Technological Institute of the Philippines	**Quezon City - Computer Engineering**
Course Code:	CPE 313
Code Title:	Emerging Technologies in CpE 1 - Fundamentals of Computer Vision
2st Semester	AY 2024-2025
ACTIVITY NO. 3	Basic I/O Scripting, Part 2
Name	Catulay, Weslie Jee
Section	CPE32S3
Date Performed:	02/19/25
Date Submitted:	02/21/25
Instructor:	Dr. Jonathan V. Taylar / Engr. Verlyn V. Nojor / Engr. Roman M. Richard

1. Objectives

This activity aims to introduce students to OpenCV's I/O Functionality for video processing.

2. Intended Learning Outcomes (ILOs)

After this activity, the students should be able to:

- Read and write video files using openCV.
- Utilize openCV to capture and display images and videos.

3. Procedures and Outputs

NOTE: For this laboratory activity, it is recommended that you download and run the Python notebook on *Spyder IDE*. You must install dependencies by running `!pip install numpy` and `!pip install opencv-python==4.6.0.66`.

Reading/Writing a Video File

OpenCV provides the `VideoCapture` and `VideoWriter` classes that support various video file formats. The supported formats vary by system but should always include an AVI. Via its `read()` method, a `VideoCapture` class may be polled for new frames until it reaches the end of its video file. Each frame is an image in a BGR format.

Conversely, an image may be passed to the `write()` method of the `VideoWriter` class, which appends the image to a file in VideoWriter. Let's look at an example that reads frames from one AVI file and writes them to another with a YUV encoding:

```
import cv2

videoCapture = cv2.VideoCapture('MyInputVid.avi')

fps = videoCapture.get(cv2.CAP_PROP_FPS)
size = (int(videoCapture.get(cv2.CAP_PROP_FRAME_WIDTH)),
        int(videoCapture.get(cv2.CAP_PROP_FRAME_HEIGHT)))

videoWriter = cv2.VideoWriter(
    'MyOutputVid.avi', cv2.VideoWriter_fourcc('I', '4', '2', '0'),
    fps, size)

success, frame = videoCapture.read()
while success: # Loop until there are no more frames
  videoWriter.write(frame)
  success, frame = videoCapture.read()
```

The arguments to the VideoWriter class constructor deserve special attention. A video's filename must be specified. Any preexisting file with this name is overwritten. A video codec must also be specified. The available codecs may vary from system to system. These are the options that are included:

- `cv2.VideoWriter_fourcc('I','4','2','0')` : This option is an uncompressed YUV encoding, 4:2:0 chroma subsampled. This encoding is widely compatible but produces large files. The file extension should be .avi.
- `cv2.VideoWriter_fourcc('P','I','M','1')` : This option is MPEG-1. The file extension should be .avi.
- `cv2.VideoWriter_fourcc('X','V','I','D')` : This option is MPEG-4 and a preferred option if you want the resulting video size to be average. The file extension should be .avi.
- `cv2.VideoWriter_fourcc('T','H','E','O')` : This option is Ogg Vorbis. The file extension should be .ogv.
- `cv2.VideoWriter_fourcc('F','L','V','1')` : This option is a Flash video. The file extension should be .flv.

A frame rate and frame size must be specified too. Since we are copying video frames from another video, these properties can be read from the get() method of the VideoCapture class.

Capturing camera frames

A stream of camera frames is represented by the VideoCapture class too. However, for a camera, we construct a VideoCapture class by passing the camera's device index instead of a video's filename. Let's consider an example that captures 10 seconds of video from a camera and writes it to an AVI file:

```
import cv2

cameraCapture = cv2.VideoCapture(0)
fps = 30 # an assumption

size = (int(cameraCapture.get(cv2.CAP_PROP_FRAME_WIDTH)),
        int(cameraCapture.get(cv2.CAP_PROP_FRAME_HEIGHT)))

videoWriter = cv2.VideoWriter(
    'MyOutputVid.avi', cv2.VideoWriter_fourcc('I', '4', '2', '0'),
    fps, size)

success, frame = cameraCapture.read()
numFramesRemaining = 10 * fps - 1

while success and numFramesRemaining > 0:
  videoWriter.write(frame)
  success, frame = cameraCapture.read()
  numFramesRemaining -= 1

cameraCapture.release()
```

Unfortunately, the `get()` method of a VideoCapture class does not return an accurate value for the camera's frame rate; it always returns 0. The official documentation at http://docs.opencv.org/modules/highgui/doc/reading_ and_writing_images_and_video.html reads:

> "When querying a property that is not supported by the backend used by the VideoCapture class, value 0 is returned."

