VIET NAM NATIONAL UNIVERSITY,HO CHI MINH CITY
UNIVERSITY OF SCIENCE
FACULTY OF: INFORMATIONTECHNOLOGY

Báo cáo đồ án 2

Toán ứng dụng và thống kê cho Công nghệ Thông tin

Họ và tên sinh viên: Lại Minh Thông
Mã số sinh viên: 20127635

Giáo viên hướng dẫn:

Thầy Nguyễn Văn Quang Huy

Thầy Vũ Quốc Hoàng

Cô Phan Thị Phương Uyên

Thầy Lê Thanh Tùng

Thành phố Hồ Chí Minh – 2022

Đồ án 2 – Image Processing

I. Thông tin cá nhân.

Họ tên người làm đồ án: Lại Minh Thông

Mã số sinh viên: 20127635

II. Các vấn đề đã hoàn thành.

STT	Tên chức năng	Mức độ hoàn thành
1	Thay đổi độ sáng cho ảnh	100%
2	Thay đổi độ tương phản	100%
3	Lật ảnh (ngang – dọc)	100%
4	Chuyển đổi ảnh RGB thành ảnh xám	100%
5	Chồng 2 ảnh cùng kích thước	100%
6	Làm mờ ảnh	100%
7	Gắn khung tròn cho hình ảnh	100%
8	Gắn khung 2 ellipses xoay chéo	100%

III. Cài đặt chi tiết

Ngôn ngữ lập trình được sử dụng để giải quyết bài toán của đồ án là Python và trên môi trường Jupyter Notebook.

Mã nguồn: https://github.com/ThongLai/ImageProcessing

Phần lớn công thức tính toán các chức năng biến đổi hình ảnh được kham khảo từ Code Project - *Image Processing using C#*. Các đường dẫn đến website, tài liệu tham khảo được đặt ở phần *Tài liệu tham khảo*

Trong quá trình cài đặt có một số hàm chức năng sẽ ép mảng hình ảnh dạng *uint8* về dạng số nguyên hoặc số thực. Lý do là vì kiểu *unit8* chỉ có thể chạy trong khoảng 0-255 nên khi tính toán sẽ có trường hợp làm cho vượt quá miền giá. Vì thế sẽ ép về dạng *int* hay *float* trước khi tính toán, sau đó xử lý các giá trị vượt khỏi miền giá trị này và return lại kiểu *uint8*.

Các thư viện cần thiết:

+ numpy – Tính toán các phép toán ma trận.

+ PIL.Image – Các thủ thuật trên hình ảnh như đọc, lưu.

+ matplotlib.pyplot – Hiển thị hình ảnh.

+ math – Sử dụng cho các phép tính toán học sin, cos, đổi radian(chức năng 7, 8)

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import math
```

1 - Khai báo thư viện

Các hàm:

1) *brightenImage*.

Hàm thay đổi độ sáng hình ảnh.

1 - brightenImage

- Ý tưởng: Cộng từng giá trị màu R,B,G với 1 giá trị xác định.
- Tham số đầu vào: mảng vector pixels của ảnh có kiểu *np.array*, giá trị *scale* (mặc định là 30%),
- Kết quả trả về: mảng vector pixels của ảnh có kiểu *np.array* đã làm sáng.
- Mô tả:
  - Cộng với 1 số nguyên theo tỉ lệ *scale*.

```
image = image.astype(int)
image += int(scale*255)
```

➢ Đưa tất cả các giá trị màu vượt quá 255 về lại 255.

```
image[image > 255] = 255
```

2) *contrastImage.*

Hàm thay đổi độ tương phản hình ảnh.

2 - contrastImage

❖ Ý tưởng: Đổi miền giá trị màu từ *0->255* về *-0.5->0.5*, sau đó nhân với 1 giá trị xác định. Tham khảo [1](*4. Contrast*).

❖ Tham số đầu vào: mảng vector pixels của ảnh có kiểu *np.array*, giá trị *scale* (mặc định là 30%),

❖ Kết quả trả về: mảng vector pixels của ảnh có kiểu *np.array* đã làm tương phản.

❖ Mô tả:

➢ Đổi miền giá trị màu từ *0->255* về *-0.5->0.5*. [1]

```
image = image/255 - 0.5
```

➢ Cộng với 1 số nguyên theo tỉ lệ *scale*. [1]

```
image *= float(1 + scale)**2
```

➢ Đổi miền giá trị màu về lại ban đầu. [1]

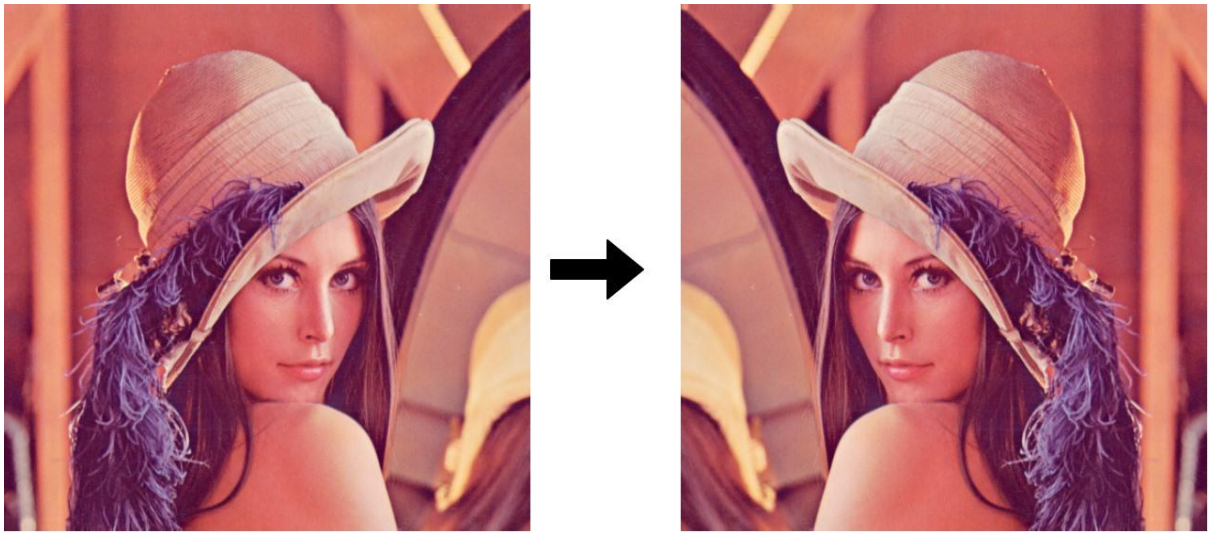
```
image = (image + 0.5) * 255
```

➢ Đưa tất cả các giá trị màu vượt quá *0->255* về lại 0, 255.

