RTES Exercise 4 Report
By: Suhas Srinivasa Reddy and John Little

1) [10 points] Warning: START EARLY. Obtain a Logitech C200 or C270 camera or equivalent and verify that is detected by the DE1-SoC, Raspberry Pi or Jetson Board USB driver. You can check the camera out from eStore or purchase one of your own or use another camera that has a compliant UVC driver. Use lsusb, lsmod and dmesg kernel driver configuration tool to make sure your Logitech C2xx USB camera is plugged in and recognized by your DE1-SoC, Raspberry Pi or Jetson (note that on the Jetson, it does not use driver modules, but rather a monolithic kernel image, so lsmod will not look the same as other Linux systems – see what you can find by exploring /cat/proc on you Jetson to find the camera USB device). For the Jetson, do lsusb | grep C200 (or C270) and prove to the TA (and more importantly yourself) with that output (screenshot) that your camera is recognized. For systems other than a Jetson, do lsmod | grep video and verify that the UVC driver is loaded as well (http://www.ideasonboard.org/uvc/). To further verify, or debug if you don't see the UVC driver loaded in response to plugging in the USB camera, do dmesg | grep video or just dmesg and scroll through the log messages to see if your USB device was found. Capture all output and annotate what you see with descriptions to the best of your understanding.

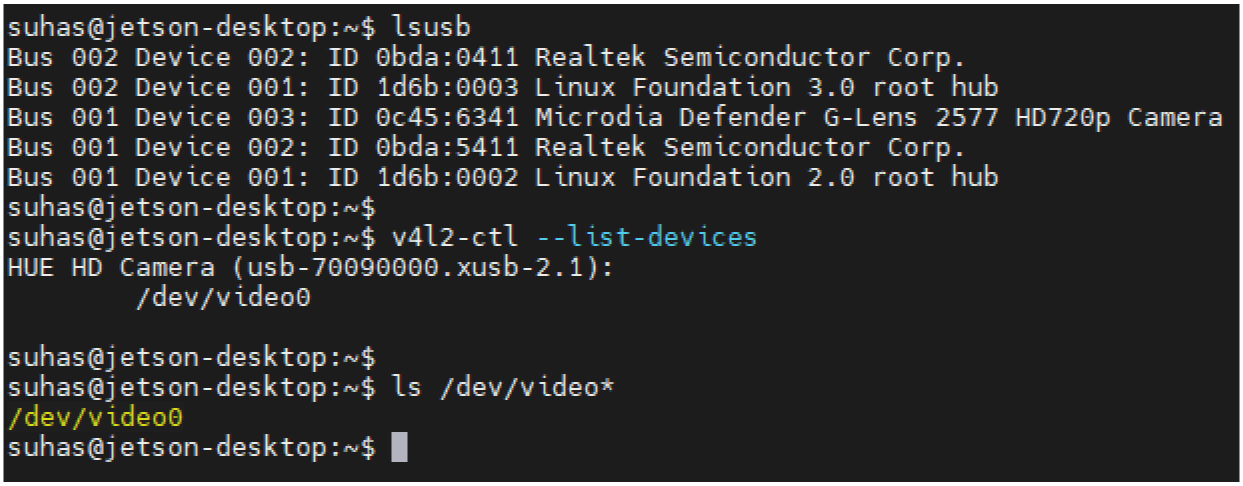
Answer:

After connecting the USB Webcam to the Jetson Nano, the Jetson identifies the device class and its various pipes with the UVC (usb video class) driver, and makes the device video stream available as an attached device (at /dev/video0 in our case).

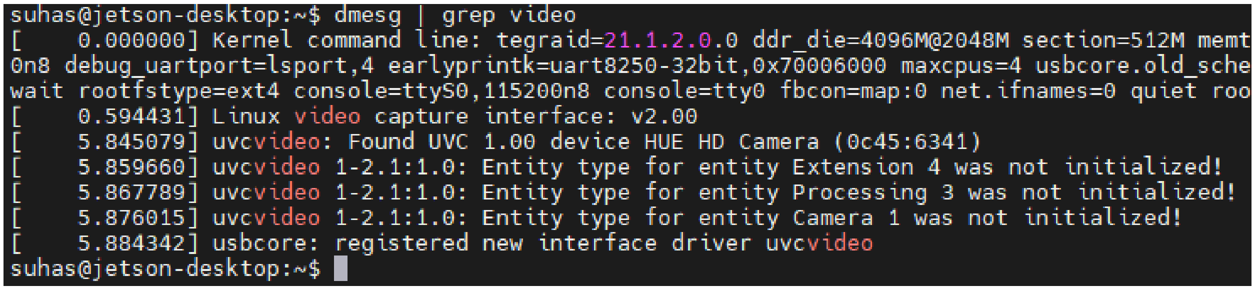
Brief: The "lsusb" command shows that the camera is connected via "Bus 001 Device 003 shows that the camera is connected." The command "V4l2-ctl –list-devices" further shows that the camera connect is HUE HD. The "ls /dev/video* " lists all the video devices. The "dmesg | grep video" shows the HUE HD camera was initialized.

Commands executed:

- lsusb
- V4l2-ctl –list-devices
- ls /dev/video*
- dmesg | grep video



```
suhas@jetson-desktop:~$ lsusb
Bus 002 Device 002: ID 0bda:0411 Realtek Semiconductor Corp.
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 003: ID 0c45:6341 Microdia Defender G-Lens 2577 HD720p Camera
Bus 001 Device 002: ID 0bda:5411 Realtek Semiconductor Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
suhas@jetson-desktop:~$
suhas@jetson-desktop:~$ v4l2-ctl --list-devices
HUE HD Camera (usb-70090000.xusb-2.1):
        /dev/video0

suhas@jetson-desktop:~$
suhas@jetson-desktop:~$ ls /dev/video*
/dev/video0
suhas@jetson-desktop:~$
```



```
suhas@jetson-desktop:~$ dmesg | grep video
[    0.000000] Kernel command line: tegraid=21.1.2.0.0 ddr_die=4096M@2048M section=512M memt
0n8 debug_uartport=lsport,4 earlyprintk=uart8250-32bit,0x70006000 maxcpus=4 usbcore.old_sche
wait rootfstype=ext4 console=ttyS0,115200n8 console=tty0 fbcon=map:0 net.ifnames=0 quiet roo
[    0.594431] Linux video capture interface: v2.00
[    5.845079] uvcvideo: Found UVC 1.00 device HUE HD Camera (0c45:6341)
[    5.859660] uvcvideo 1-2.1:1.0: Entity type for entity Extension 4 was not initialized!
[    5.867789] uvcvideo 1-2.1:1.0: Entity type for entity Processing 3 was not initialized!
[    5.876015] uvcvideo 1-2.1:1.0: Entity type for entity Camera 1 was not initialized!
[    5.884342] usbcore: registered new interface driver uvcvideo
suhas@jetson-desktop:~$
```

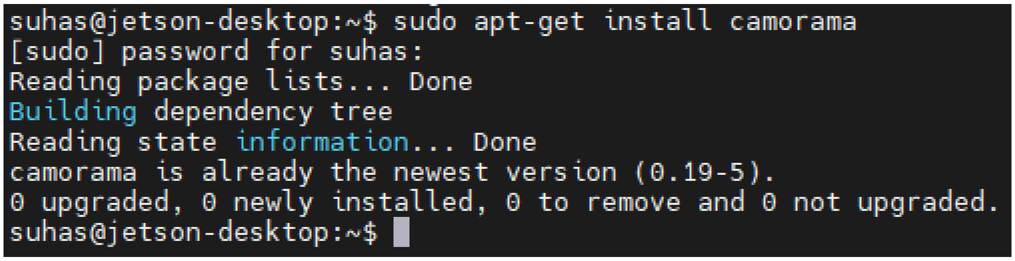
2) [10 points] Camera Tools – You only need to do either Option 1 or Option 2.

Option 1: Camorama If you do not have camorama, do apt-get install camorama on your DE1-SoC, Raspberry Pi, or Jetson board [you may need to first do sudo add-apt-repository universe; sudo apt-get update]. This should not only install nice camera capture GUI tools, but also the V4L2 API (described well in this series of Linux articles - http://lwn.net/Articles/203924/). Running camorama should provide an interactive camera control session for your Logitech C2xx camera – if you have issues connecting to your camera do a "man camorama" and specify your camera device file entry point (e.g. /dev/video0). Run camorama and play with Hue, Color, Brightness, White Balance and Contrast, take an example image and take a screen shot of the tool and provide both in your report. If camorama doesn't appear to work, then consider the next option, Cheese.

