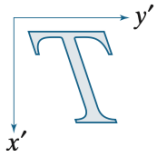
Mục lục

1	Affine transformations	2
2	Translation	3
3	Rotation	3
4	Scaling	7
5	Interpolation for Resizing in OpenCV (Scaling khi chưa biết tỉ lệ c)	8
6	Flipping	9
7	Cropping	10
8	Splitting and merging channels	11

1 Affine transformations

An affine transformation is a geometric transformation that preserves points, lines and planes. It satisfies the following conditions:

- Collinearity: Points which lie on a line before the transformation continue to lie on the line after the transformation.
- Parallelism: Parallel lines will continue to be parallel after the transformation.
- Convexity: A convex set will continue to be convex after the transformation.
- Ratios of parallel line segments: The ratio of the length of parallel line segments will continue to be the same after transformation.

Transformation Name	Affine Matrix, A	Coordinate Equations	Example
Identity	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x' = x$ $y' = y$	
Scaling/Reflection (For reflection, set one scaling factor to -1 and the other to 0)	$\begin{bmatrix} c_x & 0 & 0 \\ 0 & c_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x' = c_x x$ $y' = c_y y$	
Rotation (about the origin)	$\begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x' = x \cos \theta - y \sin \theta$ $y' = x \sin \theta + y \cos \theta$	
Translation	$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$	$x' = x + t_x$ $y' = y + t_y$	
Shear (vertical)	$\begin{bmatrix} 1 & s_v & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x' = x + s_v y$ $y' = y$	
Shear (horizontal)	$\begin{bmatrix} 1 & 0 & 0 \\ s_h & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x' = x$ $y' = s_h x + y$	

Hình 1: Affine transformations.

2 Translation

Translation is the process of shifting the image along the various axes (x-, y- and z-axis). For a **2D image**, if we consider a *pixel coordinate* $(x, y, 1)$ and perform the **dot product** with the *translation matrix*, we will obtain the pixel coordinate of the transformed matrix

$$C_{transformed} = [x \ y \ 1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} = [x + t_x \ y + t_y \ 1].$$

Task: translate an original pixel coordinates: (10, 20) of image 10 pixels to the right and 20 pixels down?

