PYTHON 2-D: COMPUTER VISION

1. 2-D DATA (IMAGES)

Images (and video) can be stored in a variety of formats. However, the most important distinction is between "compressed" and "raw" (uncompressed) data. Raw data can be loaded into system memory (RAM) by directly reading the stored bytes, whereas compressed images must first be decoded, a process which requires knowing how the image was encoded in the first place. Fortunately, many image reading functions handle the detection and application of the correct decoder for you, but it is important to understand what is happening. We will explore differences in compression techniques by loading, processing, and displaying the same image encoded with different methods.

Loading image data requires a special library that understands how to deal with image compression. There are many such libraries, and here we will be using OpenCV. *Make sure that you have OpenCV installed in your Python environment.*

2. IMAGE PROCESSING

Let's try to open a random image. Open Spyder and start writing our first Python+OpenCV program. Enter the following code into a new Python script.

```
# Import the OpenCV package
import cv2

# Import the plotting package
import matplotlib.pyplot as plt

# Make a string that contains the file path of an image (explain r and filepath)
filename= r'C:/SomeImagePath\image.jpg'

# Load the image
image = cv2.imread(filename)

# Show image
plt.imshow(image)

# Save the image
cv2.imwrite(r'C:/SomeImagePath\new_image.jpg', image)

#FIN
```

EXERCISE 2.1.1: BINARY IMAGE

- Add to the code above functions from the OpenCV library, to convert your image from colour to grayscale. Then, threshold the image thus converting it from grayscale to binary zero/one, and display the result.

EXERCISE 3.1.2: COLOURED IMAGES

- Which function would you use in order to extract a selected object from a coloured image?

3. VIDEO PROCESSING

It is very common to want to specifically detect the features/objects in a movie that are moving, i.e. changing in their intensity relative to static features. Therefore, many methods in computer vision have been developed to generate an estimate of the static scene, i.e. a "background" from which objects in the "foreground" can be isolated. Most of these methods normally require having access to multiple images in which the object of interest is at different locations. Therefore, to start developing methods to differentiate static vs dynamic images we must first learn how to load the frames of a video sequence.

TASK 3.1: VIDEO PROCESSING

Videos can be encoded (compressed) in many different formats (mpeg, dvix, h.264 etc.). In order to read the data (decode) decompress video, Python/OpenCV requires the correct "codec" - a separate piece of software that can interpret a specific style of video compression. The appropriate codec is not always immediately accessible to Python/OpenCV. For example, and as we use below, the code for interpreting mpeg-h.264 (a very common video compression format) must be explicitly located in the main Python directory. [Yes, I know this is clunky, however, there is always something clunky, and overcoming these seemingly irrational obstacles to getting a computer program do what you want is often the main challenge. Google it or email it to your instructor!].

```
# OpenCV.package
import cv2
import matplotlib.pyplot as plt

# Make a string that contains the file path of the movie
filename= r'C:/Users/You/Desktop/video.avi'
cap = cv2.VideoCapture(filename)

for i in range(0,2):

    # Capture frame-by-frame
    ret, frame = cap.read()

    #Your algorithm goes in here!!

    # Show result on a screen
    plt.imshow(frame)
    plt.draw()
    plt.pause(0.001)
    plt.clf()

cap.release()

#FIN
```

Very Important: In order to make Python/OpenCV aware of the "FFmpeg codec", copy the "opencv_ffmpeg249_64.dll" (this should be the path "…\opencv\build\x64\vc14\bin\ opencv_ffmpeg330_64.dll") to the main Python directory (where the Python.exe is located; e.g. "C:\Anaconda'').