Answer:

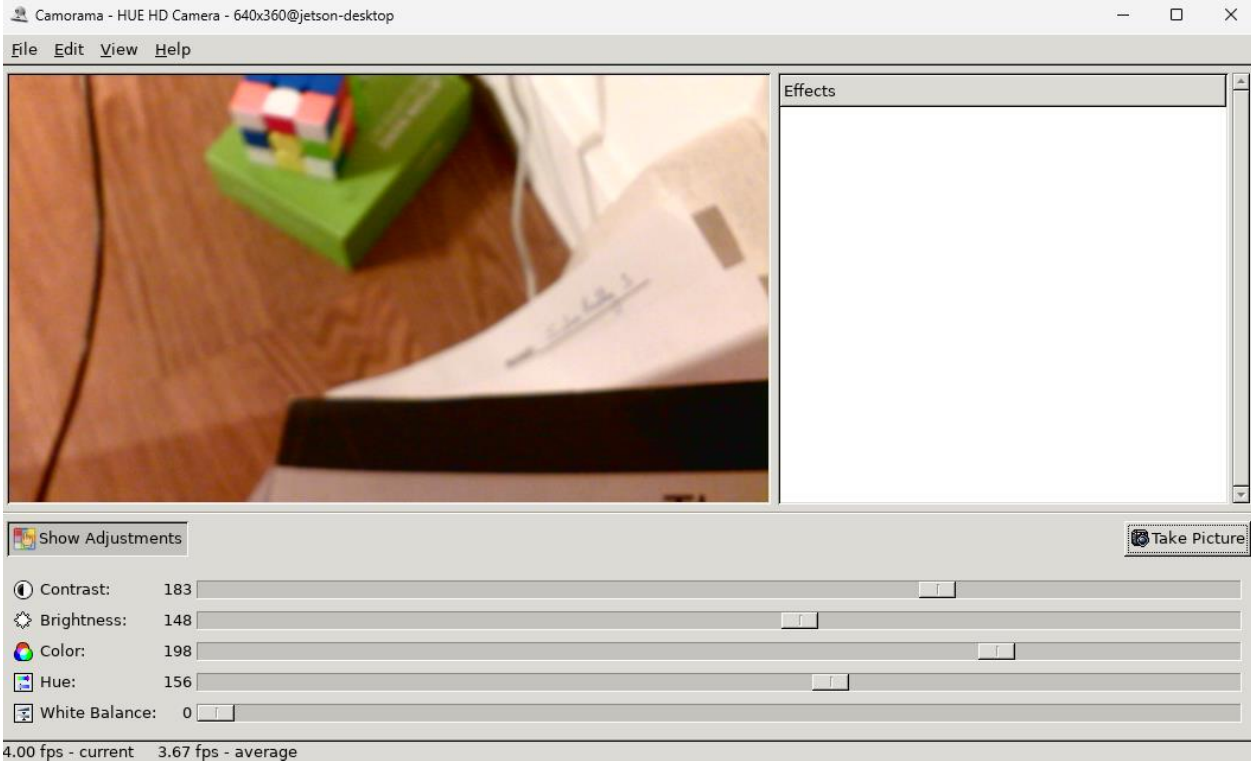
The following images show installation and use of the camorama tool.

Camorama installation:



```
suhas@jetson-desktop:~$ sudo apt-get install camorama
[sudo] password for suhas:
Reading package lists... Done
Building dependency tree
Reading state information... Done
camorama is already the newest version (0.19-5).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
suhas@jetson-desktop:~$ 
```

Camorama Console:



```
Contrast:       183
Brightness:     148
Color:          198
Hue:            156
White Balance:    0
4.00 fps - current    3.67 fps - average
```

Image before adjustment:

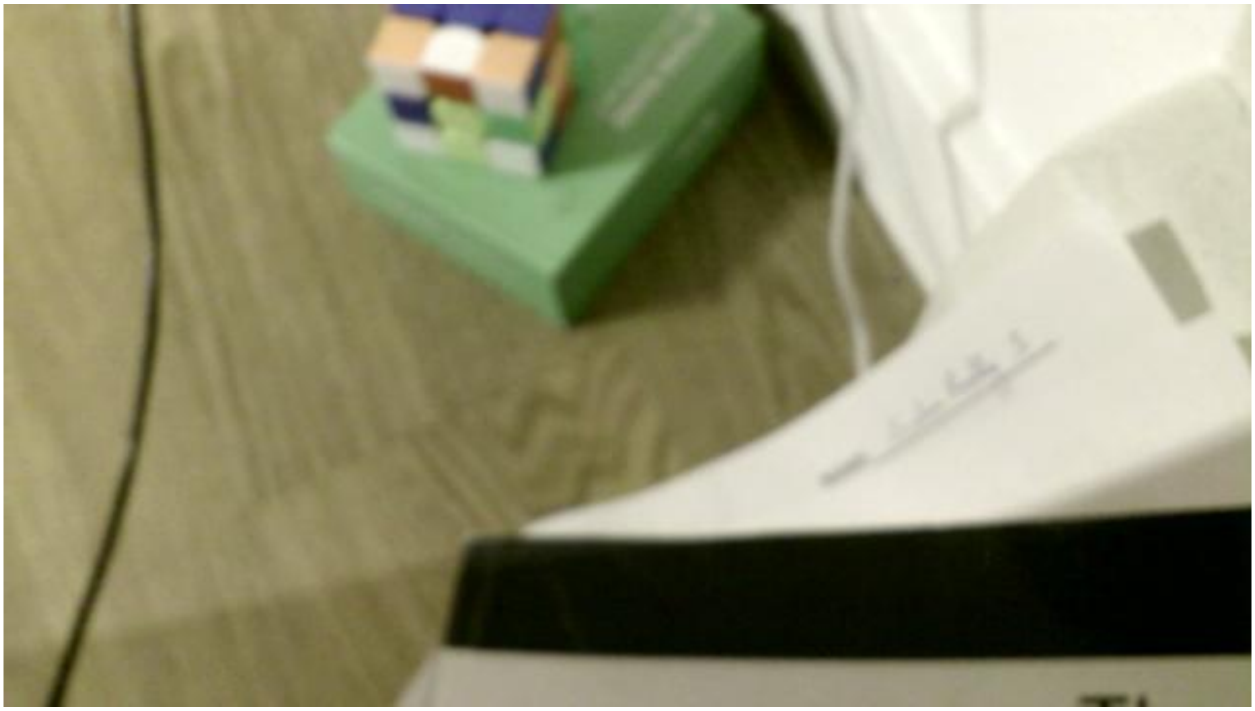
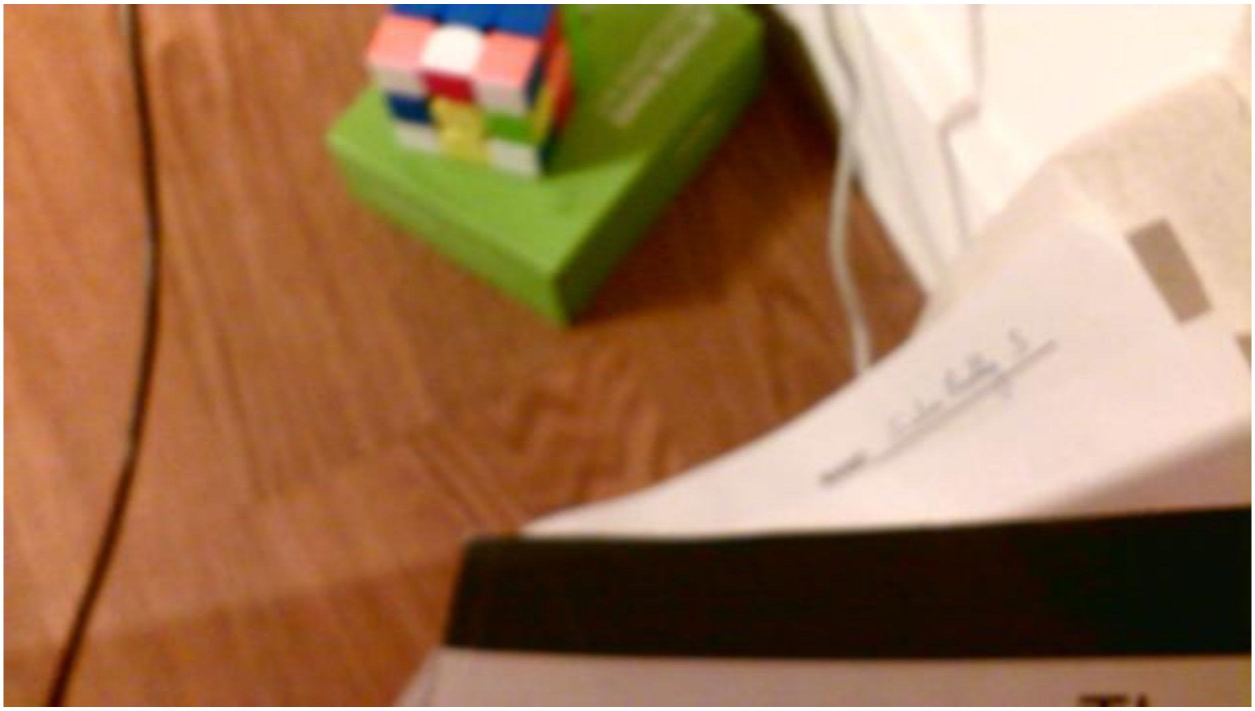
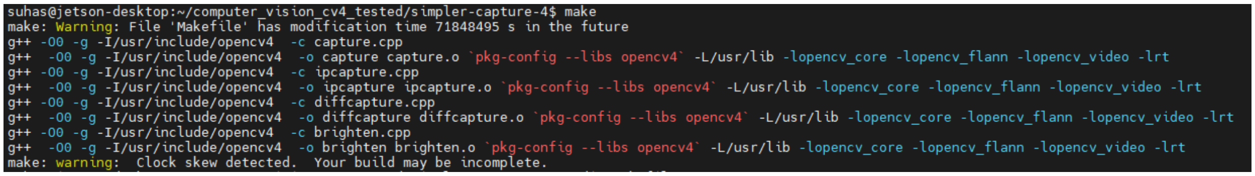


Image after adjustment:



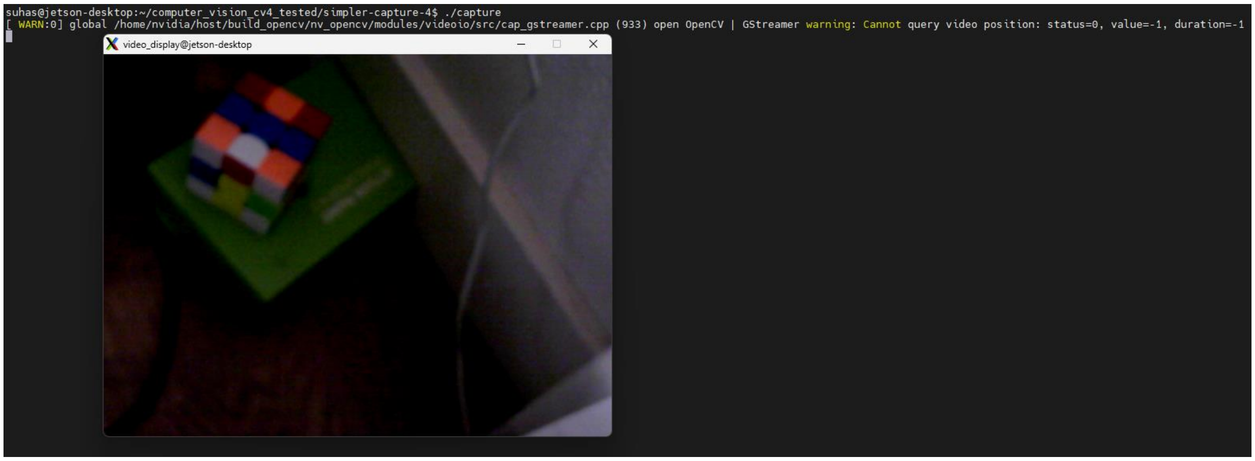
3) [10 points] Using your verified Logitech C2xx camera on a DE1-SoC, Raspberry Pi or Jetson, verify that it can stream continuously to a raw image buffer for transformation and processing using any one of the below methods, either a) or b):
a) Use example code from the computer-vision or computer_vision_cv3_tested folder such as simple-capture, simpler-capture, or simpler-capture-2. Read the code and modify the device that is opened if necessary to get this to work. Provide a screen shot to prove that you got continuous capture to work. Note that simpler capture requires installation of OpenCV on your DE1-SoC, Raspberry Pi, Jetson, or native Linux system. For the Jetson Nano or TK1 OpenCV will likely already be available on your board, but if not, please follow simple instructions found here to install openCV [the "Option 2, Building the public OpenCV library from source" is the recommended approach with – DWITH_CUDA=OFF. Don't install CUDA and please leave it off when you build OpenCV.] For the DE1-SoC please find the files soc_system.rbf and SettingUp.pdf on Canvas. They contain instructions for setting up board and installing OpenCV. The TAs have set up using these in the past and were able to complete exercise 4 requirements. The cmake command for opencv installation has been changed so that it works on the board too. Alternatively, you can use OpenCV port found here: http://rocketboards.org/foswiki/view/Projects/OpenCVPort. If you have trouble getting OpenCV to work on your board, try running it on your laptop under Linux first. You can use OpenCV install, OpenCV with Python bindings for 2.x and 3.x for this.
Answer:

Simpler_Capture-4 Built:



```
suhas@jetson-desktop:~/computer_vision_cv4_tested/simpler-capture-4$ make
make: Warning: File 'Makefile' has modification time 71848495 s in the future
g++ -O0 -g -I/usr/include/opencv4  -c capture.cpp
g++  -O0 -g -I/usr/include/opencv4  -o capture capture.o `pkg-config --libs opencv4` -L/usr/lib -lopencv_core -lopencv_flann -lopencv_video -lrt
g++ -O0 -g -I/usr/include/opencv4  -c ipcapture.cpp
g++  -O0 -g -I/usr/include/opencv4  -o ipcapture ipcapture.o `pkg-config --libs opencv4` -L/usr/lib -lopencv_core -lopencv_flann -lopencv_video -lrt
g++ -O0 -g -I/usr/include/opencv4  -c diffcapture.cpp
g++  -O0 -g -I/usr/include/opencv4  -o diffcapture diffcapture.o `pkg-config --libs opencv4` -L/usr/lib -lopencv_core -lopencv_flann -lopencv_video -lrt
g++ -O0 -g -I/usr/include/opencv4  -c brighten.cpp
g++  -O0 -g -I/usr/include/opencv4  -o brighten brighten.o `pkg-config --libs opencv4` -L/usr/lib -lopencv_core -lopencv_flann -lopencv_video -lrt
make: warning:  Clock skew detected.  Your build may be incomplete.
```

Capture.cpp

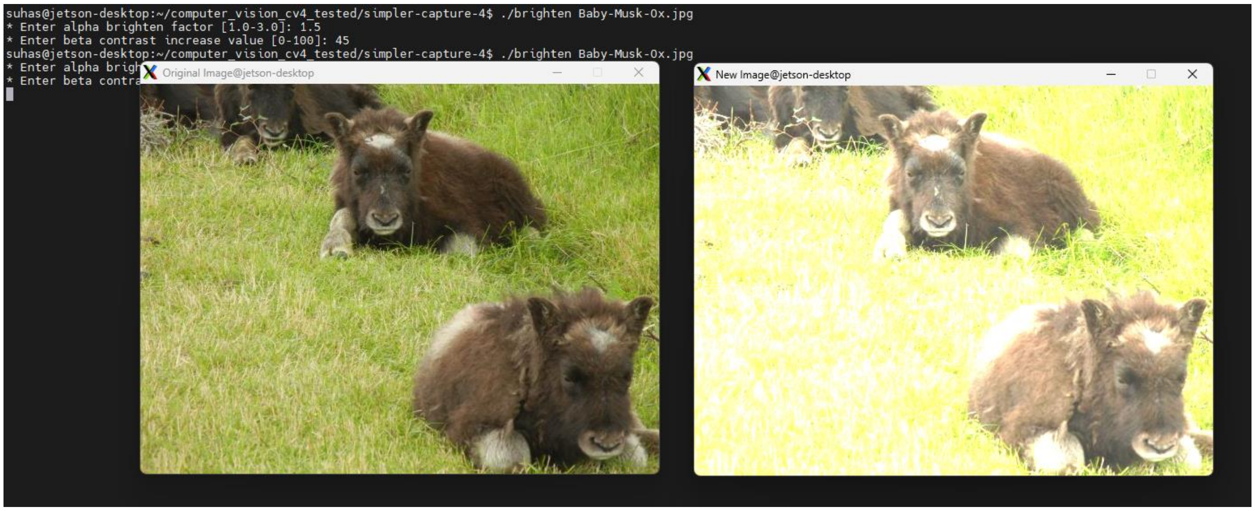


```
suhas@jetson-desktop:~/computer_vision_cv4_tested/simpler-capture-4$ ./capture
[ WARN:0] global /home/nvidia/host/build_opencv/nv_opencv/modules/videoio/src/cap_gstreamer.cpp (933) open OpenCV | GStreamer warning: Cannot query video position: status=0, value=-1, duration=-1
```

Code Brief:

The capture.cpp code starts by assigning a VideoCapture object, cam0 to dev/video0, and uses cam0.isOpened to check if the device is already in use. It is opened and initialized with a frame size, and then enters a loop to continuously read the video stream using cam0.read, and show the image using cam0.show. The graphical interface is initialized with a resolution and window to display the live stream. The waitKey(10) API checks for keyboard input and breaks the loop if the Escape key is clicked. On exiting the code, it destroys the streaming window using destroyWindow API.

Brighten.cpp (modified)

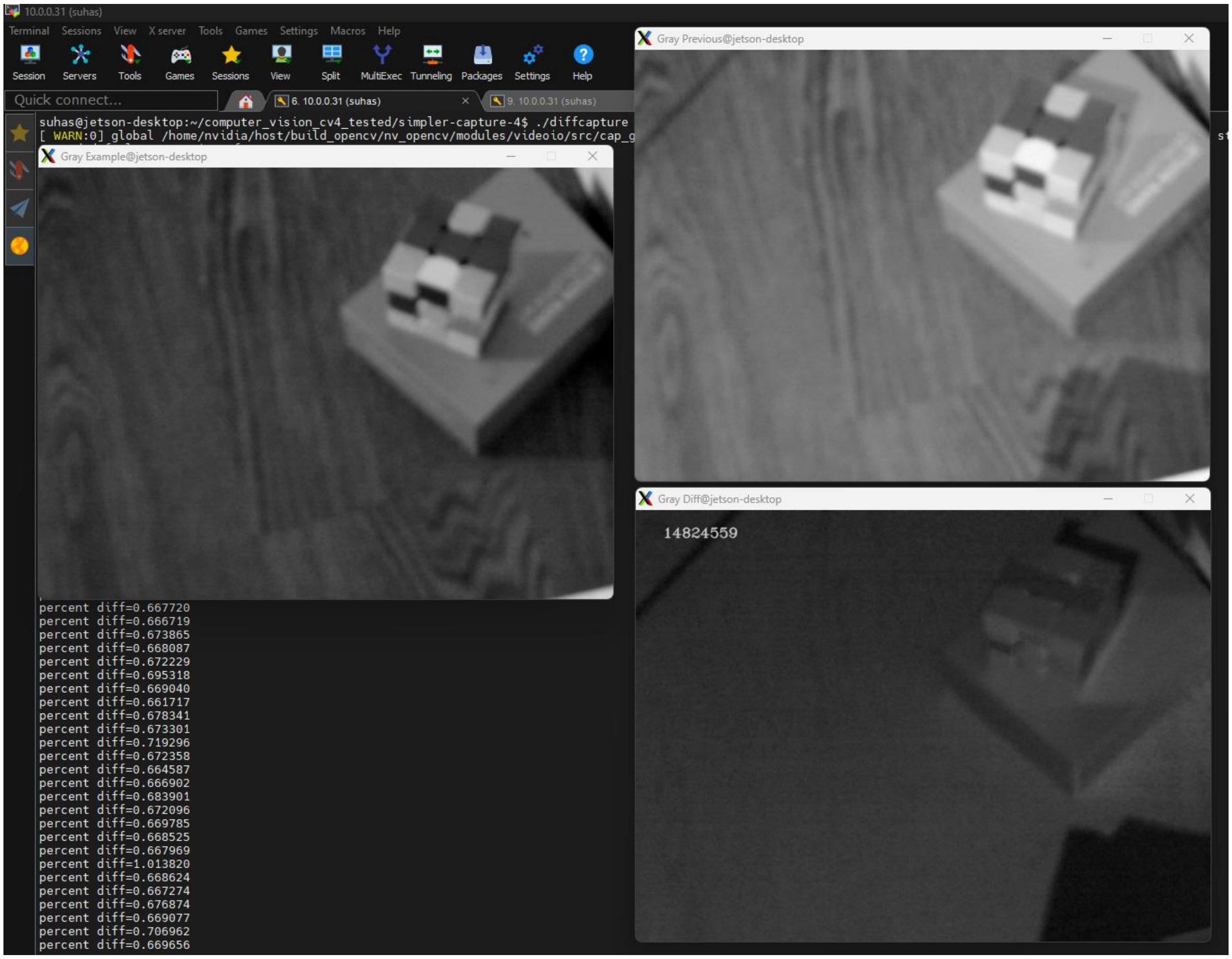


```
suhas@jetson-desktop:~/computer_vision_cv4_tested/simpler-capture-4$ ./brighten Baby-Musk-Ox.jpg
* Enter alpha brighten factor [1.0-3.0]: 1.5
* Enter beta contrast increase value [0-100]: 45
suhas@jetson-desktop:~/computer_vision_cv4_tested/simpler-capture-4$ ./brighten Baby-Musk-Ox.jpg
* Enter alpha brigh
* Enter beta contra
```