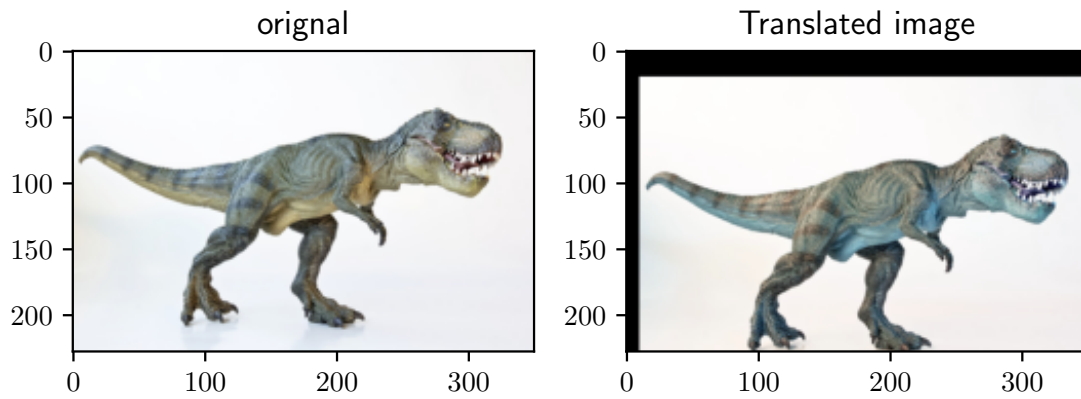
Ví dụ sử dụng OpenCV để giải Task trên.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import cv2
4 def image_translate(image,shiftX,shiftY):
5     '''
6     NOTE: Translating (shifting) an image is given by a
7           NumPy matrix in the form: [[1, 0, shiftX], [0, 1, shiftY]]
8           You simply need to specify how many pixels you want
9           to shift the image in the X and Y direction.
10          Let's translate the image 25 pixels to the right and
11          50 pixels down
12     '''
13     M = np.float32([[1, 0, shiftX], [0, 1, shiftY]])
14     return cv2.warpAffine(image,M,(image.shape[1],image.shape[0]))
15
16 img=cv2.imread('./images/trex.png')
17 fig=plt.figure()
18 ax1,ax2=fig.subplots(1,2)
19 ax1.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
20 ax1.set_title("original")
21 # shift image along x,y axis.
22 shifted = image_translate(img,10,20)
23 ax2.imshow(shifted)
24 ax2.set_title("Translated image")
25 plt.savefig("translatedi.pdf",bbox_inches='tight')
26 plt.show()
```

The missing pixel values on the left and top are given a value of 0 and hence the black pixels on the left and top edge.

3 Rotation

Rotation is the process of **changing the radial orientation of an image along the various axes with respect to a fixed point** (normaly to the pixel coordinate (0,



Hình 2: An example of applying translation on an image

0). The **transformation matrix** for a counter-clockwise *rotation around the coordinate* $(0, 0)$ of an image is defined as:

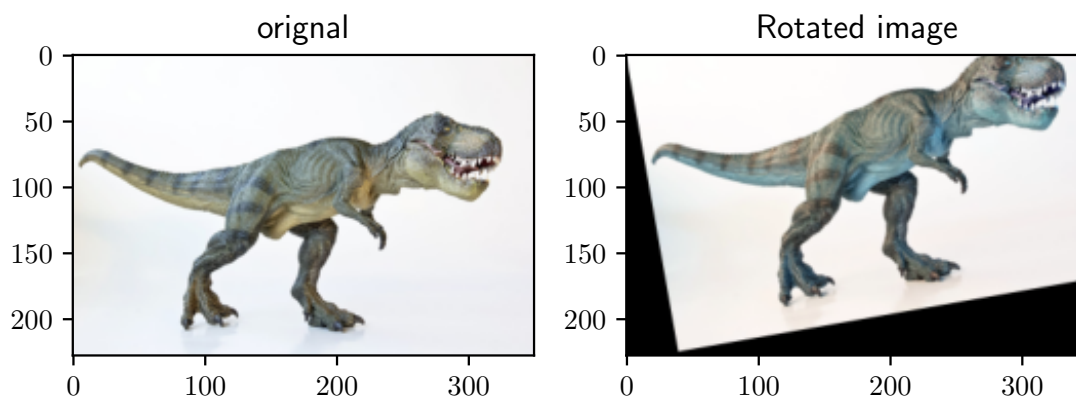
$$C_{rotated} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = [x \cos(\theta) - y \sin(\theta) \quad x \sin(\theta) + y \cos(\theta) \quad 1] \quad (1)$$