```
image[image > 255] = 255
image[image < 0] = 0
```

3) *flipImage.*

Hàm lật hình ảnh.



3 - flipImage

❖ Ý tưởng: Đưa không gian mảng hình về lại ban đầu và sử dụng hàm *np.flip* của *numpy* để lật ma trận. Tham khảo [2] .

❖ Tham số đầu vào: mảng vector pixels của ảnh có kiểu *np.array*, giá trị không gian gốc *dim*, chế độ lật (ngang/dọc) *mode* với *mode = 0* lật dọc, *mode = 1* lật ngang.

❖ Kết quả trả về: mảng vector pixels của ảnh có kiểu *np.array* đã được lật ngang/dọc.

❖ Mô tả:

➢ Sử dụng *reshape* để đưa không gian mảng hình về lại ban đầu.

```
image = image.reshape(dim)
```

➢ Sử dụng hàm *np.flip* của *numpy* để lật ma trận với trục tương ứng với giá trị *mode*(0 là ngang, 1 là dọc) [2]

```
image = np.flip(image, mode)
```

➢ Đưa không gian mảng hình về dạng mảng các vector pixels.

```
image = image.flatten()
```

4) *greyImage.*

Hàm làm xám hình ảnh.

4 - greyImage

- Ý tưởng: Chuyển 3 giá trị R, B, G về 1 giá trị. Tham khảo [1] (*5.Greyscale*), [3] .
- Tham số đầu vào: mảng vector pixels của ảnh có kiểu *np.array*, tham số *alpha, beta, gamma* tương ứng với độ lớn phần giá trị của R, B, G với *alphe + beta + gamma = 1*.
- Kết quả trả về: mảng vector pixels của ảnh có kiểu *np.array* đã được làm xám.
- Mô tả:
  - Chuyển 3 giá trị R, B, G về 1 giá trị với công thức nhân tuyến tính như sau:

$$Grey = (alpha*R + beta*B + gamma*G)$$ [1]

Với giá trị mặc định [alpha, beta, gamma] = [0.299, 0.587, 0.114]

Thực hiện bằng cách nhân ma trận chứa các vector pixel với vector có giá trị [alpha, beta, gamma].

```
image = np.dot(image, np.array([alpha, beta, gamma]))
```

  - Sau khi nhân với vector, các vector pixel sẽ có đúng 1 giá trị. Vì vậy để hàm *plt.imshow* của *pyplot* có thể đọc được hình cần

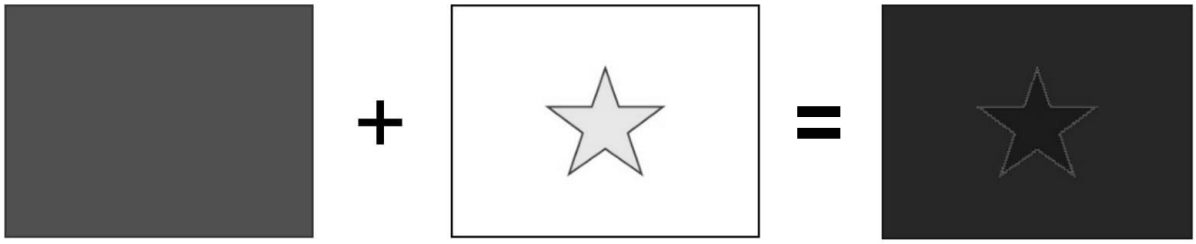
nhân bản thành 3 giá trị để tạo thành các vector pixels.

Thực hiện bằng cách nhân với vector dòng có 3 giá trị, sau đó chuyển vị ma trận kết quả. [3]