EXERCISE 3.1.1: COMPUTE THE BACKGROUND

- Copy the code above and load one of the videos you have saved.
- Now that you can load all the frames of a movie how would you go about computing the "background", i.e. the intensity of each pixel in the static scene when there is nothing changing?
- One obvious approach is to just generate the average frame by selecting many frames from the movie where the object is in different locations. The code to do this is shown below. However this technique is not ideal. Why? What could you do to generate a more accurate estimate of the background?

```
# OpenCV.package
import cv2
import matplotlib.pyplot as plt
import numpy as np

# Make a string that contains the file path of the movie
filename= r'C:/Users/Elena/Desktop/video.avi'

# It is the handle that gives you the location of the movie
cap = cv2.VideoCapture(filename)

# Get width and height properties of the image
width= cap.get(cv2.cv.CV_CAP_PROP_FRAME_WIDTH)
height= cap.get(cv2.cv.CV_CAP_PROP_FRAME_HEIGHT)

# Compute the background
Bkg= np.zeros((height,width,20))
Bkg_count=0
for i in range (0,200,10):

        # It sets in which position you are in the movie
        cap.set(cv2.cv.CV_CAP_PROP_POS_FRAMES, i)

        # It opens the frame of the movie located at the position defines above
        ret, frame = cap.read()

        gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Sum of all the Bkg frames
        Bkg[:,:,Bkg_count]= gray_frame
        Bkg_count=1+Bkg_count

Bkg_image = np.mean(Bkg, axis=2)
plt.imshow(Bkg_image)
plt.draw()

cap.release()

#FIN
```

EXERCISE 3.1.2: BETTER BACKGROUND

- Change the code above to generate and display a better estimate of the background

4. OBJECT TRACKING

Tracking an object is a basic goal of computer vision and fundamental to many neuroscience experiments. The fundamental premise of tracking is to find a way to segment the object you want to track from the rest of the image, and then use operations on the binary image to compute the location and other features of the object of interest (area, size, orientation, shape, etc.).

It is often not possible to uniquely identify the pixels belonging to the desired object vs others simply due to noise. Therefore, it is important that a tracking algorithm is able to distinguish which grouping of foreground pixels is valid (e.g. they are all grouped together, they are the largest group, they are not on the border, etc.)

In OpenCV, binary regions are located by finding the contours of connected islands of foreground pixels, i.e. the boundary between background and foreground. When connected contours have been detected, one can begin to measure several features of these isolated regions (e.g. length, centroid, size, area), or to exclude, for example, contours that are too big or too small.

```
# Some pseudo code
cap = cv2.VideoCapture(filename)
plt.figure()

for i in range(0,2000):

        # Capture frame-by-frame
        ret, frame = cap.read()
        gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        Bkg_image=Bkg_image.astype(np.uint8)

        # SUBSTRACT THE BACKGROUND from the movie and find absolute difference
        Bkg_Sub_Frame=cv2.absdiff(Bkg_image, gray_frame)

        # THRESHOLD the absolute difference movie
        ret,thresh_frame = cv2.threshold(Bkg_Sub_Frame,5,255,cv2.THRESH_BINARY)

        # Find all the particles as CONTOURS in the movie
        Particle, hierarchy=cv2.findContours(thresholded_frame,
        cv2.cv.CV_RETR_EXTERNAL,cv2.cv.CV_CHAIN_APPROX_NONE)

        # GET THE INDEX of BIGGEST particle
        P_size=np.zeros(np.size(Particle))
        for p,idx in enumerate (Particle):
                P_size[p]= np.size(idx)
                Big_Particle=np.argmax(P_size)
        # returns the index within P_size array that contains the biggest particle

        # Get the centroid of the biggest particle by averaging the centroid coordinates
        Particle_CenroidXY=np.mean(Particle[Big_Particle], axis=0)

        # Show result in a plot
        plt.imshow(gray_frame)
        plt.plot(Particle_CenroidXY[0,0],Particle_CenroidXY[0,1],'o')
        plt.draw()
        plt.pause(0.001)
        plt.clf()

cap.release()
```

Bonus Exercise 4.1.1: Try to get more information beyond the position of the object (area, orientation, shape)