```

1  #Code minh hoạ xoay ảnh một góc 10 độ quanh pixel (0,0)
2  import numpy as np
3  import matplotlib.pyplot as plt
4  import cv2
5  import math
6  def image_rotation(image, degree):
7      theta = -degree * math.pi / 180 #chuyển độ sang radian
8      (h, w) = image.shape[:2]
9      M = np.float32([[math.cos(theta), -math.sin(theta), 0], [math.sin(theta),
10         math.cos(theta), 0]])
11      return cv2.warpAffine(image, M, (w, h))
12  img = cv2.imread('./images/trex.png')
13  fig = plt.figure()
14  ax1, ax2 = fig.subplots(1, 2)
15  ax1.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
16  ax1.set_title("original")
17  rotated = image_rotation(img, 10)
18  ax2.imshow(rotated)
19  ax2.set_title("Rotated image")
20  plt.savefig("Rotatedim.pdf", bbox_inches='tight')
  
```

How to rotate image around the center of an image?

To rotate an image around its center, we need to **translate image to the its center**, **apply the rotation matrix**, and then **translate it back to its original position**. This involves two translation matrices and the rotation matrix.



Hình 3: The transformed image is rotated by 10 degrees.

- First Translation: Translate image to the its center

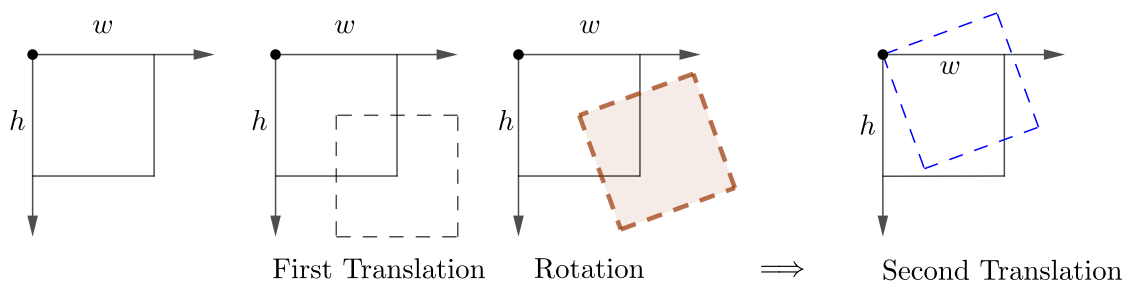
$$\begin{bmatrix} 1 & 0 & w//2 \\ 0 & 1 & h//2 \\ 0 & 0 & 1 \end{bmatrix}.$$

- Rotation: Apply the rotation matrix (1) obtained from the formula above with the desired angle.
- Second Translation: Translate the rotated image back to its origin.

$$\begin{bmatrix} 1 & 0 & -w//2 \\ 0 & 1 & -h//2 \\ 0 & 0 & 1 \end{bmatrix}.$$

- Combine: Multiply all three matrices for the final transformation matrix

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) & w(1 - \cos(\theta))//2 + h\sin(\theta)//2 \\ \sin(\theta) & \cos(\theta) & h(1 - \cos(\theta))//2 - w\sin(\theta)//2 \\ 0 & 0 & 1 \end{bmatrix} \quad (2)$$



Hình 4: Hình minh họa quá trình xoay ảnh một góc 20° tại tâm của ảnh.

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 import cv2
4 import math

```

```

5 def image_rotation(image,degree):
6     theta=-degree*math.pi/180
7     (h,w)=image.shape[:2]
8     MT1=np.float32([[1, 0, w//2], [0, 1, h//2]])
9     MR=M=np.float32([[math.cos(theta), -math.sin(theta), 0],[math.sin(theta),
        math.cos(theta), 0],[0,0,1]])
10    MT2=np.float32([[1, 0, -w//2], [0, 1, -h//2],[0,0,1]])
11    #M: final transformation matrix
12    M=MT1@MR@MT2
13    return cv2.warpAffine(image, M,(w,h))
14 img=cv2.imread('./images/trex.png')
15 fig=plt.figure()
16 ax1,ax2=fig.subplots(1,2)
17 ax1.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
18 ax1.set_title("original")
19 rotated=image_rotation(img,10)
20 ax2.imshow(rotated)
21 ax2.set_title("Rotated image")
22 plt.savefig("Rotatedim_center.pdf",bbox_inches='tight')
23 plt.show()

