SIT789 - Robotics, Computer Vision and Speech Processing

Pass Task 1.1: Image IO, colour conversion and geometric transformations

---

Objectives

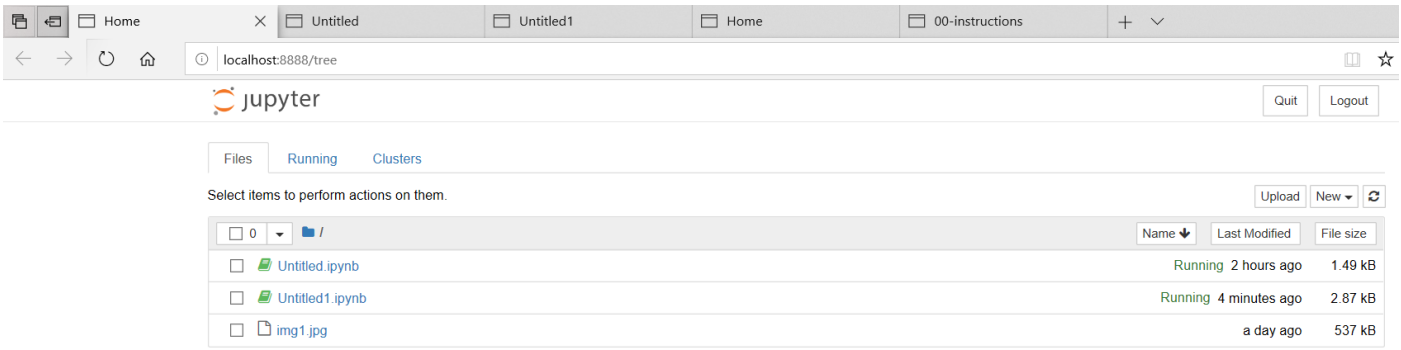
This lab introduces basic steps of image processing using OpenCV including:

- Image IO
- Colour conversion
- Geometric transformations

Tasks

1. Image IO with OpenCV

In this task, we will practise with reading/writing images from/to files. First, you need to launch your Jupyter Notebook.



You then need to copy and unzip the resources provided to you (in OnTrack) into your working directory. For this task, you are provided with a notebook and an image file (Lenna.png). You need to navigate to this notebook from File/Open on the menu of your Jupyter Notebook.

In the provided notebook, you will see the following command lines

```
import numpy as np
import cv2 as cv
img = cv.imread('Lenna.png')
```

You need to select the cell above (by clicking it) and then click the **Run** button on the toolbar of the Jupyter Notebook or alternatively press **Shift+Enter** to execute the cell. The above code loads the image file 'Lenna.png'

into a variable named 'img'. After running the code in the cell, a new cell is created for you. You can also create a new cell by clicking the **+** symbol on the toolbar of the Jupyter Notebook. To get the dimension of img (i.e., the height and width of img in pixels), you can perform:

```
height, width = img.shape[:2]
print (height, width)
```

To display img, you can use the following commands:

```
from matplotlib import pyplot as plt
plt.imshow(img)
```

Note: The displayed image does not look like the image in Lenna.png. This is because OpenCV uses BGR (Blue-Green-Red) as the default setting for image displaying. To show the image in RGB. You can do as follow.

```
# cv.cvtColor(img, cv.COLOR_BGR2RGB): convert img from BGR to RGB
plt.imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))
```

The provided image (Lenna.png) is stored in PNG format. You can change its format, e.g., to JPEG, by performing the following code:

```
cv.imwrite('Lenna.jpeg', img)
```

You can compare Lenna.png and Lenna.jpeg in terms of their quality and size by using any image viewer software, e.g., Microsoft Paint.