Code Brief:

The code takes an image as an input which is read and loaded. It further takes alpha brighten factor and beta contrast factor. These factors are used scale RGB pixel values in every row and column.

Diffcapture.cpp



10.0.0.31 (suhas)
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help
Quick connect...
6. 10.0.0.31 (suhas)
9. 10.0.0.31 (suhas)
Gray Example@jetson-desktop
Gray Previous@jetson-desktop
Gray Diff@jetson-desktop
14824559

```
suhas@jetson-desktop:~/computer_vision_cv4_tested/simpler-capture-4$ ./diffcapture
[ WARN:0] global /home/nvidia/host/build_opencv/nv_opencv/modules/videoio/src/cap_g
```

```
percent diff=0.667720
percent diff=0.666719
percent diff=0.673865
percent diff=0.668087
percent diff=0.672229
percent diff=0.695318
percent diff=0.669040
percent diff=0.661717
percent diff=0.678341
percent diff=0.673301
percent diff=0.719296
percent diff=0.672358
percent diff=0.664587
percent diff=0.666902
percent diff=0.683901
percent diff=0.672096
percent diff=0.669785
percent diff=0.668525
percent diff=0.667969
percent diff=1.013820
percent diff=0.668624
percent diff=0.667274
percent diff=0.676874
percent diff=0.669077
percent diff=0.706962
percent diff=0.669656
```

Code

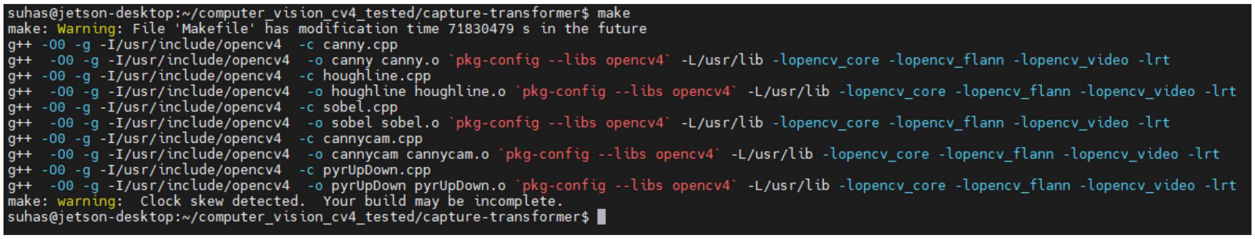
The DiffCapture algorithm continuously captures each image and compares it to the previous one. The comparison is made by measuring the greyscale brightness value, pixel by pixel, and calculating the absolute value of the difference between the previous image's value. A new image is constructed with the absolute value of the difference in brightness, and the percent diff metric calculates and displays the overall difference – this is equal to the average of all difference values divided by the maximum possible diff value of 255.

4) [20 points]

a) Choose a continuous transformation OpenCV example from computer vision cv3 tested such as the canny-interactive, hough-interactive, hough-eliptical-interactive, or stereo-transformimpoved . Show a screen shot to prove you built and ran the code. Provide a detailed explanation of the code and research uses for the continuous transformation by looking up API functions in the OpenCV manual (http://docs.opencv.org) and for stereo-transform-improved either implement or explain how you could make this work continuously rather than snapshot only. Repeat with a second choice of these transformations.

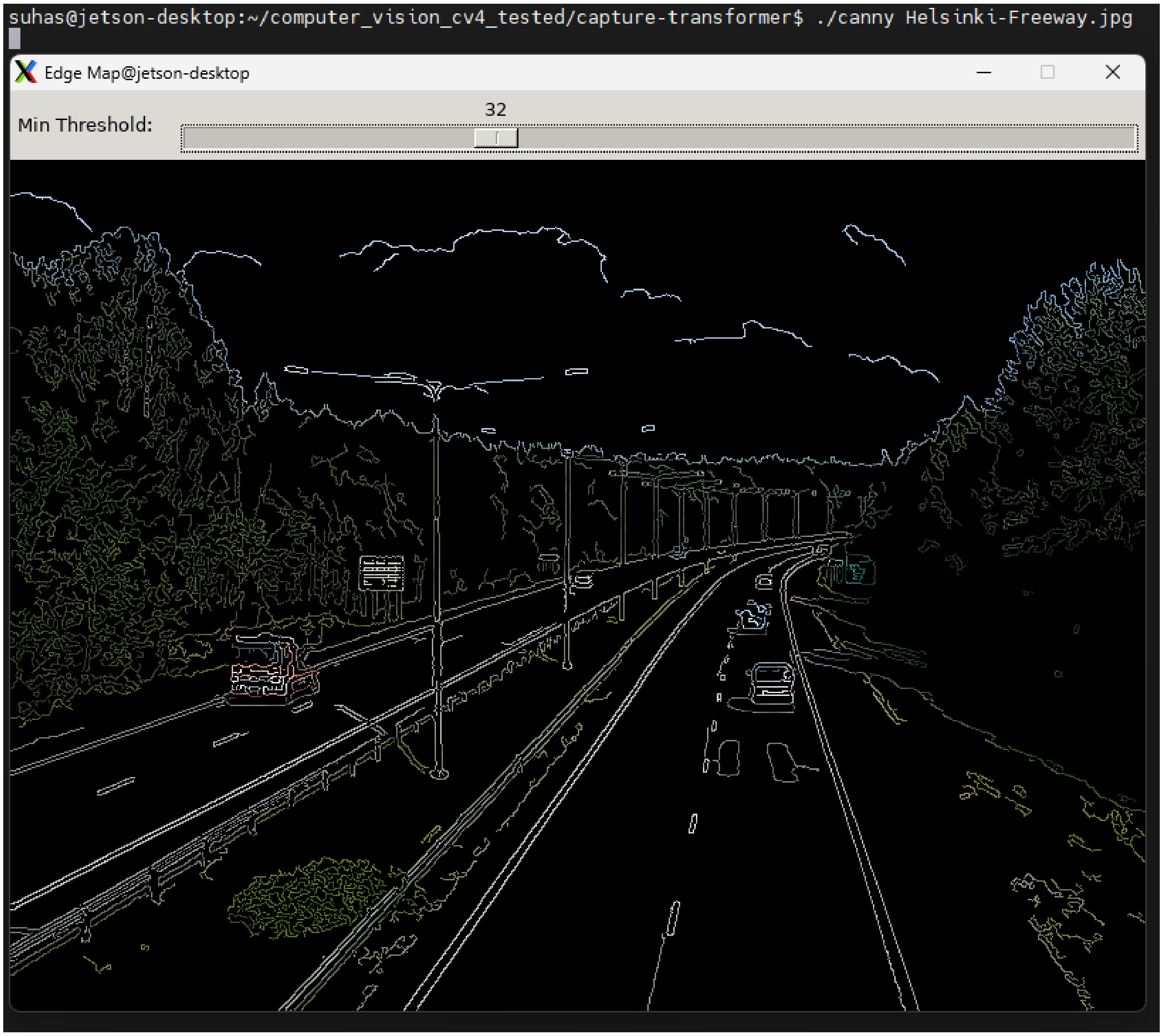
Answer:

Capture_transform built:



```
suhas@jetson-desktop:~/computer_vision_cv4_tested/capture-transformer$ make
make: Warning: File 'Makefile' has modification time 71830479 s in the future
g++ -O0 -g -I/usr/include/opencv4  -c canny.cpp
g++  -O0 -g -I/usr/include/opencv4  -o canny canny.o `pkg-config --libs opencv4` -L/usr/lib -lopencv_core -lopencv_flann -lopencv_video -lrt
g++ -O0 -g -I/usr/include/opencv4  -c houghline.cpp
g++  -O0 -g -I/usr/include/opencv4  -o houghline houghline.o `pkg-config --libs opencv4` -L/usr/lib -lopencv_core -lopencv_flann -lopencv_video -lrt
g++ -O0 -g -I/usr/include/opencv4  -c sobel.cpp
g++  -O0 -g -I/usr/include/opencv4  -o sobel sobel.o `pkg-config --libs opencv4` -L/usr/lib -lopencv_core -lopencv_flann -lopencv_video -lrt
g++ -O0 -g -I/usr/include/opencv4  -c cannycam.cpp
g++  -O0 -g -I/usr/include/opencv4  -o cannycam cannycam.o `pkg-config --libs opencv4` -L/usr/lib -lopencv_core -lopencv_flann -lopencv_video -lrt
g++ -O0 -g -I/usr/include/opencv4  -c pyrUpDown.cpp
g++  -O0 -g -I/usr/include/opencv4  -o pyrUpDown pyrUpDown.o `pkg-config --libs opencv4` -L/usr/lib -lopencv_core -lopencv_flann -lopencv_video -lrt
make: warning:  Clock skew detected.  Your build may be incomplete.
suhas@jetson-desktop:~/computer_vision_cv4_tested/capture-transformer$ █
```

Canny.cpp
Command: ./canny Helsinki-Freeway.jpg



```
suhas@jetson-desktop:~/computer_vision_cv4_tested/capture-transformer$ ./canny Helsinki-Freeway.jpg
```

Min Threshold: 32

Code Flow Explanation:

The canny.cpp performs edge detection using canny edge detection algorithm provided by openCv libraries. Canny Edge Detection algorithm is a technique used to determine the edges of objects in the image. It is robust against image noise and provides accurate edge detection and localization.