```

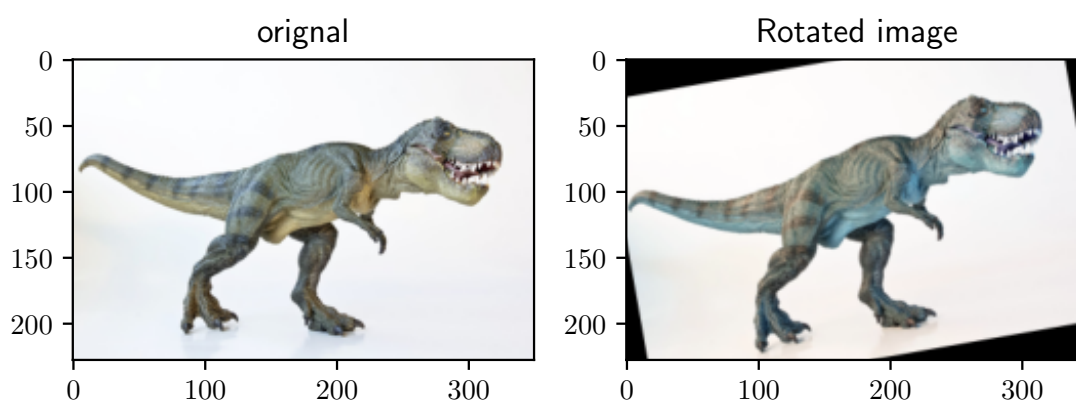
Lưu ý, trong đoạn code từ dòng 8–12, chúng ta dùng phép nhân ma trận để ra được final transformation matrix, hoặc chúng ta có thể code trực tiếp kết quả như công thức tính tay (2):

```

1 M=np.float32([[math.cos(theta), -math.sin(theta),
    w*(1-math.cos(theta))/2 + h*math.sin(theta)/2],[math.sin(theta),
    math.cos(theta), h*(1-math.cos(theta))/2-w*math.sin(theta)/2]])

```

Tuy nhiên điều này dễ gây sai sót và không thuận tiện khi thay đổi điểm xoay.

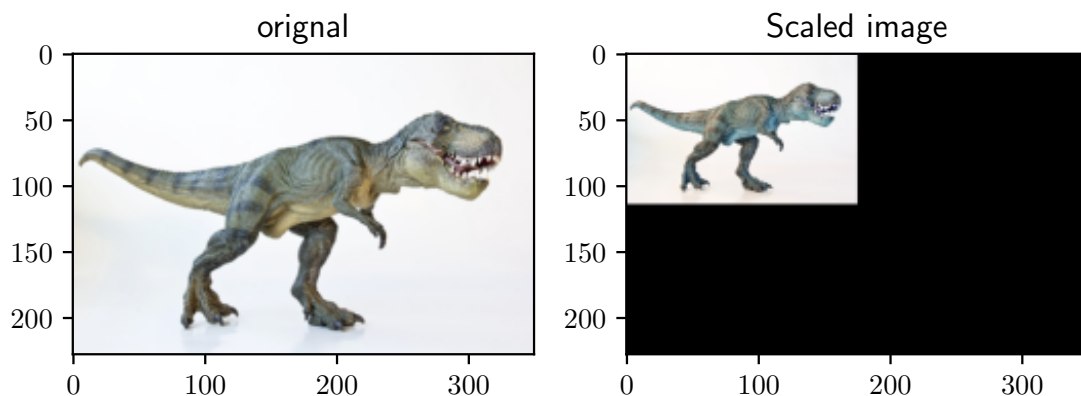


Hình 5: The transformed image is rotated by 10 degrees around it's center

4 Scaling

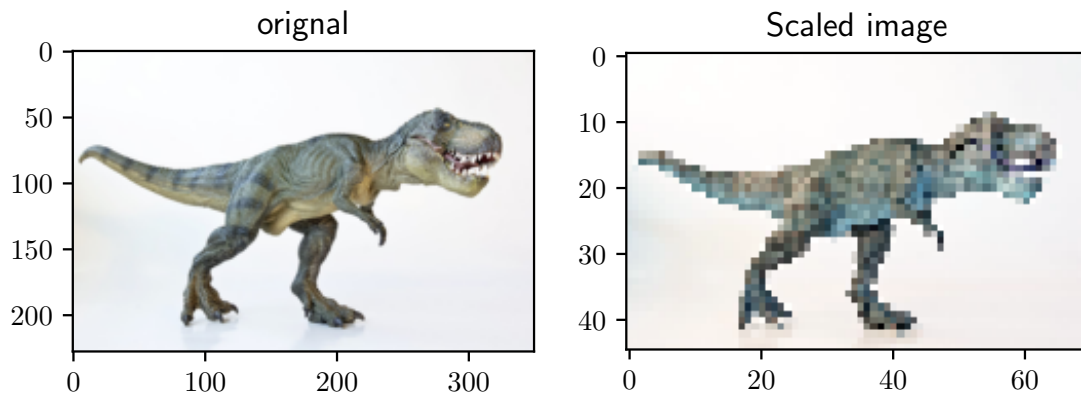
Scaling is a process of changing the distance (compression or elongation) between points in one or more axes. The scaling factor c may be different across different axes (see figure 1).

```
1  #code minh hoạ scaled ảnh
2  import numpy as np
3  import matplotlib.pyplot as plt
4  import cv2
5  def image_scaling(image,scaleX=1,scaleY=1):
6      (h,w)=image.shape[:2]
7      M=np.float32([[scaleX,0,0],[0,scaleY,0]])
8      return cv2.warpAffine(image, M,(w,h))
9
10 if __name__=='__main__':
11     img=cv2.imread('./images/trex.png')
12     fig=plt.figure()
13     ax1,ax2=fig.subplots(1,2)
14     ax1.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
15     ax1.set_title("original")
16     scaled=image_scaling(img, 0.5,0.5)
17     ax2.imshow(scaled)
18     ax2.set_title("Scaled image")
19     plt.savefig("Scale_image.pdf",bbox_inches='tight')
20     plt.show()
```