```
image = (image*np.ones((3,1))).transpose()
```

5) *overlapImage.*

Hàm chồng 2 hình ảnh.



5 - overlapImage

- Ý tưởng: Ảnh kết quả sẽ có pixel tại 1 điểm bằng trung bình cộng của 2 pixel của 2 ảnh được chồng lên nhau.
- Tham số đầu vào: 2 mảng vector pixels của 2 ảnh được chồng có kiểu *np.array*, tham số *alpha, beta* lần lượt tương ứng với mức độ phủ của ảnh 1, 2.
- Kết quả trả về: mảng vector pixels của ảnh có kiểu *np.array* đã được chồng từ 2 ảnh lên nhau.
- Mô tả:
  - Nhân ma trận vector pixels ảnh 1, ảnh 2 với tham số phần trăm chiếm *alpha*, *beta* đã cho. Sau đó lấy trung bình cộng.

    ```
    image = (image1*alpha + image2*beta)/(alpha + beta)
    ```

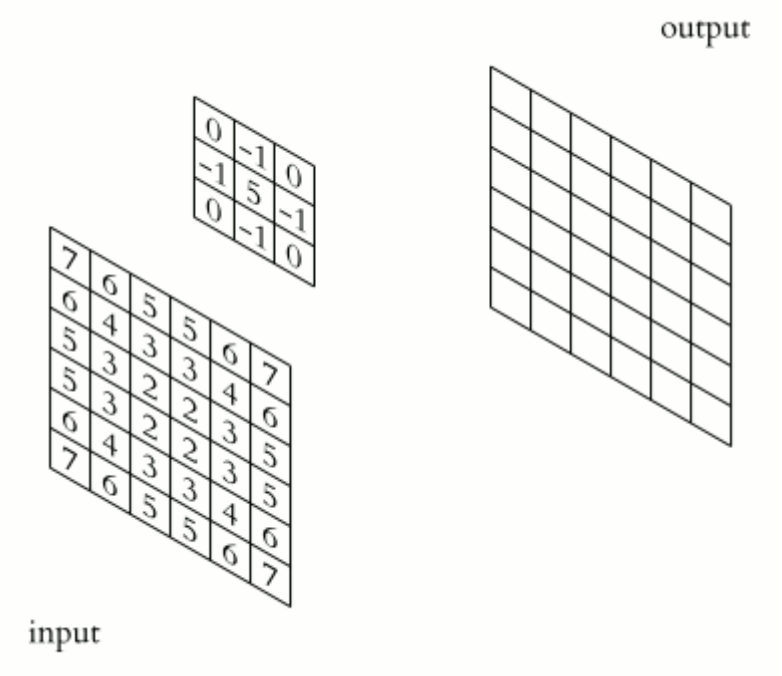
6) *blurImage.*

Hàm làm mờ hình ảnh.

6 - blurImage

❖ <u>Ý tưởng:</u> Sử dụng phép tích chập (Convolution) với 1 ma trận kernel để tạo hiệu ứng mờ ảnh. Tham khảo [4] [5] [6] [7] [8] [9] .

Phép tích chập giữa 1 ma trận với ma trận kernel vuông sẽ cho ra ma trận kết quả có thể cho thấy được sự biến dạng của ma trận bởi kernel tương ứng. [5] [9]



2- Trực quan phép tích chập

Phép tích chập tại 1 điểm trung tâm được thực hiện bằng cách tính tổng của tích từng giá trị của kernel với vùng ma trận con có không gian bằng với không gian của kernel. Trước khi thực hiện phép toán, ta cần lật kernel theo chiều ngang và dọc.

Hình trên mô tả phép tích chập với 1 kernel 3x3 [6] , kernel này được dùng cho thao tác làm nét ảnh(*Sharpen*). Ở đồ án này để làm mờ ảnh, ta sử dụng các kernel để làm mờ ảnh như Box blur, Gaussian Blur 3x3, Gaussian Blur 5x5 [5] .

Các giá trị của ma trận kết quả là kết quả quả phép tích chập tại vùng ma trận con có trung tâm là từng giá trị trong ma trận.
Ở đồ án này, mỗi giá trị trong ma trận ảnh là 1 pixel có 3 giá trị R, B, G.

Numpy có cung cấp hàm thực hiện tích chập: *np.convolve* [7] . Tuy nhiên chỉ thao tác được trên mảng và kernel 1 chiều. Vì vậy tham khảo ở *Stackoverflow* - [4] (*reply của Filipe Alves*), ta có kế hoạch để thực hiện tích chập khác như sau:
Thay vì duyệt từng giá trị và thực hiện phép chập ở từng vị trí trong mảng hình, ta dịch ma trận theo dòng và cột sau đó nhân với từng giá trị của kernel, sau đó lấy tổng các ma trận vừa tính, ta sẽ được ma trận kết quả sau khi thực hiện phép tích chập.

Ví dụ:

$$A = \begin{bmatrix} a_0 & b_0 & c_0 \\ a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{bmatrix} \quad kernel = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$$Result = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Lấy ma trận A gốc dịch 1 đơn vị theo dòng hướng lên, 1 đơn vị theo cột hướng phải, sau đó nhân với giá trị của kernel tại vị trí (0, 0):

$$\mathrm{C} = \begin{bmatrix} b_1 & c_1 & a_1 \\ b_2 & c_2 & a_2 \\ b_0 & c_0 & a_0 \end{bmatrix} * kernel[0][0] = \begin{bmatrix} 1 * b_1 & 1 * c_1 & 1 * a_1 \\ 1 * b_2 & 1 * c_2 & 1 * a_2 \\ 1 * b_0 & 1 * c_0 & 1 * a_0 \end{bmatrix}$$

$$Result = Result + C = \begin{bmatrix} 1 * b_1 & 1 * c_1 & 1 * a_1 \\ 1 * b_2 & 1 * c_2 & 1 * a_2 \\ 1 * b_0 & 1 * c_0 & 1 * a_0 \end{bmatrix}$$

Tiếp tục, lấy ma trận A gốc dịch 1 đơn vị theo dòng hướng lên, 0 đơn vị theo cột hướng phải, sau đó nhân với giá trị của kernel tại vị trí (0, 1):

$$C = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_0 & b_0 & c_0 \end{bmatrix} * kernel[0][1] = \begin{bmatrix} 2*a_1 & 2*b_1 & 2*c_1 \\ 2*a_2 & 2*b_2 & 2*c_2 \\ 2*a_0 & 2*b_0 & 2*c_0 \end{bmatrix}$$

$$Result = Result + C$$
$$= \begin{bmatrix} 1*b_1 + 2*a_1 & 1*c_1 + 2*b_1 & 1*a_1 + 2*c_1 \\ 1*b_2 + 2*a_2 & 1*c_2 + 2*b_2 & 1*a_2 + 2*c_2 \\ 1*b_0 + 2*a_0 & 1*c_0 + 2*b_0 & 1*a_0 + 2*c_0 \end{bmatrix}$$

Lặp lại các phép tính cho tới khi duyệt qua tất cả các phần tử của kernel. Ma trận Result sau cùng chứa các kết quả của phép tích chập tại vị trí tương ứng.

Kiểm tra:
Xét vị trí trung tâm của ma trận kết quả:

$$Result[1][1] = 1*c_2 + 2*b_2 + 1*a_2 + 2*c_1 + 4*b_1 + 2*a_1 + 1*c_0 + 2*b_0 + 1*a_0$$

Đây chính là kết quả của phép tích chập của ma trận A với kernel 3x3 tại vị trí A[1][1].

❖ Tham số đầu vào: mảng vector pixels của ảnh có kiểu *np.array*, giá trị không gian gốc *dim*, ma trận vuông *kernel*.
❖ Kết quả trả về: mảng vector pixels của ảnh có kiểu *np.array* đã được làm mờ.
❖ Mô tả:

Mã nguồn được tham khảo trong *Stackoverflow* - [4] (reply của *Filipe Alves)*

- Sử dụng *reshape* để đưa không gian mảng hình về lại ban đầu.

```
image = image.reshape(dim)
```

- Khởi tạo mảng hình ảnh kết quả *image_result*.

```
image_result = np.zeros(image.shape)
```

- Phép tích chập của ma trận hình với kernel được thực hiện như sau [4]:
  - Duyệt từng phần tử trong kernel theo dòng(*i*) và cột(*j*).
  - Sử dụng hàm *np.roll* của *numpy* để dịch chuyển dòng hoặc cột tương ứng theo phần tử hiện tại của kernel. [8]
  - Nhân giá trị hiện tại của kernel với mảng đã được dịch.
  - Cộng dồn của mảng tích kết quả vào *image_result*.

  Sau khi duyệt qua tất cả các phần tử của kernel, image_result chứa các kết quả của phép tích chập tại vị trí tương ứng.

```
kernelDim = kernel.shape[0]
for i in range(kernelDim):
    for j in range(kernelDim):
        rowShiftValue = int(i - kernelDim/2)
        colShiftValue = int(j - kernelDim/2)
        shiftedArray = np.roll(image, (rowShiftValue, colShiftValue), axis=(0, 1))
        image_result += shiftedArray * kernel[i,j]
```

- Đưa không gian mảng hình kết quả về dạng mảng các vector pixels.

