Aim: To perform Handling Files, Cameras and GUIs

Objective: To perform Basic I/O Scripts, Reading/Writing an Image File, Converting Between an Image and raw bytes, Accessing image data with numpy.array, Reading /writing a video file, Capturing camera, Displaying images in a window, Displaying camera frames in a window

Theory:

To use the OpenCV library in python, we need to install Numpy Library and OpenCV python.

Basic I/O script

Reading/Writing an Image File

The steps to read and display an image in OpenCV are :

1. Read an image using imread() function. This function reads the image from a specific location from a directory in the computer.

syntax: cv2.imread(path, flag)

Parameters:

path: It is a string that specifies the location of the image.

flag: It specifies the way in which the image should be read. It’s default value of flag is cv2.IMREAD_COLOR

Return Value: This method returns an image that is loaded from the specified file.

2. Create a GUI window and display an image using imshow() function.

syntax:imshow(string,img)

it creates GUI window to display an image on screen

first Parameter is windows title (should be in string format)

Second Parameter is the image array.

3. Use function waitkey(0) to hold the image window on the screen by the specified number of seconds, o means till the user closes it, it will hold GUI window on the screen.

4. Delete image window from the memory after displaying using destroyAllWindows() function.

Converting between image and raw bytes

Determine the Raw Image Properties: Before reading a raw image, you need to know some crucial information about the image, such as its dimensions, pixel format, and whether it has any header or metadata. You should have this information beforehand, as it's essential for reading the raw data correctly.

Read the Raw Image Data: To read the raw image data from the file, you can use the open() function to open the file and then use the read() function to read the binary data into a buffer

Converting between an image and raw bytes involves two main steps: encoding an image to raw bytes and decoding raw bytes back into an image.

Converting an Image to Raw Bytes (Encoding):

byte_stream = io.BytesIO()

In this example, we load an image file named "example.jpg" using the PIL library, then create an in-memory byte stream (io.BytesIO) to save the image into. Finally, we save the image in PNG format to the byte stream and obtain the raw bytes using byte_stream.getvalue().

Converting Raw Bytes to an Image (Decoding):

byte_stream = io.BytesIO(raw_bytes)

In this example, we create another in-memory byte stream using the raw bytes obtained earlier. We then use Image.open() to open the image from the byte stream, effectively converting the raw bytes back into an image.

Accessing image data with numpy. Array

image_path = 'path/to/your/image.jpg'

image = cv2.imread(image_path)

Now, image will be a NumPy array representing the loaded image.

pixel_value = image[y, x]

One can access and modify individual pixel values using array indexing.

For example, to get the value of a pixel at coordinates (x, y) in the image

Reading/Writing a video file

To read and write a video file using OpenCV in Python, you'll use the cv2.VideoCapture class for reading and the cv2.VideoWriter class for writing

```
cap = cv2.VideoCapture(input_video_path)


while True:
    # Read the frame
    ret, frame = cap.read()


    # Break the loop if no frame is retrieved
    if not ret:
        break
```

For writing the video, One can create each frame individually and use the cv2.VideoWriter.write() method to write it to the video file.

Capturing camera frames

OpenCv lets you to create a video capture object helpful to capture videos through webcam and perform the desired operations.

Steps:

1. Use cv2.VideoCapture() to0 get a video capture object from the camera.

```
    # import the openCv library
```

```
import cv2
# define a video capture object
vid = cv2.VideoCapture(0)
```

2. Set up an infinite while loop and use the read() method to read the frames using the above created object.

```
while(True) :
ret,  frame = vid.read()
```

3. Use cv2.imshow() method to show the frames in the video.

```
cv2.imshow('frame' , frame)
```

4. Break the loop when the user clicks a specific key.

```
if cv2.waitKey(1) & 0xFF ==ord('q'):
        break
vid.release()
cv2.destroyAllWindows()
```

Displaying images in a window

```
image = cv2.imread(image_path)
cv2.imshow('Image', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

To display images in a window using OpenCV in Python, you can use the cv2.imshow function.

The cv2.imshow function takes two arguments: the window name (a string) and the image data (a NumPy array).

The cv2.waitKey(0) function waits until any key is pressed and returns the ASCII value of the pressed key. If you pass a positive integer argument (e.g., cv2.waitKey(100)), it will wait for the specified number of milliseconds.

Finally, the cv2.destroyAllWindows() function closes all the windows that were created during the execution.

Displaying camera frames in a window

OpenCV is a library that helps in providing various functions for image and video operations. With OpenCV, we can capture a video from the camera. It lets you create a video capture object which is helpful to capture videos through webcam and then you may perform desired operations on that video.

Steps :

Use cv2.VideoCapture() to get a video capture object for the camera.

Set up an infinite while loop and use the read() method to read the frames using the above created object.

Use cv2.imshow() method to show the frames in the video.

Breaks the loop when the user clicks a specific key.

code:

```
vid = cv2.VideoCapture(0)
while(True):
    ret, frame = vid.read()
    cv2.imshow('frame', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
vid.release()
cv2.destroyAllWindows()
```

Here the vid variable captures the video objects.where as vid.read() reads the video frame by frame while imshow() shows the results of the captured frame and ord(‘q’) is used as the quit button for exit.

Conclusion:

Upon completion of this experiment, one will gain a comprehensive understanding of fundamental operations utilizing openCv. These operations encompass tasks like image and video processing, including the introduction of time delays before image capture. The experiment will delve into techniques for image manipulation, covering the conversion between images and raw bytes in both directions. Additionally, the process of capturing individual video frames sequentially will be explored, along with the ability to designate a specific key as an exit command to promptly close the displayed capture window.