Hình 6: The image is scaled to 0.5 of its original size with reference to the input image along both axes

Bài tập: cập nhật kích thước mới cho ảnh sau scaled (ảnh mới sẽ không còn chứa các pixels đen). Kết quả được như hình 7



Hình 7: The image is scaled to 0.2 of its original size with reference to the input image along both axes

5 Interpolation for Resizing in OpenCV (Scaling khi chưa biết tỉ lệ c)

Consider an image of size 2×2 . If this image is scaled to four times its size in all linear dimensions, the new image will be of size 8×8 . **The original image has only 4 pixel values while the new image needs 64 pixel values.** The question is: **How can we fill 64 pixels with values given that there are only 4 pixel values?** The answer is **interpolation**. The various interpolation schemes available in OpenCV are:

- `cv2.INTER_LINEAR`
- `cv2.INTER_AREA`
- `cv2.INTER_CUBIC`
- `cv2.INTER_NEAREST`

When resizing an image, we need to keep in mind **the aspect ratio of the image**. If we have the new image width, we compute the ratio of the new height to the old height, we simply define our **ratio r to be the new width divided by the old width** (Trường hợp resize theo độ dài width).

```

1 # code minh hoạ chung cho resize ảnh theo độ dài width hoặc height
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import cv2
5 def image_resized(image, width=None, height=None, inter=cv2.INTER_AREA):
6     dim=None
7     (h,w)=image.shape[:2]
8     if width is None and height is None:
9         return image
10    if height is None:
11        '''we only resized the image by specifying the width.'''
12        r = width / float(w)
13        '''From the image width, we compute the ratio of the new height to the
            old height, we simply define our ratio r to be the new width divided
            by the old width'''

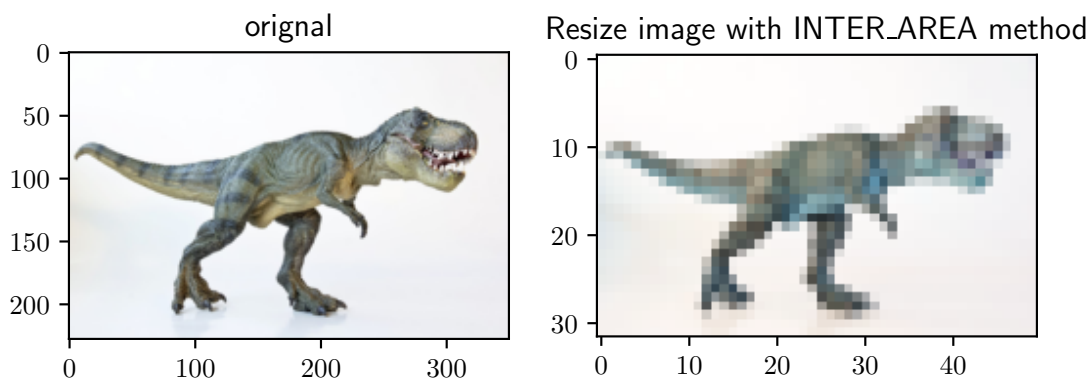
```



```

14     dim=(width,int(h*r))
15 else:
16     '''we only resized the image by specifying the height.'''
17     r = height / float(h)
18     dim=(int(w*r),height)
19     resize=cv2.resize(image,dim,interpolation=inter)
20     return resize
21 if __name__=='__main__':
22     img=cv2.imread('./images/trex.png')
23     fig=plt.figure()
24     ax1,ax2=fig.subplots(1,2)
25     ax1.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
26     ax1.set_title("original")
27     resized=image_resized(img,width=50)
28     ax2.imshow(resized)
29     ax2.set_title("Resize image with INTER_AREA method")
30     plt.savefig("Resize_image.pdf",bbox_inches='tight')
31     plt.show()