2. Colour conversion

The image img is represented in the BGR (Blue-Green-Red) space by default. To convert img into the HSV space, you can do as follows:

```
img_hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)
plt.imshow(img_hsv, cmap='hsv')
```

You can also convert img into a gray-scale image by:

```
img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
plt.imshow(img_gray, cmap='gray')
```

3. Geometric transformations

3.1. Scaling

Scaling aims to change the dimensions of an image and can be done using cv.resize method. For example, to resize the image img by a horizontal scale of 0.5 and vertical scale of 0.4, you can do,

```
height, width = img.shape[:2]
h_scale = 0.5
v_scale = 0.4
new_height = (int) (height * v_scale) # (int) is used to make new_height an integer
new_width = (int) (width * h_scale) # (int) is used to make new_width an integer
img_resize = cv.resize(img, (new_width, new_height), interpolation = cv.INTER_LINEAR)
plt.imshow(cv.cvtColor(img_resize, cv.COLOR_BGR2RGB))
```

3.2. Translation

Translation transformation shifts an image to a new location determined by a translation vector. In this task, you are to apply geometric transformations on img using cv.warpAffine function. In particular, you first need to determine the translation vector t=[$t.x$,$t.y$]

```
t_x = 100
t_y = 200
```

You then define a transformation matrix $M = \begin{bmatrix} 1 & 0 & t.x \\ 0 & 1 & t.y \end{bmatrix}$ using the following command,

```
M = np.float32([[1, 0, t_x], [0, 1, t_y]])
```

To make sure that the translated image will have the same dimension with the original image, you need to pass the height and width of img into cv.warpAffine as follows,

```
height, width = img.shape[:2] # height/width: the number of rows/columns in img
img_translation = cv.warpAffine(img, M, (width, height))
plt.imshow(cv.cvtColor(img_translation, cv.COLOR_BGR2RGB))
```

3.3. Rotation

For rotation, the transformation matrix M can be defined based on the rotation angle θ, rotation centre c, and isotrophic scale factor s as,

$$M = \begin{bmatrix} \alpha & \beta & (1-\alpha)c.x - \beta c.y \\ -\beta & \alpha & \beta c.x + (1-\alpha)c.y \end{bmatrix}$$

where

$$\alpha = s\cos\theta$$
$$\beta = s\sin\theta$$

For example, if we want to rotate the image img 45 degrees in anti-clockwise, we can set,

```
theta = 45 # rotate 45 degrees in anti-clockwise
```

If we want to rotate img around its centre c (note that the coordinate origin is assumed to be the top-left corner of the image), we can define:

```
c_x = (width - 1) / 2.0 # column index varies in [0, width-1]
c_y = (height - 1) / 2.0 # row index varies in [0, height-1]
c = (c_x, c_y) # A point is defined by x and y coordinate
print(c)
```

Assuming no scaling is applied in rotation, i.e., the scale s is set to 1, the matrix M is finally calculated using cv.getRotationMatrix2D as follow,

```
s = 1
M = cv.getRotationMatrix2D(c, theta, s)
```

You can check the matrix MM by printing it out.

```
print(M)
```

The rotation is then performed as,

```
img_rotation = cv.warpAffine(img, M, (width, height))
plt.imshow(cv.cvtColor(img_rotation, cv.COLOR_BGR2RGB))
```

3.4. Affine

In general, an affine transformation can be defined via any matrix *M* expressed in the form,

$$M = \begin{bmatrix} m_{0,0} & m_{0,1} & m_{0,2} \\ m_{1,0} & m_{1,1} & m_{1,2} \end{bmatrix}$$

In this task, you are to experiment with cv.warpAffine via various settings of *M*. For example, you can define,

```
m00 = 0.5
m01 = 0.3
m02 = -47.18
m10 = -0.4
m11 = 0.5
m12 = 95.32

M = np.float32([[m00, m01, m02], [m10, m11, m12]])
```

Apply cv.warpAffine with *M* defined above:

```
height, width = img.shape[:2]
img_affine = cv.warpAffine(img, M, (width, height))
plt.imshow(cv.cvtColor(img_affine, cv.COLOR_BGR2RGB))
```

Open question: Can we use cv.warpAffine to replace cv.resize?

Submission instructions

1. Complete the supplied answer sheet with required contents. Make a pdf for the answer sheet and submit it to OnTrack.
2. Save your notebook into a python (.py) file by doing (from the Menu of the Jupyter Notebook) File → Download as → Python (.py), and submit that python (.py) file to OnTrack. **Note**: please do NOT submit your notebook to OnTrack.