In greater detail, the Canny edge detection algorithm performs edge detection in two steps: First, the image is blurred with a 5x5 gaussian filter – this filter smooths the image by assigning each pixel to be a weighted average of it and the pixels around it in a 5x5 pixel grid. The weights of each pixel are calculated by the two-variable gaussian function – a scaled value proportional to (exp(- (x^2) + (y^2)) for x and y centered at the pixel value. The center pixel has the highest weight of all 25 pixels.

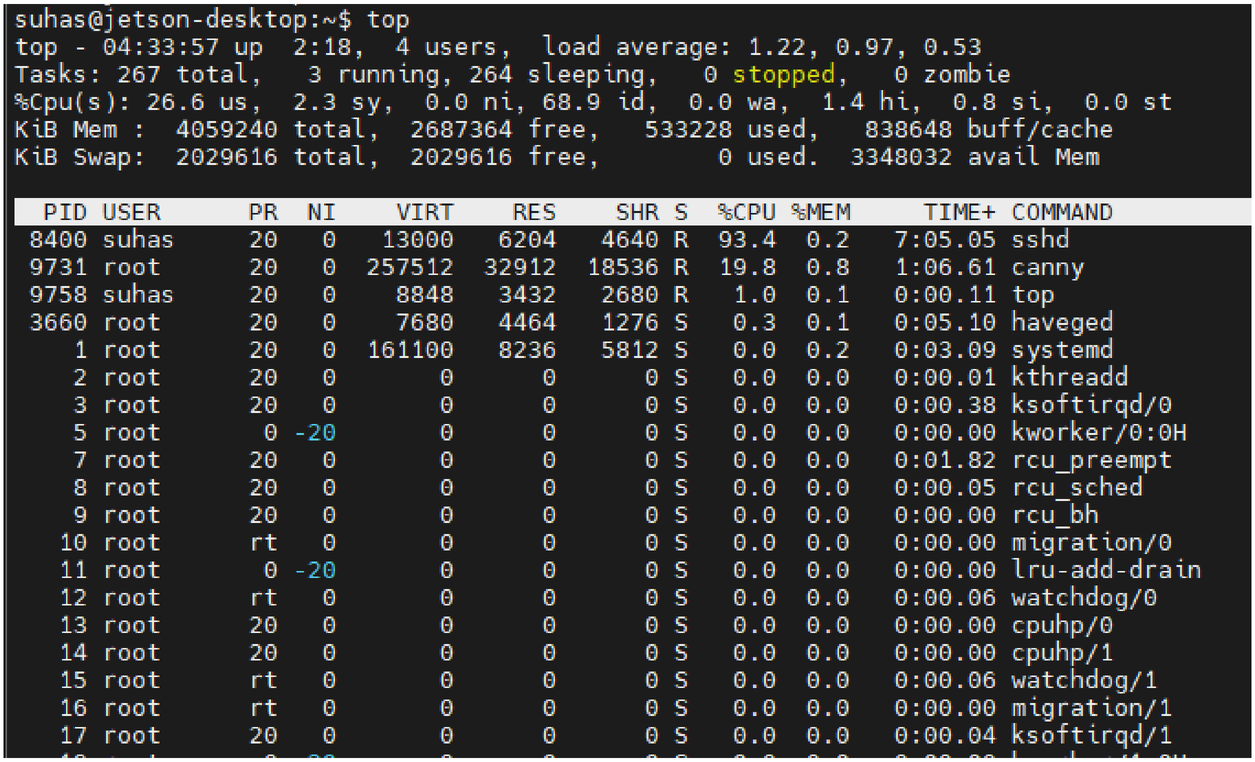
Second, the smoothed image is processed again with a gradient function. The gradient function calculates the local multivariate derivative with a Sobel kernel; this is a two-step matrix multiplication for a 3*3 matrix which calculates both the slope and direction of gradient for each pixel.

The Sobel operator is a matrix operator which is convolved with the entire image. This operator performs multivariate differentiation by first calculating the slope in the x direction and then calculating the slope in the y direction. In a simplified explanation, the x-direction slope is calculated by subtracting the values of the pixels at x – 1 from the values of the pixels at x + 1; after this value is convolved with the entire image, the same process is repeated in the vertical direction. This gives the overall gradient value at each pixel. The algorithm can be performed quickly because its convolution matrices decompose to two simple [1x3] vector multiplication steps for each direction.

Finally, gradient values are filtered to 0 unless they are at the local maxima in the direction of the gradient, and then values that are between nearby local maxima and smaller than both are filtered out once again.

<u>CPU Core Loading:</u>

When processing with the Canny edge detection algorithm with a single CPU core, the core is running with a 79% load; this is shown by the output of top.



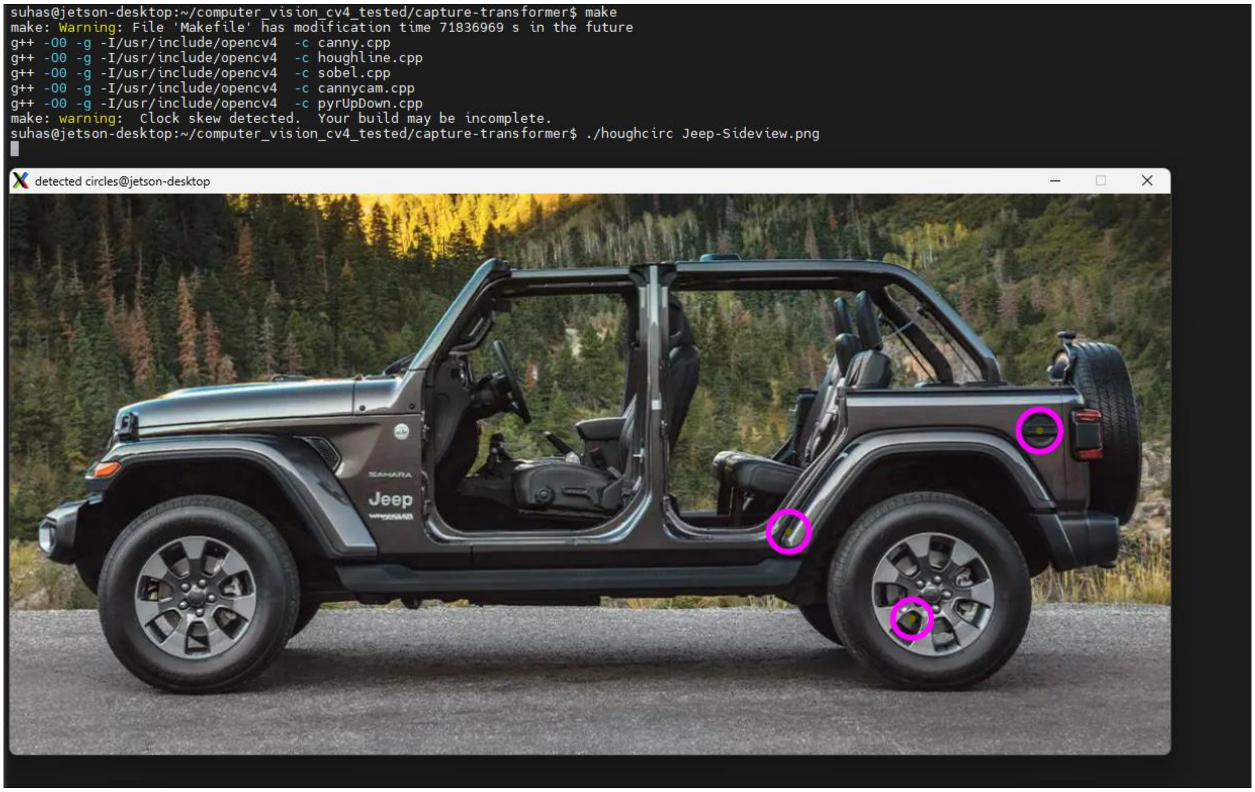
```
suhas@jetson-desktop:~$ top
top - 04:33:57 up  2:18,  4 users,  load average: 1.22, 0.97, 0.53
Tasks: 267 total,   3 running, 264 sleeping,   0 stopped,   0 zombie
%Cpu(s): 26.6 us,  2.3 sy,  0.0 ni, 68.9 id,  0.0 wa,  1.4 hi,  0.8 si,  0.0 st
KiB Mem :  4059240 total,  2687364 free,   533228 used,   838648 buff/cache
KiB Swap:  2029616 total,  2029616 free,        0 used.  3348032 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 8400 suhas     20   0   13000   6204   4640 R  93.4  0.2   7:05.05 sshd
 9731 root      20   0  257512  32912  18536 R  19.8  0.8   1:06.61 canny
 9758 suhas     20   0    8848   3432   2680 R   1.0  0.1   0:00.11 top
 3660 root      20   0    7680   4464   1276 S   0.3  0.1   0:05.10 haveged
    1 root      20   0  161100   8236   5812 S   0.0  0.2   0:03.09 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.01 kthreadd
    3 root      20   0       0      0      0 S   0.0  0.0   0:00.38 ksoftirqd/0
    5 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/0:0H
    7 root      20   0       0      0      0 S   0.0  0.0   0:01.82 rcu_preempt
    8 root      20   0       0      0      0 S   0.0  0.0   0:00.05 rcu_sched
    9 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcu_bh
   10 root      rt   0       0      0      0 S   0.0  0.0   0:00.00 migration/0
   11 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 lru-add-drain
   12 root      rt   0       0      0      0 S   0.0  0.0   0:00.06 watchdog/0
   13 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/0
   14 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/1
   15 root      rt   0       0      0      0 S   0.0  0.0   0:00.06 watchdog/1
   16 root      rt   0       0      0      0 S   0.0  0.0   0:00.00 migration/1
   17 root      20   0       0      0      0 S   0.0  0.0   0:00.04 ksoftirqd/1
```