```



Hình 8: The T-Rex resized to have a width of 50 pixels.

Bài tập: nếu resize tăng width, height lớn hơn kích thước thực tế của ảnh thì ảnh có tốt hơn không?

6 Flipping

We can flip an image around either the x or y axis, or even both. An image can be flipped horizontally (lật ảnh xoay theo trục y : flipCode=1), vertically (lật ảnh xoay theo trục x flipCode=0), and both horizontally and vertically at the same time (flipCode=-1).

```

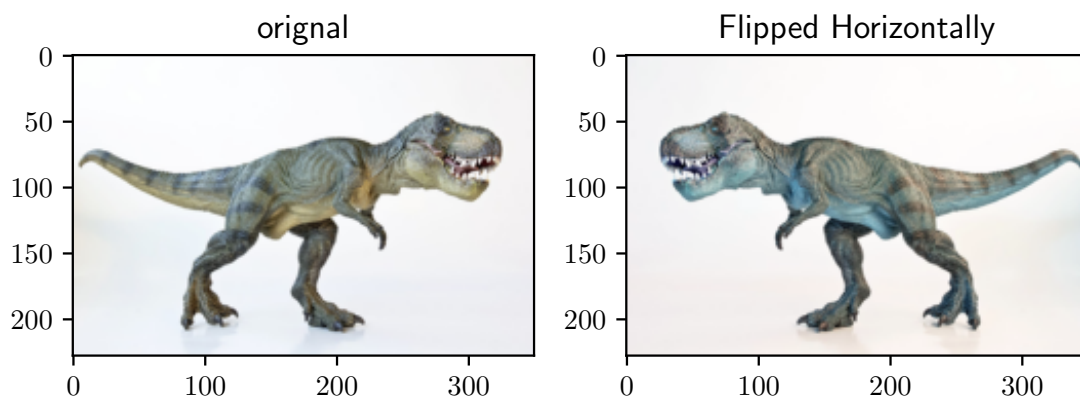
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import cv2
4 plt.rcParams.update({"text.usetex":True})
5 def image_flipping(image,flip_mode=0):
6     return cv2.flip(image,flipCode=flip_mode)

```

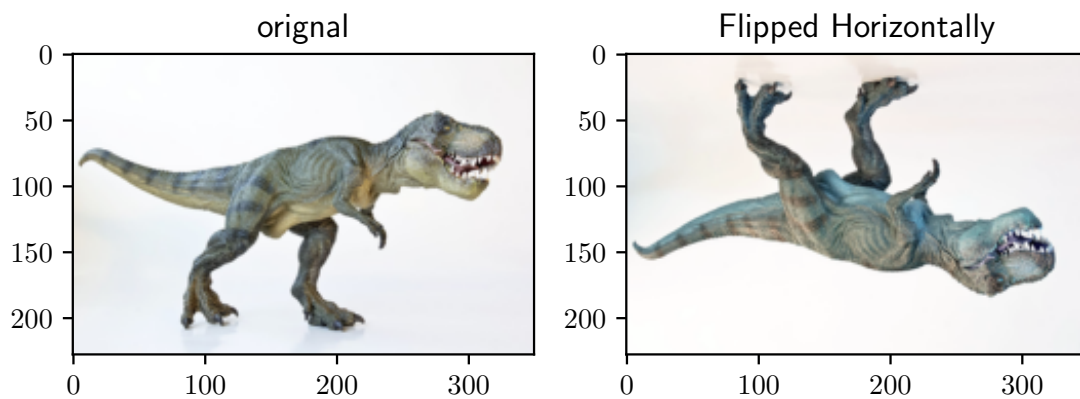
```

7  if __name__=='__main__':
8      img=cv2.imread('./images/trex.png')
9      fig=plt.figure()
10     ax1,ax2=fig.subplots(1,2)
11     ax1.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
12     ax1.set_title("original")
13     flipped=image_flipping(img,flip_mode=1)
14     ax2.imshow(flipped)
15     ax2.set_title("Flipping image")
16     plt.savefig("Fliped_image.pdf",bbox_inches='tight')
17     plt.show()