```
image_result = image_result.flatten()
```

7) *circleFrame*.

Hàm gắn khung tròn cho hình ảnh.

❖ Ý tưởng: Vẽ ma trận hình tròn có tâm ở giữa khung hình, sau đó phủ lên ảnh gốc (*mask*). Tham khảo [10][11][12]

❖ Tham số đầu vào: mảng vector pixels của ảnh có kiểu *np.array*, giá trị không gian gốc *dim*, chế độ khung *mode* với *mode = 0* đường kính đường tròn bằng chiều dài không gian ngắn nhất của khung hình, *mode = 1* đường kính đường tròn bằng chiều dài không gian dài nhất của khung hình.

❖ Kết quả trả về: mảng vector pixels của ảnh có kiểu *np.array* đã gắn khung tròn.

❖ Mô tả:

➢ Khởi tạo tọa độ tâm của đường tròn trùng với vị trí trung tâm của hình và bán kính khung tròn phụ thuộc vào giá trị *mode*

```
center = np.array([height/2, width/2])
radius = mode*max(height/2, width/2) + (1 - mode)*min(height/2, width/2)
```

➢ Sử dụng hàm *np.ogrid* [11] của *numpy* để tạo giá trị tọa độ *x, y* tương ứng với chiều dài và rộng của khung hình. [10]

```
x, y = np.ogrid[:height, :width]
```

➢ Sau đó tạo mảng *mask_circle* với kiểu boolean chứa kết quả chân trị của từng cặp *x, y* trong biểu thức bất phương trình nằm ngoài hình tròn [12]:

$$(x-a)^2 + (y-b)^2 > R^2$$

$$R \text{ là } bán\ kính \text{ đường } tròn\ tâm\ I(a,b)$$

```
mask_circle = (x - center[0])**2 + (y - center[1])**2 > radius**2
```

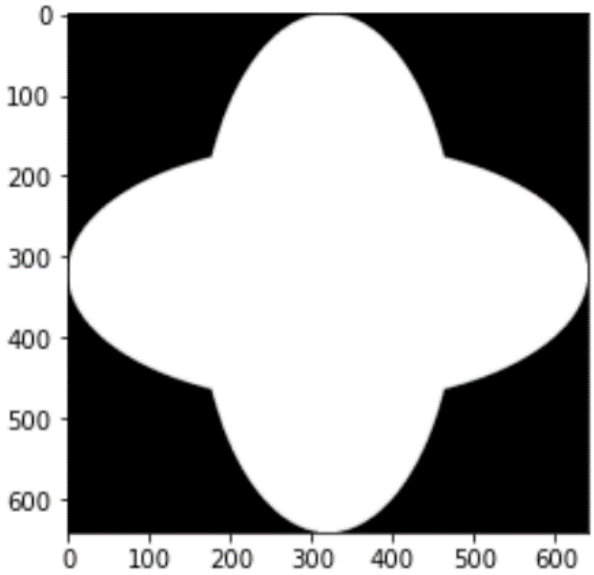
➢ Xóa giá trị màu tại các vị trí mà kết quả chân trị của *mask_circle* là *True* bằng cách gán với vector [0, 0, 0]

```
image[mask_circle] = np.zeros(dim[2])
```

8) *interlaceEllipseFrame.*

Hàm gắn khung 2 ellipses xoay chéo chồng lên nhau cho hình ảnh.

❖ Ý tưởng: Vẽ ma trận hình ellipse có tâm ở giữa khung hình thẳng đứng và ngang chồng lên nhau với kích thước lớn hơn, xác định theo kích thước của khung hình.



Thực hiện phép xoay ảnh (*Shear*) góc 45 độ, sau đó phủ lên ảnh gốc. Tham khảo [13] [14]

❖ Tham số đầu vào: mảng vector pixels của ảnh có kiểu *np.array*, giá trị không gian gốc *dim*, giá trị *scale* của khung ellipse mặc định là *1.25*, góc xoay khung ellipse *angle* mặc định là *45*.

❖ Kết quả trả về: mảng vector pixels của ảnh có kiểu *np.array* đã gắn khung ellipse.

❖ Mô tả:

➢ Đổi đơn vị độ của *angle* thành radian, tạo biến *sine, cosine* ứng với góc xoay *angle*.

```
angle = math.radians(angle)
sine = math.sin(angle)
cosine = math.cos(angle)
```

- Xác định cạnh khung ellipse bằng chiều dài cạnh nhỏ nhất của hình, độ lớn có thể thay đổi theo *scale*. Đồng thời xác định cạnh khung sau khi đã được xoay.
  Ta có công thức tính cạnh sau khi xoay như sau: [13]

$$new\_width = |old\_width \times cos\theta| + |old\_height \times sin\theta|$$
$$new\_height = |old\_height \times cos\theta| + |old\_width \times sin\theta|$$

```
#Define edge of the ellipse picture
edge = int(min(width, height)*scale)
#Define new edge after rotation of the ellipse picture
new_edge  = np.ceil(abs(edge * cosine) + abs(edge * sine)).astype(int)
```

- Sử dụng hàm *np.ogrid* [11] của *numpy* để tạo giá trị tọa độ *x_ellips, y_ellips* tương ứng với cạnh của khung. Xác định bán kính nhỏ và to của ellips.

```
x_ellips, y_ellips = np.ogrid[:edge, :edge]
smallRadius = edge/4
largeRadius = edge/2
```

- Sau đó tạo mảng *mask_ellips1*, *mask_ellips2* với kiểu boolean chứa kết quả chân trị của từng cặp *x_ellips*, *y_ellips* trong biểu thức bất phương trình nằm ngoài ellipse:

$$Ellipse\ ngang: \frac{(x-x_0)^2}{a^2} + \frac{(y-y_0)^2}{b^2} > 1$$

$$Ellipse\ dọc: \frac{(x-x_0)^2}{b^2} + \frac{(y-y_0)^2}{a^2} > 1$$

a là bán kính lớn, b là bán kính nhỏ, tâm tại $I(x_0, y_0)$

```
mask_ellips1 = (x_ellips-edge/2)**2/largeRadius**2 + (y_ellips-edge/2)**2/smallRadius**2 > 1
mask_ellips2 = (x_ellips-edge/2)**2/smallRadius**2 + (y_ellips-edge/2)**2/largeRadius**2 > 1
```

Sau đó dùng phép giao logic or để giao hai hình lại.

