

Technological Institute of the Philippines		Quezon City - Computer Engineering	
Course Code:		CPE 313	
Code Title:		Advanced Machine Learning and Deep Learning	
2nd Semester		AY 2024-2025	
<u>ACTIVITY NO. 3</u>		Basic I/O Scripting, Part 2	
Name		Apuyan, Viktor Angelo	
Section		CPE32S3	
Date Performed:		February 20, 2025	
Date Submitted:		February 21, 2025	
Instructor:		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:

```
In [1]: 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:

```
In [2]: 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:

```
In [3]: """

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

```
frame1 = cameraCapture1.retrieve()
```

```
"""
```

```
Out[3]: '\n\nsuccess0 = cameraCapture0.grab()\n\nsuccess1 = cameraCapture1.grab()\n\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:

```
In [4]: import cv2
import numpy as np

img = cv2.imread('spidey.png')
cv2.imshow('spidey.png', 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:

```
In [5]: 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.destroyAllWindows()
cameraCapture.release()

```

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

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. Try reading and writing a video file in various formats

```
In [6]: #reading/writing mp4
import cv2

videoCapture = cv2.VideoCapture('Your Friendly Neighborhood SpiderMan Clip-New Suit

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.mp4', 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()
```

```
In [9]: #reading mp4 / writing mkv
import cv2

videoCapture = cv2.VideoCapture('Your Friendly Neighborhood SpiderMan Clip-New Suit

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.mkv', 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()

```

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.

```

In [8]: # Load the image
img = cv2.imread('amazingspiderman.png')

# Check if the image was loaded successfully
if img is None:
    print("Error: Could not open or read the image.")
else:
    # Display the original image
    cv2.imshow('amazingspiderman.png', img)
    cv2.waitKey(0)

    # Define the slicing coordinates for the face (adjust these values as needed)
    x_start = 5
    y_start = 10
    width = 800
    height = 700 # Replace with the height of the face

    # Slice the image to get the face
    face = img[y_start:y_start + height, x_start:x_start + width]

    # Display the sliced face
    cv2.imshow('Face', face)
    cv2.waitKey(0)

    # Clean up windows
    cv2.destroyAllWindows()

```

3. Capture video from your webcam and display on a window. Afterwards, the video should be written as a new file.

```

In [12]: # Initialize camera capture
cameraCapture = cv2.VideoCapture(0)

# Get camera properties
fps = 30 # Assuming 30 fps
size = (int(cameraCapture.get(cv2.CAP_PROP_FRAME_WIDTH)),
        int(cameraCapture.get(cv2.CAP_PROP_FRAME_HEIGHT)))

# Create VideoWriter object

```

```

videoWriter = cv2.VideoWriter(
    'MyOutputVid.avi', cv2.VideoWriter_fourcc('M', 'J', 'P', 'G'),
    fps, size)

# Check if camera opened successfully
if not cameraCapture.isOpened():
    print("Error opening camera")
    exit()

# Create a named window
cv2.namedWindow("Camera Feed", cv2.WINDOW_NORMAL) # Use WINDOW_NORMAL for resizable

# Read and display frames until 'q' is pressed
success, frame = cameraCapture.read()
while success:
    cv2.imshow("Camera Feed", frame)
    videoWriter.write(frame)

    # Break the loop if 'q' key is pressed
    key = cv2.waitKey(1)
    if key == ord('q'):
        break
    success, frame = cameraCapture.read()

# Release resources
cameraCapture.release()
videoWriter.release()
cv2.destroyAllWindows()

```

5. Summary, Conclusions and Lessons Learned

In this activity, I learned how to read video files and live cameras using OpenCV. I learned the VideoCapture class for reading frames from the video file and that VideoCapture(0) means that it uses my default webcam as live feed. Using the VideoWriter class, I wrote those frames into a new video file with a certain codec and format. I explored capturing live camera frames by getting access to a system's webcam with VideoCapture(0) and reading in a loop. I put the live video feed into the OpenCV window using cv2.imshow(), which worked to provide a way of live viewing, in addition to implementing controls for the user to close the window or stop a video stream. I also learned a great deal about some important functions, such as waitKey — which stops the program to wait for a keyboard event — enabling frame-by-frame control or exit commands. I came to learn about destroyAllWindows, which ensures to close all OpenCV windows safely. Furthermore, I applied cv2.VideoWriter_fourcc to set the codec to encode the video format successfully. Besides that, I studied mouse callbacks like EVENT_LBUTTONDOWN, EVENT_RBUTTONDOWN, and EVENT_MOUSEMOVE, which permit interactivity, like drawing on frames, detecting right-clicks, or following mouse movement over images.

Learning video I/O and real-time camera processing using OpenCV was facilitated by this activity. Insight into how video processing applications work came when I learned how to read, write, and display frames in a video, as well as how to handle user interaction via keyboard and mouse events. This knowledge is important for building advanced projects leveraging computer vision, such as video editors, object tracking, or real-time surveillance. It enhanced my skills in handling multimedia data and building interactive applications.

Proprietary Clause

Property of the Technological Institute of the Philippines (T.I.P.). No part of the materials made and uploaded in this learning management system by T.I.P. may be copied, photographed, printed, reproduced, shared, transmitted, translated, or reduced to any electronic medium or machine-readable form, in whole or in part, without the prior consent of T.I.P.