Hough Line:

./houghline Jeep-sideview.jpg

The Hough line detection algorithm detects a likelihood of a line being present by summing log-likelihood contributions from lines that pass through all points. First off, the transform creates a distrubution of all possible lines based on their parameters m and b (or polar parameters r and theta). For each pixel that it passes through, that pixel makes a contribution proportional to the value of the gradient passing through it. (Ideally, an edge detection transform such as Canny Edge detection, described above, has already been applied). The pixel’s gradient is part of a line with some parameters m and b, and the slope of the gradient helps define the likelihood it is part of a line. The log of this likelihood for the line is added to the distribution of line parameters.

When each pixel has contributed to its most likely line, then values with total line likelihoods greater than a user-specified threshold are determined to be a straight line.



```
suhas@jetson-desktop:~/computer_vision_cv4_tested/capture-transformer$ make
make: Warning: File 'Makefile' has modification time 71836969 s in the future
g++ -O0 -g -I/usr/include/opencv4  -c canny.cpp
g++ -O0 -g -I/usr/include/opencv4  -c houghline.cpp
g++ -O0 -g -I/usr/include/opencv4  -c sobel.cpp
g++ -O0 -g -I/usr/include/opencv4  -c cannycam.cpp
g++ -O0 -g -I/usr/include/opencv4  -c pyrUpDown.cpp
make: warning:  Clock skew detected.  Your build may be incomplete.
suhas@jetson-desktop:~/computer_vision_cv4_tested/capture-transformer$ ./houghcirc Jeep-Sideview.png
```

5) [40 points] a) Using a Logitech C2xx, choose 3 real-time interactive transformations to compare in terms of average frame rate at a given resolution for a range of at least 3 resolutions

in a common aspect ratio (e.g. 4:3 for 1280x960, 640x480, 320x240, 160x120, 80x60) by adding time-stamps (there should be a logging thread) to analyze the potential throughput. You should then get at least 9 separate datasets running 1 transformation at a time. Based on average analysis, pick a reasonable soft real-time deadline (e.g. if average frame rate is 12 Hz, choose a deadline of 100 milliseconds to provide some margin) and convert the processing to SCHED_FIFO and determine if you can meet deadlines with predictability and measure statistical jitter in the frame rate relative to your deadline for both the default scheduler and SCHED_FIFO in each case.

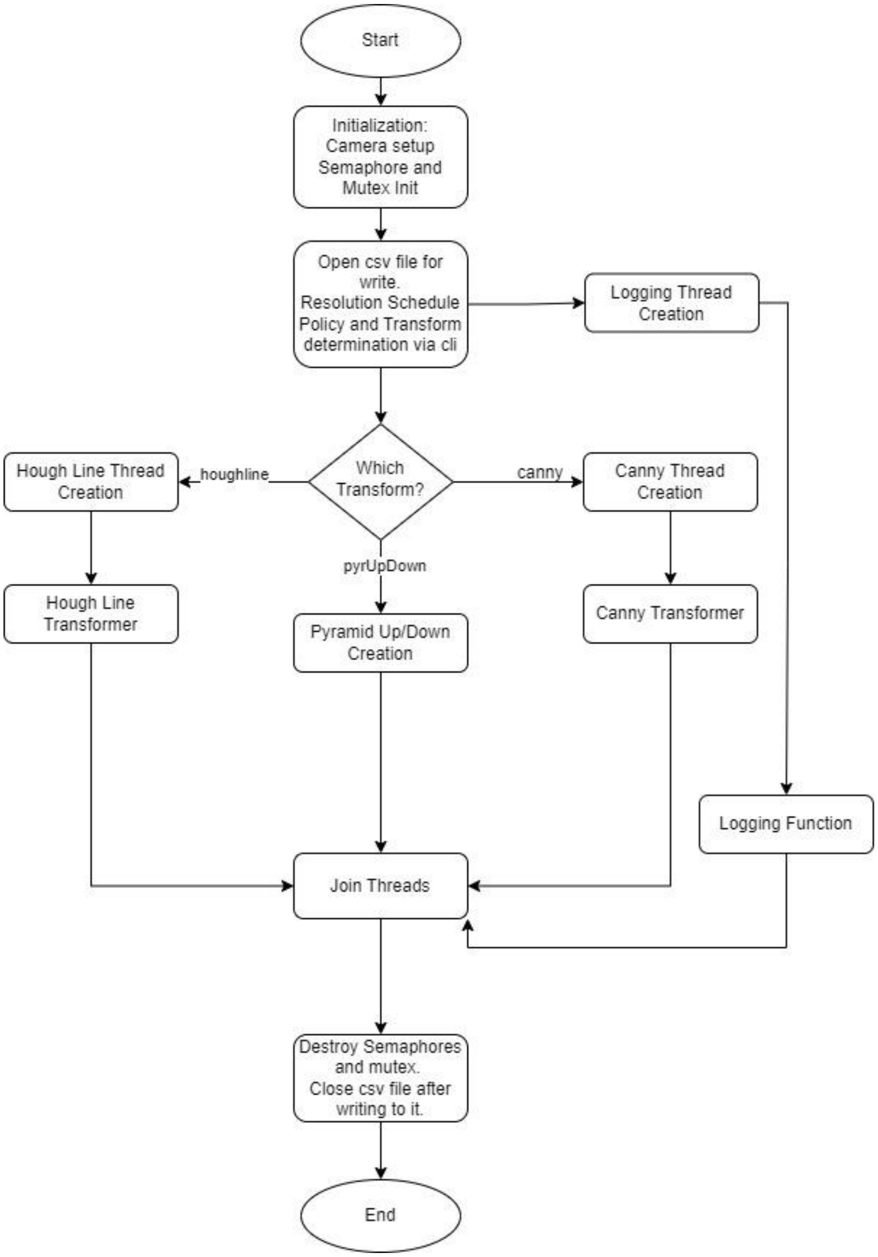
Answer:

Code Description:The code provides three transform services which can be applied over continuously streaming frames from a camera module attached to a jetson nano. The user can select the scheduling policy (defaults to SCHED_OTHER), resolution (defaults to 640x480) and transform type. After taking user inputs from the cli, an appropriate transform thread is created which is scheduled by the specified policy alongside the logging thread. The logging thread calculates the execution time required to process each frame using the given policy and logs the jitter time execution time to a csv file. Synchronization for logging accurately is achieved using semaphores where logging thread waits on a semaphore and transform thread posts the semaphore. This also makes the execution deterministic by making sure the transform thread is executed first. The code dynamically adjusts the average execution time value on the fly.

Canny Thread: This thread performs edge detection using the Canny edge detection algorithm. It first converts the captured frame to grayscale, applies a blur to reduce noise, and then uses the Canny function to detect edges. The detected edges are superimposed on the original frame, and the result is displayed in a window titled "Edge Map." This thread also records the start and end time for each frame's processing to calculate the execution time and posts a semaphore to signal the logging thread.

Hough Lines Thread: This thread detects straight lines in the captured frames. It converts the frame to grayscale, applies the Canny edge detection algorithm to find edges, and then uses the HoughLinesP function to detect line segments in the binary image. These lines are drawn on the frames, and the processed image is displayed in a window titled "Detected Lines." Like the Canny thread, it also logs the processing time for each frame.

Pyramid Up/Down Thread: This thread allows zooming in and out of the captured image using the pyramid up and down algorithms. When the user presses a key ('i' for zoom in, 'o' for zoom out), the thread alters the frame size accordingly. The pyrUp function doubles the size of the image, whereas pyrDown halves it. The modified image is then displayed in a window titled "Pyramids Demo." Like the Canny thread, it also logs the processing time for each frame.