```
mask_ellips = mask_ellips1 & mask_ellips2
```

- Xác định vị trí của các giá trị mới sau khi được xoay từ khung cũ. Ta có công thức ánh xạ qua như sau:

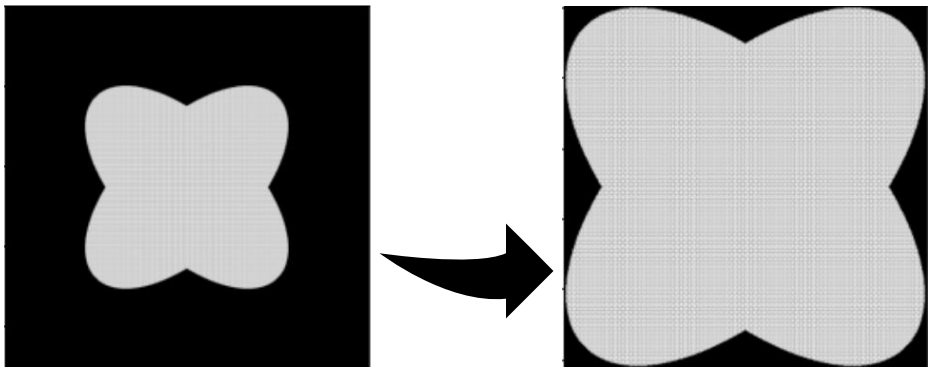
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix}$$

```
y = edge/2 - y_ellips
x = edge/2 - x_ellips
x_new = (new_edge/2 - np.ceil(x*cosine + y*sine)).astype(int)
y_new = (new_edge/2 - np.ceil(-x*sine + y*cosine)).astype(int)
```

- Sau khi có các vị trí mới của từng giá trị, thực hiện mask tất cả giá trị của khung cũ:

```
rotatedMask_ellips = np.ones((new_edge, new_edge), dtype = bool)
rotatedMask_ellips[x_new, y_new] = mask_ellips[x_ellips, y_ellips]
```

- Sau đó, cắt bỏ phần dư sau khi xoay như hình minh họa sau:



Đồng thời xử lý thêm để khung nằm giữa hình ảnh. Xem thêm trong mã nguồn.

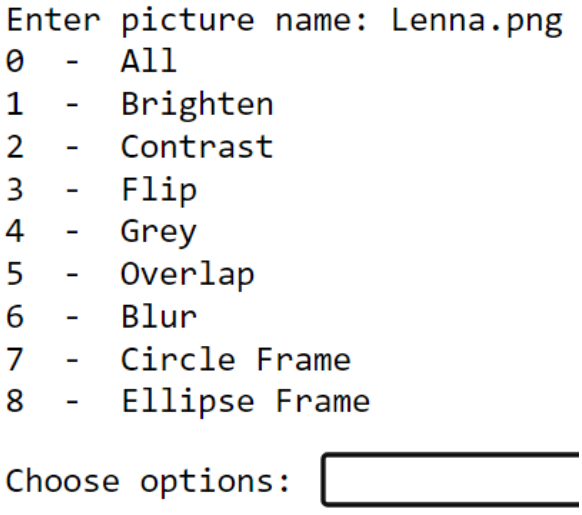
- Xóa giá trị màu tại các vị trí mà kết quả chân trị của *rotatedMask_ellips* là *True* bằng cách gán với vector [0, 0, 0]

```
image[rotatedMask_ellips] = np.zeros(dim[2])
```

9) *main.*

Hàm main chạy chính của chương trình, dùng để đọc hình ảnh người dùng và cho phép người dùng lựa chọn các thao tác trên ảnh để xử lý ảnh của người dùng và xuất ra kết quả.

- ❖ Cho phép người dùng nhập tên ảnh để đọc và lưu lại dưới dạng ma trận theo numpy.
- ❖ Hiện menu các chức năng xử lý ảnh và cho phép người dùng chọn thao tác trên ảnh hoặc nhấn phím 0 để thực hiện tất cả các thao tác.



7 - Menu

  - Đối với chức năng lật ảnh (*Flip*), người dùng có thể lựa chọn lật ảnh theo chiều dọc (*Flip Vertically*) bằng cách chọn phím 0 hoặc theo chiều ngang (*Flip Horizontally*) bằng cách chọn phím 1.
  - Đối với chức năng chồng 2 ảnh (*Overlap*), người dùng được yêu cầu nhập tên của tấm ảnh thứ 2 với kích thước bằng với ảnh đầu để thực hiện thao tác chồng.
- ❖ Xử lý tháo tác ảnh được gửi dùng chọn bằng hàm *generate_image*. Hàm có chức năng đổi không gian ma trận ảnh thành dạng mảng các vector pixels (dài*rông, 3) và xử lý thao tác theo người dùng chọn, trả về mảng vector pixels của ảnh có kiểu *np.array* đã được xử lý xong.
- ❖ Hiện ảnh kết quả thông qua hàm *show_image* bằng thư viện *matplotlib.pyplot*. Thực hiện việc sắp xếp lại không gian các vector pixel thành không gian ma trận lúc đầu để hiển thị hình ảnh kết

quả.

❖ Lưu ảnh mới vừa được xử lý về máy thông qua hàm *save_image* ở dạng .png. Đồng thời tự động đặt tên file kèm đuôi thể hiện chức năng xử lý tương ứng.

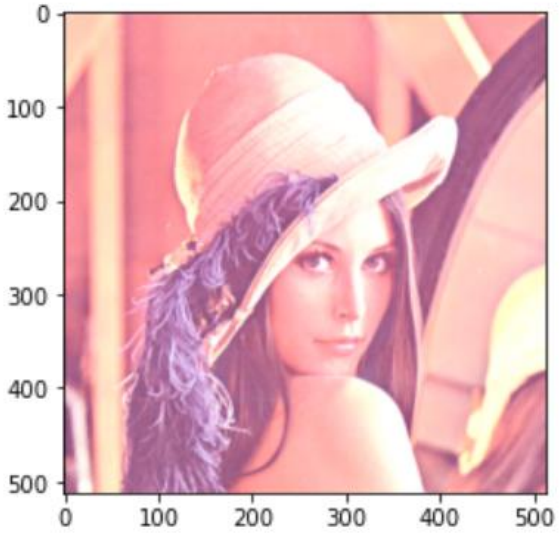
IV. Hình ảnh kết quả.

Thử nghiệm xử lý hình ảnh để đánh giá mức độ hoàn thiện của thuật toán như sau:

8 - Ảnh gốc 512x512

Với chức năng thay đổi độ sáng hình ảnh - *Brighten*:

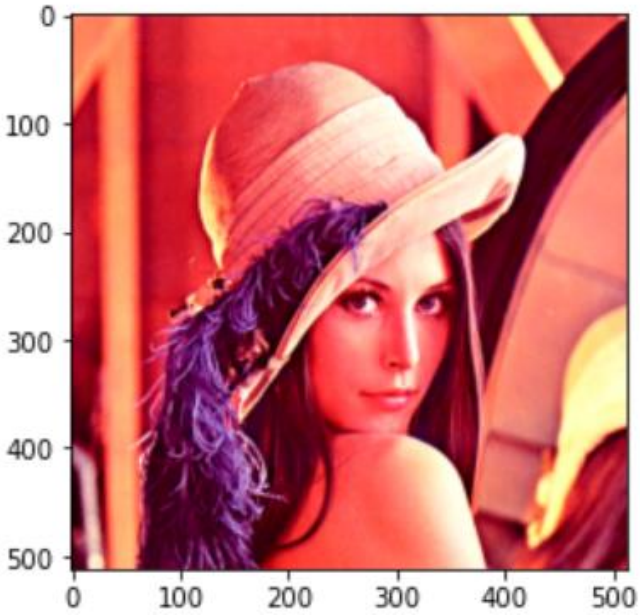
```
Image's size:  (512, 512, 3)
Option:  1 - Brighten
```



9 - Ảnh 512x512 xử lý thay đổi độ sáng

Với chức năng thay đổi độ tương phản hình ảnh - *Contrast*:

```
Image's size:  (512, 512, 3)
Option:  2 - Contrast
```

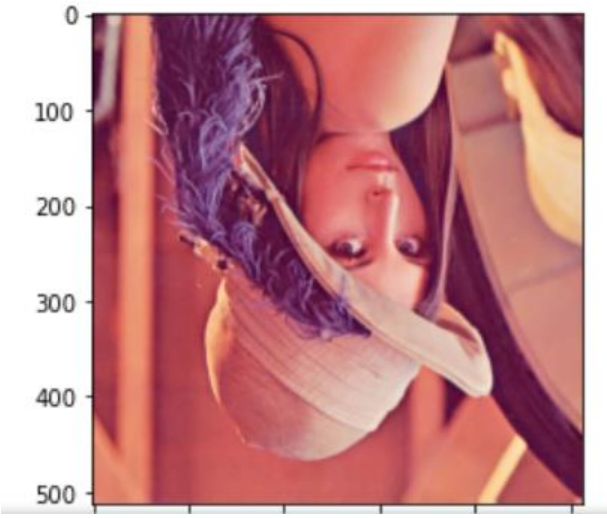


10 - Ảnh 512x512 xử lý thay đổi độ tương phản

Với chức năng lật hình ảnh - *Flip*:

+ Lật dọc:

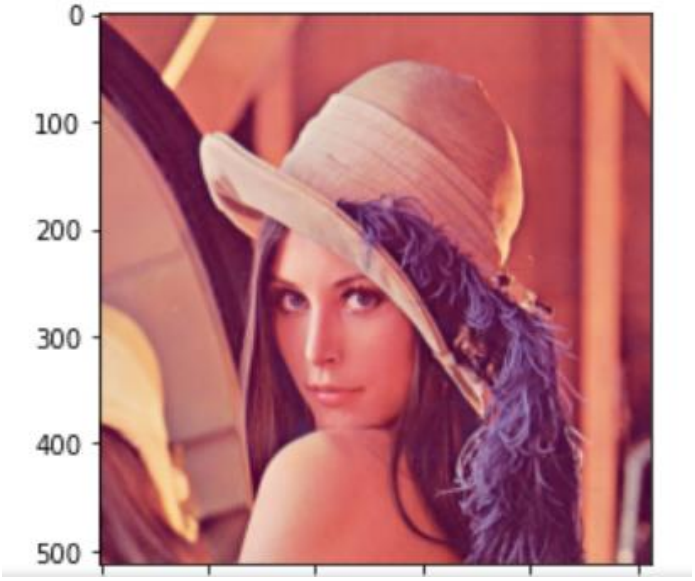
```
0 - Flip Vertically
1 - Flip Horizontally
Choose options: 0
Image's size:  (512, 512, 3)
Option:  3 - Flip
```



11 - Ảnh 512x512 xử lý lật dọc

+ Lật ngang:

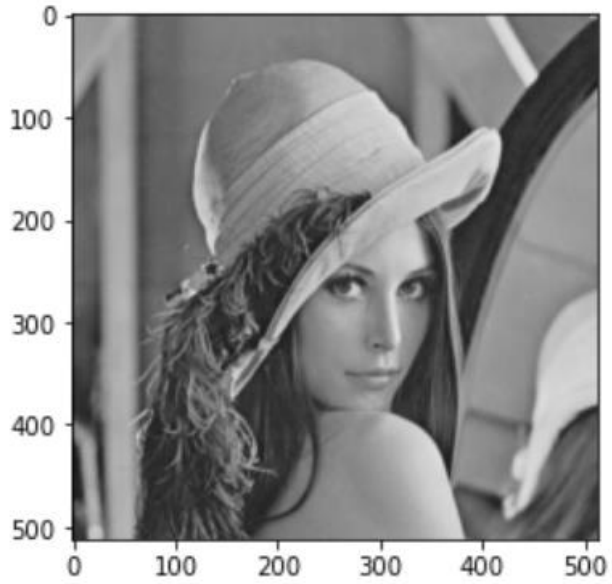
```
0 - Flip Vertically
1 - Flip Horizontally
Choose options: 1
Image's size:  (512, 512, 3)
Option:  3 - Flip
```



12 - Ảnh 512x512 xử lý lật ngang

Với chức năng làm xám hình ảnh - *Grey*:

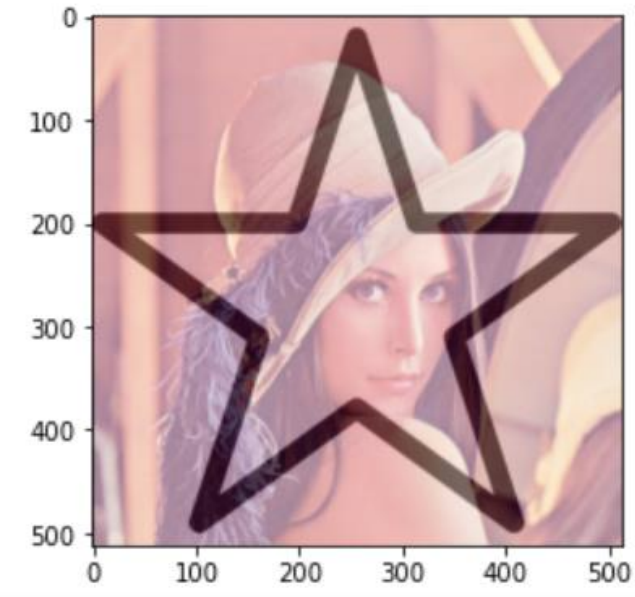
```
Image's size:  (512, 512, 3)
Option:  4 - Grey
```



13 - Ảnh 512x512 xử lý làm xám

Với chức năng chồng 2 hình ảnh - *Overlap*:

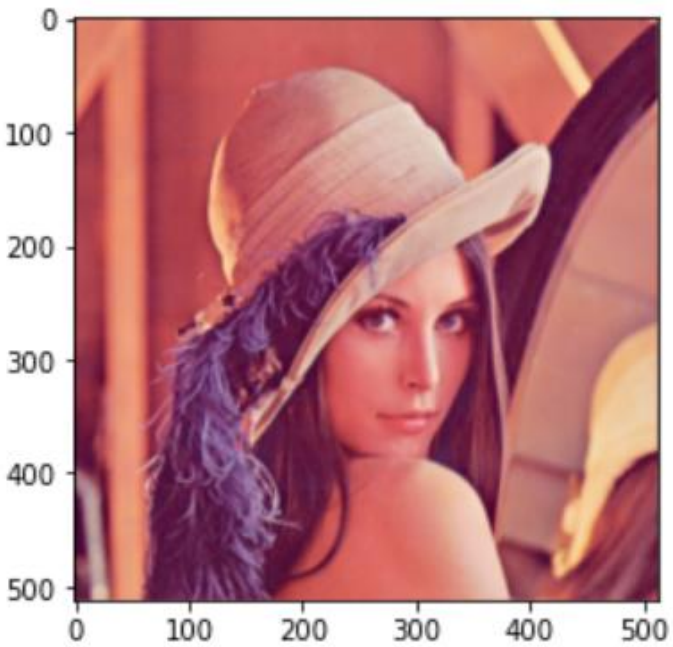
```
Enter another picture name to overlap: Star.png
Image's size:  (512, 512, 3)
Option:  5 - Overlap
```