This occurs most commonly on systems where the driver only supports basic functionalities. For the purpose of creating an appropriate VideoWriter class for the camera, we have to either make an assumption about the frame rate (as we did in the code previously) or measure it using a timer.

The `read()` method is inappropriate when we need to synchronize a set of cameras or a multihead camera (such as a stereo camera or Kinect). Then, we use the `grab()` and `retrieve()` methods instead. For a set of cameras, we use this code:

```
"""

success0 = cameraCapture0.grab()
success1 = cameraCapture1.grab()
if success0 and success1:
  frame0 = cameraCapture0.retrieve()
  frame1 = cameraCapture1.retrieve()

"""
```

```
'\n\nsuccess0 = cameraCapture0.grab()\nsuccess1 = cameraCapture1.grab()\nif success0
and success1:\n  frame0 = cameraCapture0.retrieve()\n  frame1 =
cameraCapture1.retrieve()\n\n'
```

Displaying images in a window

One of the most basic operations in OpenCV is displaying an image. This can be done with the imshow() function. If you come from any other GUI framework background, you would think it sufficient to call imshow() to display an image. This is only partially true: the image will be displayed, and will disappear immediately. This is by design, to enable the constant refreshing of a window frame when working with videos. Here's a very simple example code to display an image:

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
img = cv2.imread('Isagi Yoichi.jpg')
cv2_imshow(img)
cv2.waitKey()
cv2.destroyAllWindows()
```

The `imshow()` function takes two parameters: the name of the frame in which we want to display the image, and the image itself. We'll talk about `waitKey()` in more detail when we explore the displaying of frames in a window.

The aptly named `destroyAllWindows()` function disposes of all the windows created by OpenCV.

Displaying camera frames in a window

OpenCV allows named windows to be created, redrawn, and destroyed using the `namedWindow()`, `imshow()`, and `destroyWindow()` functions. Also, any window may capture keyboard input via the `waitKey()` function and mouse input via the `setMouseCallback()` function. Let's look at an example where we show the frames of a live camera input:

```
import cv2

clicked = False

def onMouse(event, x, y, flags, param):
  global clicked
  if event == cv2.EVENT_LBUTTONUP:
    clicked = True

cameraCapture = cv2.VideoCapture(0)
cv2.namedWindow('MyWindow')
cv2.setMouseCallback('MyWindow', onMouse)

print('Showing camera feed. Click window or press any key to stop.')

success, frame = cameraCapture.read()

while success and cv2.waitKey(1) == -1 and not clicked:
  cv2.imshow('MyWindow', frame)
  success, frame = cameraCapture.read()

cv2.destroyWindow('MyWindow')
cameraCapture.release()
```

```
Showing camera feed. Click window or press any key to stop.
```

MyWindow_screenshot_21.02.2025.png

The argument for `waitKey()` is a number of milliseconds to wait for keyboard input. The return value is either `-1` (meaning that no key has been pressed) or an ASCII keycode, such as `27` for Esc. For a list of ASCII keycodes, see http://www.asciitable.com/. Also, note that Python provides a standard function, `ord()`, which can convert a character to its ASCII keycode. For example, `ord('a')` returns `97`.

OpenCV's window functions and `waitKey()` are interdependent. OpenCV windows are only updated when `waitKey()` is called, and `waitKey()` only captures input when an OpenCV window has focus.

The mouse callback passed to `setMouseCallback()` should take five arguments, as seen in our code sample. The callback's param argument is set as an optional third argument to `setMouseCallback()`. By default, it is 0. The callback's event argument is one of the following actions:

- `cv2.EVENT_MOUSEMOVE`: This event refers to mouse movement
- `cv2.EVENT_LBUTTONDOWN`: This event refers to the left button down
- `cv2.EVENT_RBUTTONDOWN`: This refers to the right button down
- `cv2.EVENT_MBUTTONDOWN`: This refers to the middle button down
- `cv2.EVENT_LBUTTONUP`: This refers to the left button up

- `cv2.EVENT_RBUTTONUP` : This event refers to the right button up
- `cv2.EVENT_MBUTTONUP` : This event refers to the middle button up
- `cv2.EVENT_LBUTTONDBLCLK` : This event refers to the left button being double-clicked
- `cv2.EVENT_RBUTTONDBLCLK` : This refers to the right button being double-clicked
- `cv2.EVENT_MBUTTONDBLCLK` : This refers to the middle button being double-clicked

The mouse callback's flags argument may be some bitwise combination of the following events:

- `cv2.EVENT_FLAG_LBUTTON` : This event refers to the left button being pressed
- `cv2.EVENT_FLAG_RBUTTON` : This event refers to the right button being pressed
- `cv2.EVENT_FLAG_MBUTTON` : This event refers to the middle button being pressed
- `cv2.EVENT_FLAG_CTRLKEY` : This event refers to the Ctrl key being pressed
- `cv2.EVENT_FLAG_SHIFTKEY` : This event refers to the Shift key being pressed
- `cv2.EVENT_FLAG_ALTKEY` : This event refers to the Alt key being pressed

Unfortunately, OpenCV does not provide any means of handling window events. For example, we cannot stop our application when a window's close button is clicked. Due to OpenCV's limited event handling and GUI capabilities, many developers prefer to integrate it with other application frameworks.

4. Supplementary Activity

Perform each of the following tasks.

1. Try reading and writing a video file in various formats.
2. Similar to activity #1, show an image of your favorite character on a window. Afterwards, slice so that only the character's face is displayed.
3. Capture video from your webcam and display on a window. Afterwards, the video should be written as a new file.

#1