```



Hình 9: Flipping the T-Rex image horizontally.

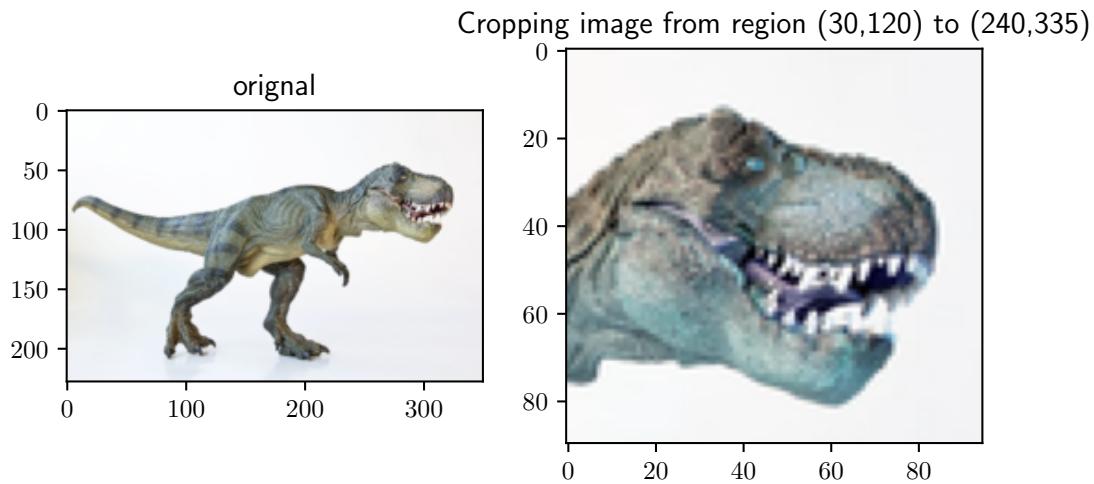


Hình 10: Flipping the T-Rex vertically.

7 Cropping

Cropping an image involves removing the outer parts of the image that are not of interest. We can accomplish image cropping by **using NumPy array slicing**. In order to perform our cropping, NumPy expects four indexes:

1. *Start y* (*shape[0]*): The starting y coordinate.



Hình 11: Image cropping.

2. *End y* (*shape[0]*): The ending y coordinate.
3. *Start x* (*shape[1]*): The starting x coordinate of the slice
4. *End x* (*shape[1]*): The ending x-axis coordinate of the slice

Dựa vào hướng dẫn sử dụng Numpy slicing, SV hãy code một hàm crop vùng đầu của ảnh T-Rex và kết quả thu được như Hình 11