Start
Initialization:
Camera setup
Semaphore and
Mutex Init
Open csv file for
write.
Resolution Schedule
Policy and Transform
determination via cli
Logging Thread
Creation
Which
Transform?
houghline
canny
pyrUpDown
Hough Line Thread
Creation
Canny Thread
Creation
Hough Line
Transformer
Pyramid Up/Down
Creation
Canny Transformer
Logging Function
Join Threads
Destroy Semaphores
and mutex.
Close csv file after
writing to it.
End

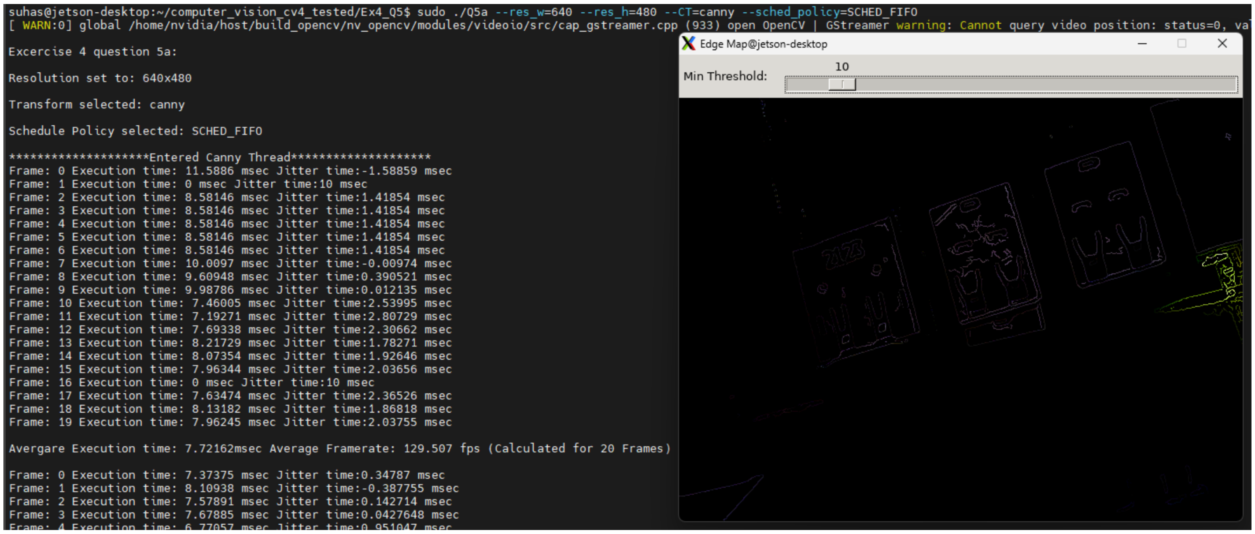
Flow chart

Table:

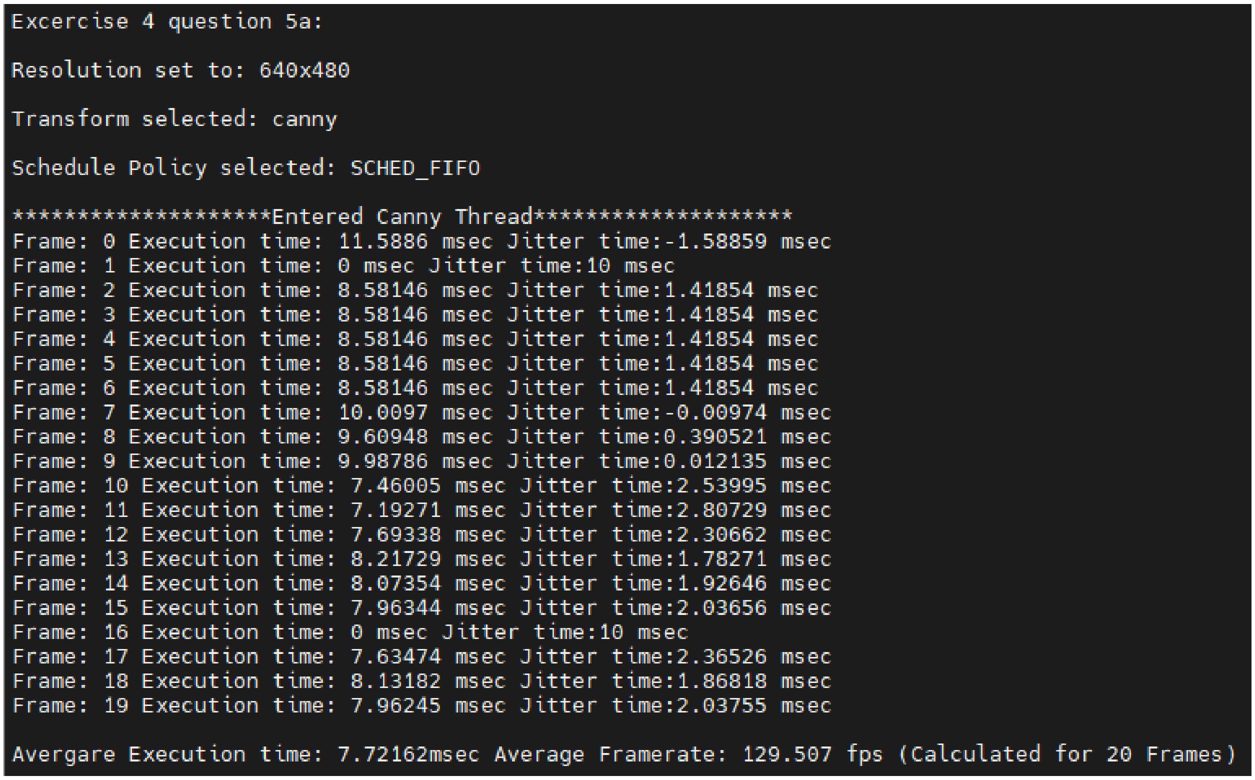
Transform	Resolution	Average Execution Time	Frame Rate	No of frames
Canny	640x480	9.81916 msec	101.842 fps	100
Canny	320x240	2.80111 msec	357.001 fps	100
Canny	160x120	0.861508 msec	1160.75 fps	100
Hough Line	640x480	8.05019 msec	124.221 fps	100
Hough Line	320x240	2.95628 msec	338.263 fps	100
Hough Line	160x120	1.3029msec	767.52 fps	100
Pyr Up/Down	640x480	9.32546 msec	107.233 fps	20
Pyr Up/Down	320x240	2.26664 msec	441.183 fps	20
Pyr Up/Down	160x120	0.809224 msec	1235.75 fps	20

Canny:

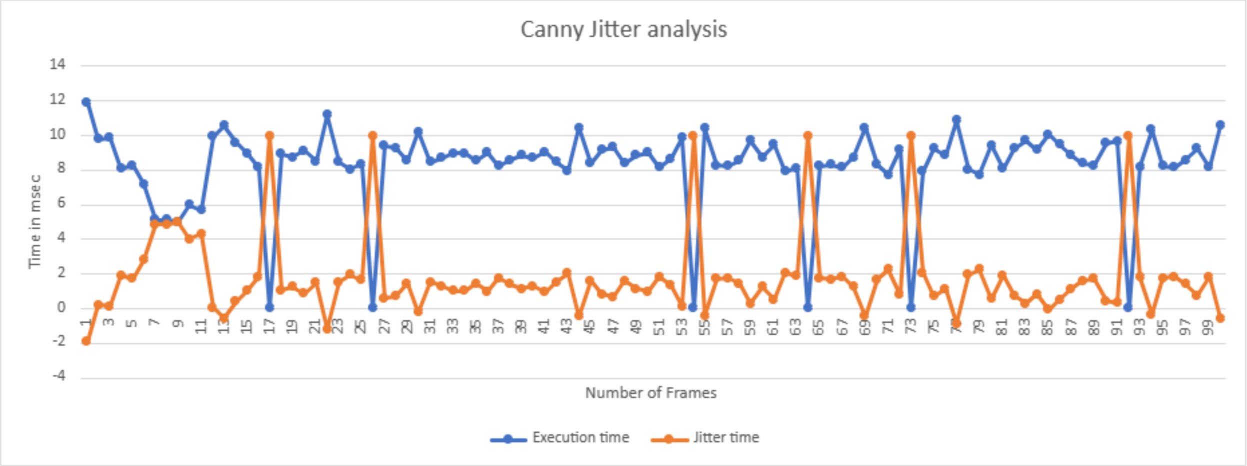
Command: sudo ./Q5a --res_w=640 –res_h=480 --CT=canny --sched_policy=SCHED_FIFO



```
suhas@jetson-desktop:~/computer_vision_cv4_tested/Ex4_Q5$ sudo ./Q5a --res_w=640 --res_h=480 --CT=canny --sched_policy=SCHED_FIFO
[ WARN:0] global /home/nvidia/host/build_opencv/nv_opencv/modules/videoio/src/cap_gstreamer.cpp (933) open OpenCV | GStreamer warning: Cannot query video position: status=0, va

Excercise 4 question 5a:

Resolution set to: 640x480

Transform selected: canny

Schedule Policy selected: SCHED_FIFO

********************Entered Canny Thread********************
Frame: 0 Execution time: 11.5886 msec Jitter time:-1.58859 msec
Frame: 1 Execution time: 0 msec Jitter time:10 msec
Frame: 2 Execution time: 8.58146 msec Jitter time:1.41854 msec
Frame: 3 Execution time: 8.58146 msec Jitter time:1.41854 msec
Frame: 4 Execution time: 8.58146 msec Jitter time:1.41854 msec
Frame: 5 Execution time: 8.58146 msec Jitter time:1.41854 msec
Frame: 6 Execution time: 8.58146 msec Jitter time:1.41854 msec
Frame: 7 Execution time: 10.0097 msec Jitter time:-0.00974 msec
Frame: 8 Execution time: 9.60948 msec Jitter time:0.390521 msec
Frame: 9 Execution time: 9.98786 msec Jitter time:0.012135 msec
Frame: 10 Execution time: 7.46005 msec Jitter time:2.53995 msec
Frame: 11 Execution time: 7.19271 msec Jitter time:2.80729 msec
Frame: 12 Execution time: 7.69338 msec Jitter time:2.30662 msec
Frame: 13 Execution time: 8.21729 msec Jitter time:1.78271 msec
Frame: 14 Execution time: 8.07354 msec Jitter time:1.92646 msec
Frame: 15 Execution time: 7.96344 msec Jitter time:2.03656 msec
Frame: 16 Execution time: 0 msec Jitter time:10 msec
Frame: 17 Execution time: 7.63474 msec Jitter time:2.36526 msec
Frame: 18 Execution time: 8.13182 msec Jitter time:1.86818 msec
Frame: 19 Execution time: 7.96245 msec Jitter time:2.03755 msec

Avergare Execution time: 7.72162msec Average Framerate: 129.507 fps (Calculated for 20 Frames)

Frame: 0 Execution time: 7.37375 msec Jitter time:0.34787 msec
Frame: 1 Execution time: 8.10938 msec Jitter time:-0.387755 msec
Frame: 2 Execution time: 7.57891 msec Jitter time:0.142714 msec
Frame: 3 Execution time: 7.67885 msec Jitter time:0.0427648 msec
Frame: 4 Execution time: 6.77057 msec Jitter time:0.951047 msec
```



```
Excercise 4 question 5a:

Resolution set to: 640x480

Transform selected: canny

Schedule Policy selected: SCHED_FIFO

********************Entered Canny Thread********************
Frame: 0 Execution time: 11.5886 msec Jitter time:-1.58859 msec
Frame: 1 Execution time: 0 msec Jitter time:10 msec
Frame: 2 Execution time: 8.58146 msec Jitter time:1.41854 msec
Frame: 3 Execution time: 8.58146 msec Jitter time:1.41854 msec
Frame: 4 Execution time: 8.58146 msec Jitter time:1.41854 msec
Frame: 5 Execution time: 8.58146 msec Jitter time:1.41854 msec
Frame: 6 Execution time: 8.58146 msec Jitter time:1.41854 msec
Frame: 7 Execution time: 10.0097 msec Jitter time:-0.00974 msec
Frame: 8 Execution time: 9.60948 msec Jitter time:0.390521 msec
Frame: 9 Execution time: 9.98786 msec Jitter time:0.012135 msec
Frame: 10 Execution time: 7.46005 msec Jitter time:2.53995 msec
Frame: 11 Execution time: 7.19271 msec Jitter time:2.80729 msec
Frame: 12 Execution time: 7.69338 msec Jitter time:2.30662 msec
Frame: 13 Execution time: 8.21729 msec Jitter time:1.78271 msec
Frame: 14 Execution time: 8.07354 msec Jitter time:1.92646 msec
Frame: 15 Execution time: 7.96344 msec Jitter time:2.03656 msec
Frame: 16 Execution time: 0 msec Jitter time:10 msec
Frame: 17 Execution time: 7.63474 msec Jitter time:2.36526 msec
Frame: 18 Execution time: 8.13182 msec Jitter time:1.86818 msec
Frame: 19 Execution time: 7.96245 msec Jitter time:2.03755 msec

Avergare Execution time: 7.72162msec Average Framerate: 129.507 fps (Calculated for 20 Frames)
```

