OpenCV Tutorial

This tutorial assumes that you are using an Debion-based distribution such as Debian Stable or Ubuntu.

Installing OpenCV:

Using your built-in package manager:

First of all, open the terminal on your computer and install the OpenCV C++ library using the following command.

```
sudo apt-get install libopencv-dev python3-opencv
```

From Source:

If you'd like the most recent release of OpenCV, you can install the library from source using the following link:
https://docs.opencv.org/4.x/d7/d9f/tutorial_linux_install.html

Creating your first Program:

The following program will help you learn the ropes of the OpenCV library and get used to some of the common functions.

Using your favorite code editor, create a cpp file, import the necessary libraries, and create a basic main function such as the one below.

```
#include <opencv2/opencv.hpp> //make sure to import the opencv library


#include <iostream>


int main(int argc, char *argv[])
{
    //Our program will go here
    return 0;
}
```

First of all we need to create an output window where are video stream will be displayed. OpenCV has a *namedWindow* function that creates an output window, meaning our first line will look like:

```
cv::namedWindow(“<Window Name>”); //window name is taken as a C-string
(char *)
```

Now, we need to provide OpenCV with a video capture stream to utilize. In this tutorial, a basic mp4 video file will be used. If you need a simple test file, the following site has a variety of free stock videos at a variety of resolutions:
https://www.pexels.com

Once you have your file, we will use this constructor to load in our capture:

```
cv::VideoCapture cap("<Video File Name>"); //video name is taken as a C-
String (char *)
```

We now have an object “cap” that represents a video stream.

In order to do anything with this stream though, we need to isolate individual frames to do any processing. OpenCV represents these frames as a “Mat” or effectively a pixel matrix.

Lets make a pixel matrix object and set it to load in the next frame from our video stream:

```
cv::Mat curFrame;
cap >> curFrame; //load in next frame
```

Now that curFrame has data from our capture, we can do any necessary image processing on it. But once done, we show it like this:

```
cv::imshow(“<Window Name>”, curFrame); //this function displays “curFrame”
in the window defined in the first argument
//Make sure your window name matches the name we gave our window earlier!
```

What we have now, will only display the first frame from the capture, so to continually display the entire stream, we wrap these functions in a while loop:

```
while(true)
{
    cap >> curFrame;
```

```
    if (curFrame.empty())
    {
        break;
    }
    cv::imshow(argv[2], sobelFrame);
    cv::waitKey(1); //required to wait 1ms before fetching the next frame
}
```

The only two functions not previously mentioned that were needed to accomplish this were:
curFrame.empty() function: returns true when the capture fetches an empty frame
waitKey() function: waits for time provided in the arguments (in our case 1ms)

Once finished with the loop, make sure to release our open capture and close the output window:

```
cap.release();
// Close all OpenCV windows
cv::destroyAllWindows();
```

Our completed program should look like this:

```
#include <opencv2/opencv.hpp>
#include <iostream>

int main(int argc, char *argv[])
{
    cv::namedWindow("<Window Name>"); //window name is taken as a C-string
(char *)
    cv::VideoCapture cap("<Video File Name>"); //video name
    cv::Mat curFrame;
    while(true)
    {
        cap >> curFrame;
        if (curFrame.empty())
        {
            break;
        }
        cv::imshow("<Window Name>", curFrame);
```

```
        cv::waitKey(1); //required to wait 1ms before fetching the next
frame
    }
    cap.release();
    // Close all OpenCV windows
    cv::destroyAllWindows();
    return 0;
}
```

Building the Program:

For the sake of simplicity, we will provide a simple make file that will appropriately link the compiler to the necessary libraries and header files needed to build the program

```
CC := g++ # name of the compiler to use
OUTPUT := tutorial # name of the output file
LDFLAGS := -o $(OUTPUT) -lopencv_core -lopencv_videoio -lopencv_highgui #
linker flags (-o is optional)
CFLAGS := -g -Wall -Wpedantic -std=c++11 -I/usr/include/opencv4 -o0 #
build flags


FILES := # add additional build files here
MAIN_FILE := tutorial.cpp # name of the file containing 'main' function


default: all # default build rule


all: # main build rule
     $(CC) $(CFLAGS) $(FILES) $(MAIN_FILE) $(LDFLAGS)


clean: # clears output files
     rm -rf *.o *.so $(OUTPUT)
```

Notice the following changes from a basic makefile:

LDFLAGS: The addition of -lopencv_core -lopencv_videoio -lopencv_highgui being linked
CFLAGS: The addition of the -I/usr/include/opencv4 being added as an include directory

These two additions ensure the compiler links the appropriate libraries and also is aware of the OpenCV header files used.

Now just build and run the program!

```
make
./tutorial
```

A window should popup and begin playing your video stream