14 - Ảnh 512x512 xử lý chồng ảnh

Với chức năng làm mờ hình ảnh - *Blur*:

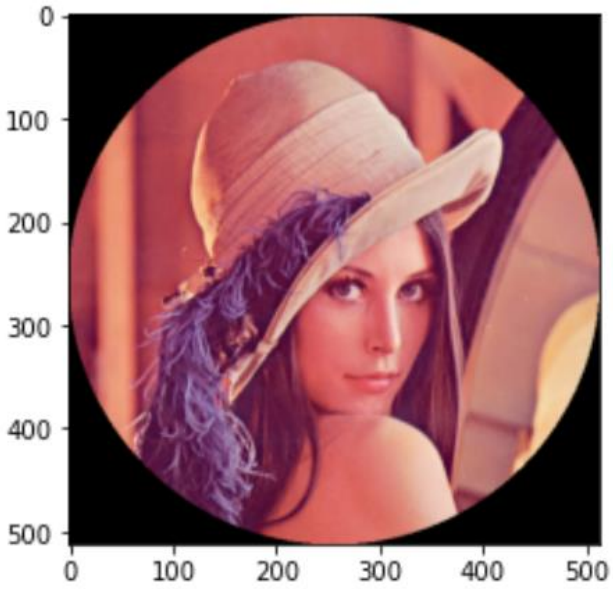
```
Image's size:  (512, 512, 3)
Option:  6 - Blur
```



15 - Ảnh 512x512 xử lý làm mờ

Với chức năng gắn khung tròn cho hình ảnh – *Circle Frame*:

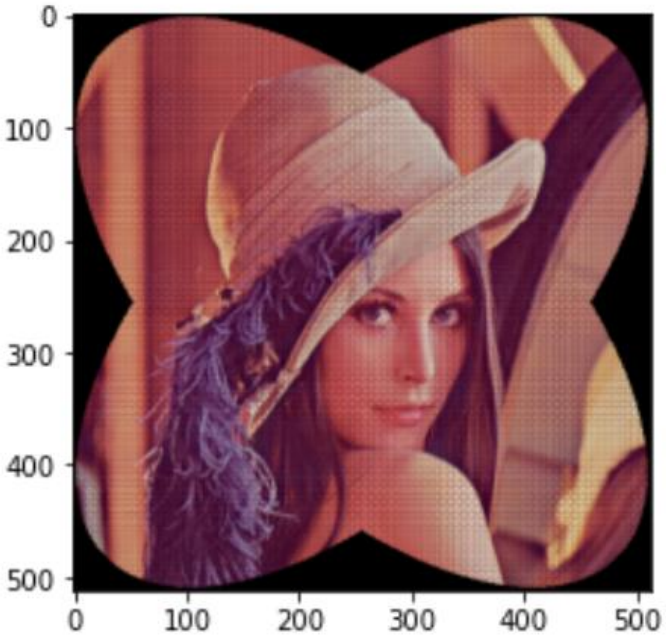
```
Image's size:  (512, 512, 3)
Option:  7 - Circle Frame
```



16- Ảnh 512x512 được gắn khung tròn

Với chức năng gắn khung 2 ellipses xoay chéo chồng lên nhau cho hình ảnh – *Ellipse Frame*.

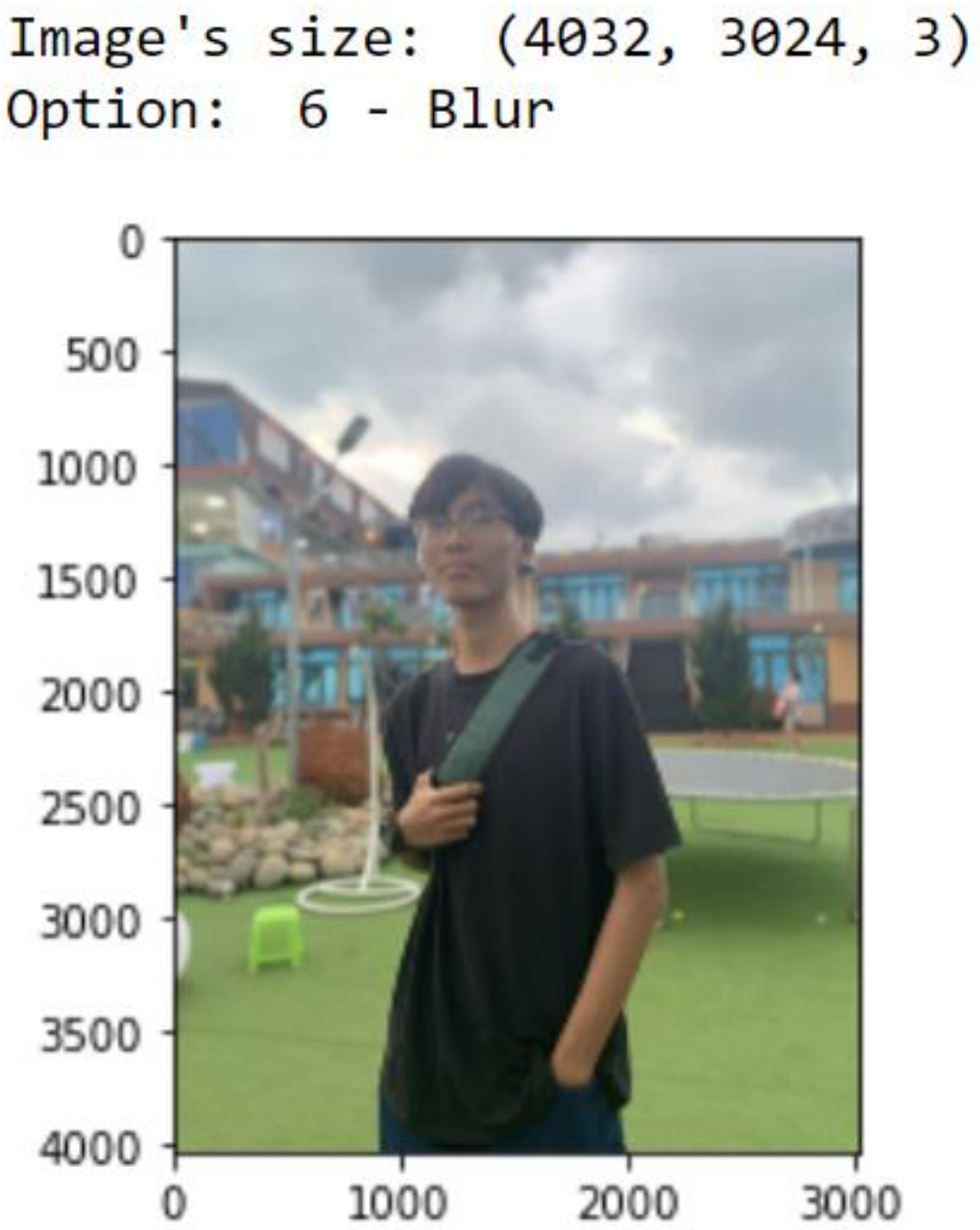
```
Image's size:  (512, 512, 3)
Option:  8 - Ellipse Frame
```



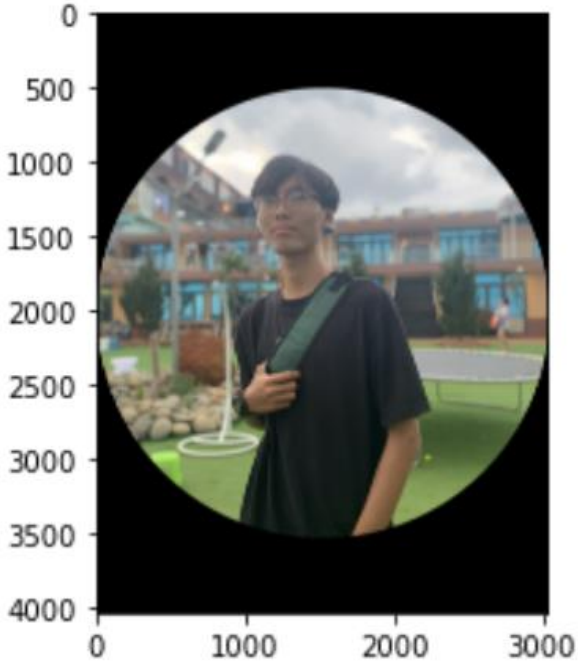
17 - Ảnh 512x512 được gắn khung 2 ellipses

Nhận xét: Mức độ hoàn thiện của ảnh kết quả xử lý các chức năng tốt. Thời gian xử lý đối với thap tác làm mờ ảnh *Blur* cho ảnh 512x512 trong vòng vài giây, khá nhanh và chấp nhận được.

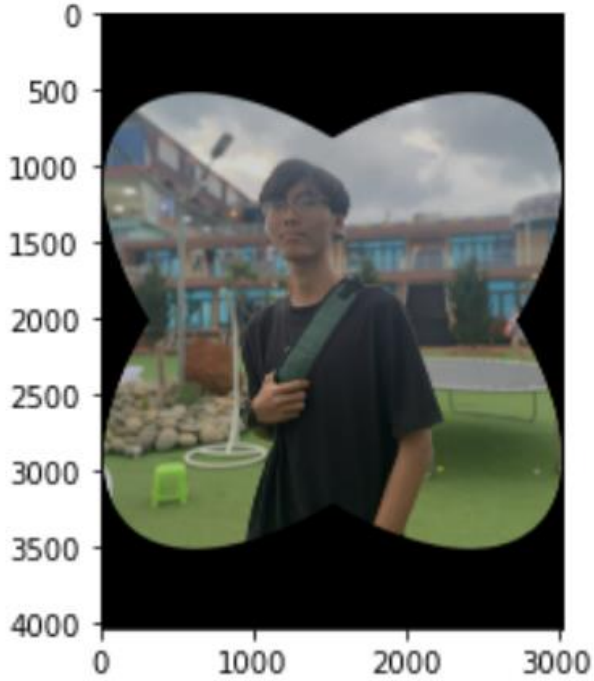
Một số chức năng cho hình kích thước không vuông:



```
Image's size:  (4032, 3024, 3)
Option:  7 - Circle Frame
```




```
Image's size:  (4032, 3024, 3)
Option:  8 - Ellipse Frame
```



V. Tài liệu tham khảo

Đồ án 2.

https://github.com/ThongLai/ImageProcessing

Tham khảo.

[1] https://www.codeproject.com/Articles/33838/Image-Processing-using-C

[2] https://numpy.org/doc/stable/reference/generated/numpy.flip.html

[3] https://stackoverflow.com/questions/1550130/cloning-row-or-column-vectors

[4] https://stackoverflow.com/questions/29920114/how-to-gauss-filter-blur-a-floating-point-numpy-array

[5] https://en.wikipedia.org/wiki/Kernel_(image_processing)

[6] https://www.stdio.vn/computer-vision/phep-tich-chap-trong-xu-ly-anh-convolution-r1vHu1

[7] https://numpy.org/doc/stable/reference/generated/numpy.convolve.html

[8] https://numpy.org/doc/stable/reference/generated/numpy.roll.html

[9] https://en.wikipedia.org/wiki/Convolution

[10] https://stackoverflow.com/questions/44865023/how-can-i-create-a-circular-mask-for-a-numpy-array

[11] https://numpy.org/doc/stable/reference/generated/numpy.ogrid.html

[12] https://towardsdatascience.com/the-little-known-ogrid-function-in-numpy-19ead3bdae40#:~:text=What%20is%20ogrid%3F,their%20row%20and%20column%20index

[13] https://gautamnagrawal.medium.com/rotating-image-by-any-angle-shear-transformation-using-only-numpy-d28d16eb5076

[14] https://en.wikipedia.org/wiki/Shear_mapping