```
import cv2

videoCapture = cv2.VideoCapture('input_video.avi')

fps = videoCapture.get(cv2.CAP_PROP_FPS)
width = int(videoCapture.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(videoCapture.get(cv2.CAP_PROP_FRAME_HEIGHT))
```

```
fourcc = cv2.VideoWriter_fourcc(*'XVID')
videoWriter = cv2.VideoWriter('output_video.avi', fourcc, fps, (width, height))

fourcc = cv2.VideoWriter_fourcc(*'H264')
videoWriter = cv2.VideoWriter('output_video.mp4', fourcc, fps, (width, height))

success, frame = videoCapture.read()
while success:
    videoWriter.write(frame)
    success, frame = videoCapture.read()

videoCapture.release()
videoWriter.release()
```

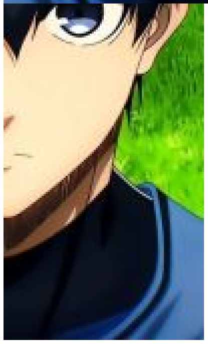
#2

```
import cv2
from google.colab.patches import cv2_imshow

img = cv2.imread('Isagi Yoichi.jpg')
cv2_imshow(img)
cv2.waitKey(0)
cv2.destroyAllWindows()

x, y, w, h = 100, 100, 200, 200
face_img = img[y:y+h, x:x+w]

cv2_imshow(face_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



#3

```
import cv2

clicked = False

def onMouse(event, x, y, flags, param):
  global clicked
  if event == cv2.EVENT_LBUTTONUP:
    clicked = True

cameraCapture = cv2.VideoCapture(0)
cv2.namedWindow('MyWindow')
cv2.setMouseCallback('MyWindow', onMouse)

print('Showing camera feed. Click window or press any key to stop.')

success, frame = cameraCapture.read()

while success and cv2.waitKey(1) == -1 and not clicked:
  cv2.imshow('MyWindow', frame)
  success, frame = cameraCapture.read()
```

```
cv2.destroyWindow('MyWindow')
cameraCapture.release()
```

```
Showing camera feed. Click window or press any key to stop.
```

![MyWindow_screenshot_21.02.2025 (2).png](<attachment:MyWindow_screenshot_21.02.2025 (2).png>)

5. Summary, Conclusions and Lessons Learned

In this activity we learned and apply all the things of how basic I_O scripting does work by utilizing openCV, By reading and writing videos in different formats, capturing and displaying webcam frames, and saving captured video, this exercise effectively investigated OpenCV's I/O capabilities for video processing. Along with functions like imshow and waitKey, we learned how to use important classes like VideoCapture and VideoWriter. I do encounterd some problems when working in the video capturing at first because colab is not capable of capturing video so i did use the VScode for this activity & using get() to obtain precise camera frame rates while successfully manipulating images and videos, which emphasizes the significance of making assumptions or using timer-based measurements. All things considered, the exercise gave me/us the insightful knowledge about OpenCV's usefulness in computer vision into its sophisticated features and variety of video formats for challenging tasks.