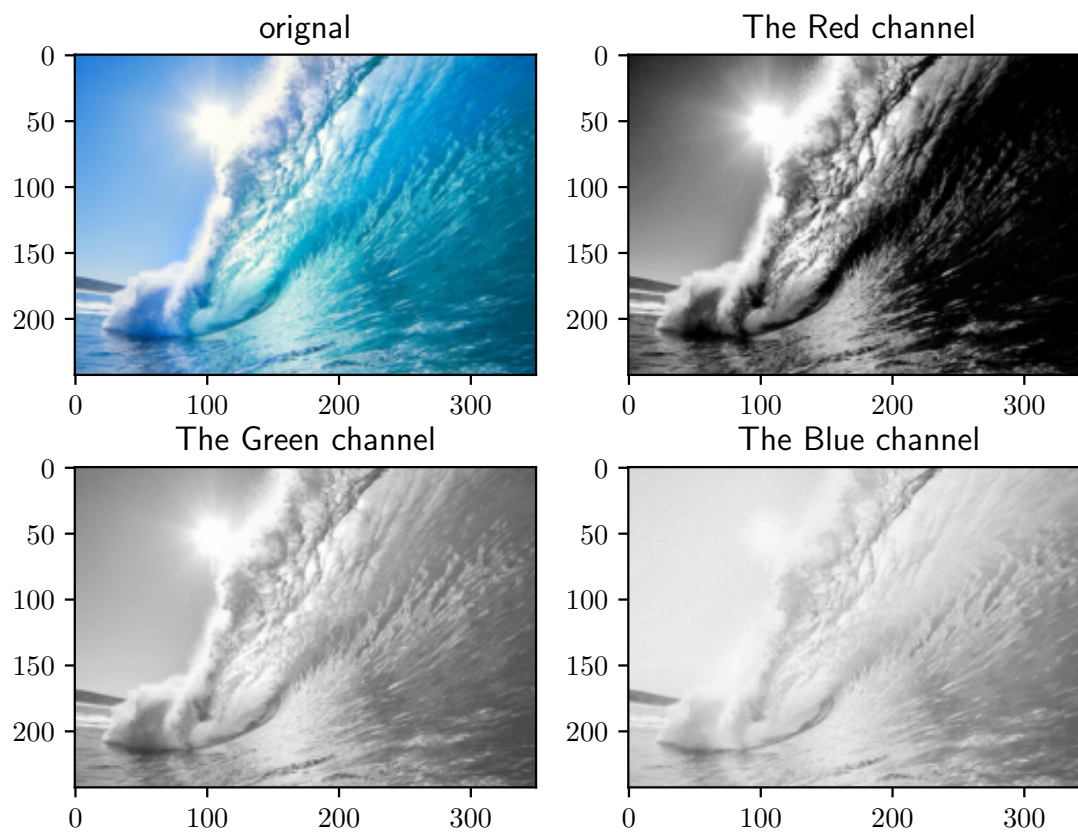
8 Splitting and merging channels

Lưu ý, OpenCV lưu trữ kênh màu theo định dạng BGR.

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 import cv2
4 def image_splitting(image):
5     return cv2.split(image)
6
7 if __name__=='__main__':
8     img=cv2.imread('./images/wave.png')
9     fig=plt.figure()
10    ax=fig.subplots(2,2)
11    ax[0,0].imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
12    ax[0,0].set_title("original")
13    B,G,R=image_splitting(img)
14    ax[0,1].imshow(cv2.cvtColor(R,cv2.COLOR_BGR2RGB))
15    ax[0,1].set_title("The Red channel")
16    ax[1,0].imshow(cv2.cvtColor(G,cv2.COLOR_BGR2RGB))
17    ax[1,0].set_title("The Green channel")
18    ax[1,1].imshow(cv2.cvtColor(B,cv2.COLOR_BGR2RGB))
19    ax[1,1].set_title("The Blue channel")
20    plt.savefig("splitting_image.pdf",bbox_inches='tight')

```



Hình 12: The three RGB channels of our wave image.

Tài liệu

- [1] Ravishankar Chityala, Sridevi Pudipeddi. Image Processing and Acquisition using Python (2020), Second Edition, Chapman & Hall/CRC.
- [2] Rafael C. Gonzalez, Richard E. Woods. Digital Image Processing (2018), Fourth Edition, Global Edition, Pearson.
- [3] Adrian Rosebrock. Practical Python and OpenCV: An Introductory, Example Driven Guide to Image Processing and Computer Vision, Third Edition.