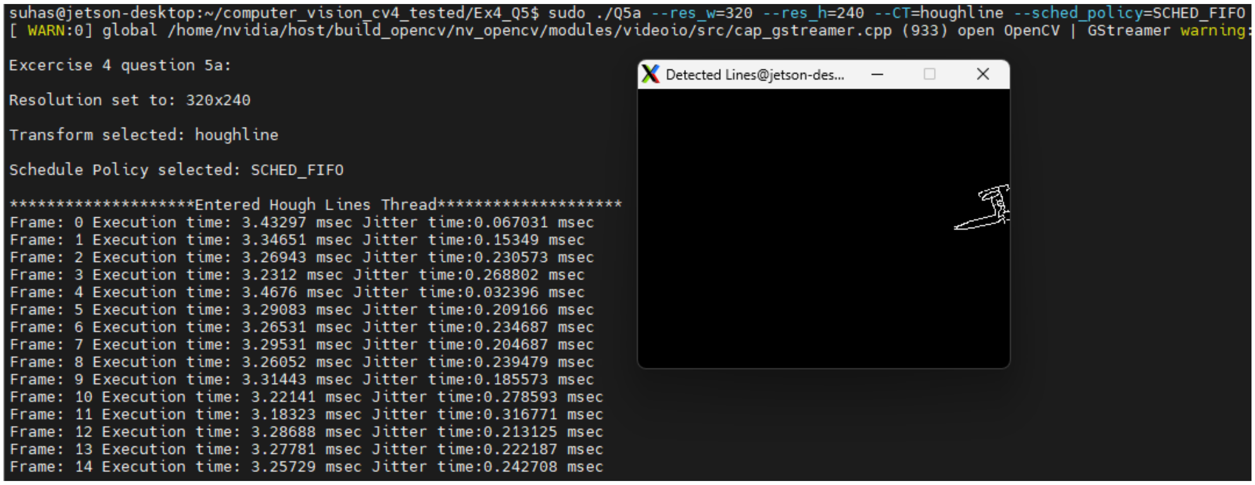


The plot takes 100 frames into consideration and has a resolution of 640x480. The graph plots execution time vs number of frames (Blue) and jitter time vs number of frames (Orange). The steep dips in blue and steep spikes in red indicate frame drops therefore the jitter is high at those points.

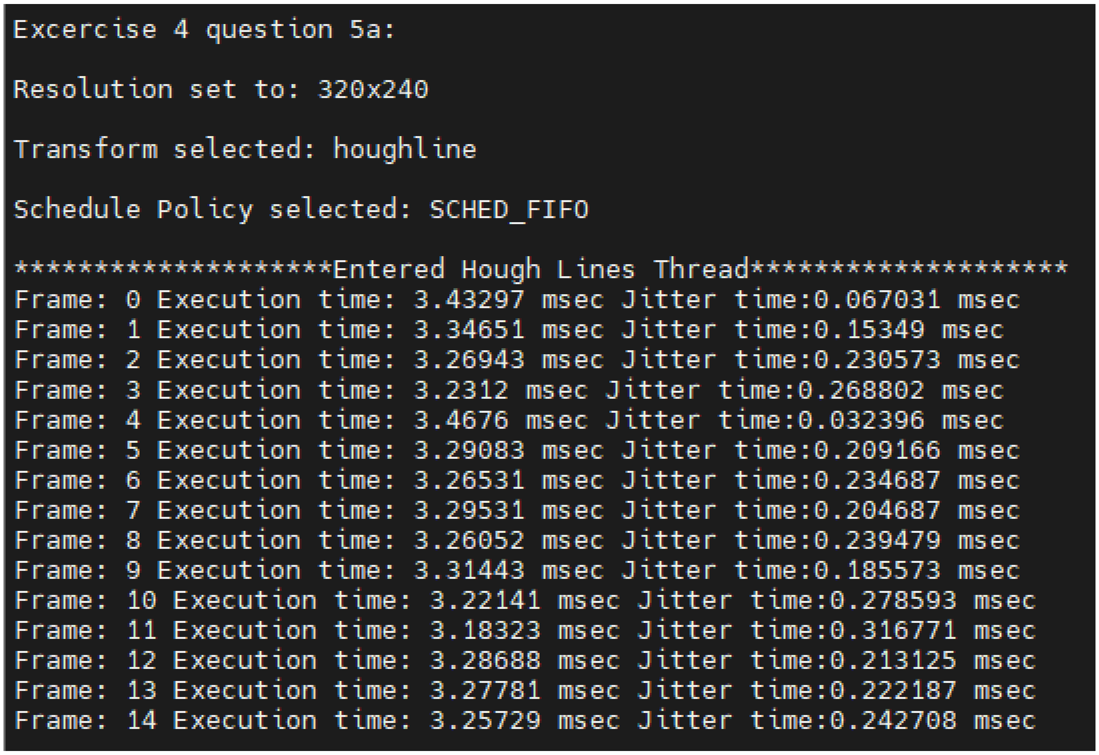
Hough Line:

Command:

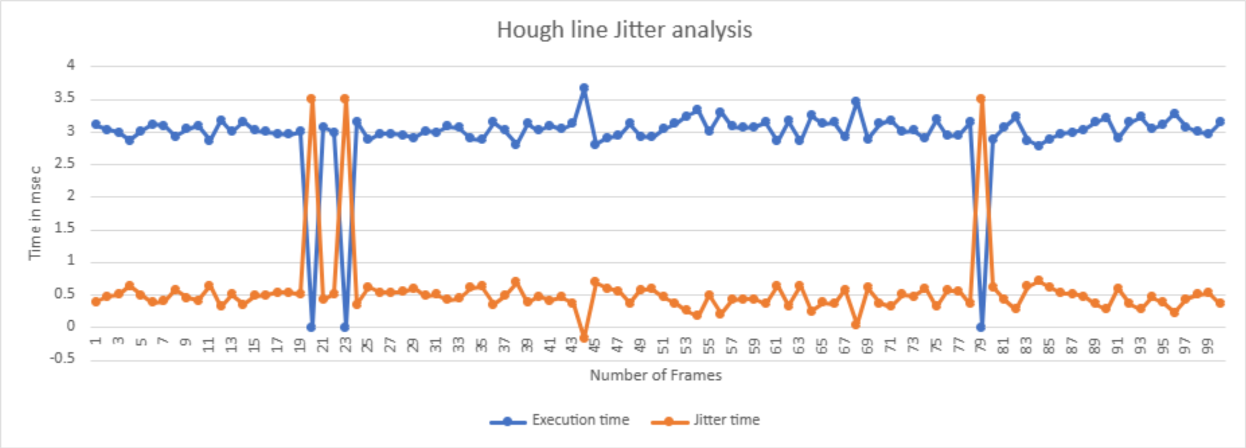
sudo ./Q5a --res_w=320 –res_h=240 --CT=houghline --sched_policy=SCHED_FIFO



```
suhas@jetson-desktop:~/computer_vision_cv4_tested/Ex4_Q5$ sudo ./Q5a --res_w=320 --res_h=240 --CT=houghline --sched_policy=SCHED_FIFO
[ WARN:0] global /home/nvidia/host/build_opencv/nv_opencv/modules/videoio/src/cap_gstreamer.cpp (933) open OpenCV | GStreamer warning:

Excercise 4 question 5a:

Resolution set to: 320x240

Transform selected: houghline

Schedule Policy selected: SCHED_FIFO

********************Entered Hough Lines Thread********************
Frame: 0 Execution time: 3.43297 msec Jitter time:0.067031 msec
Frame: 1 Execution time: 3.34651 msec Jitter time:0.15349 msec
Frame: 2 Execution time: 3.26943 msec Jitter time:0.230573 msec
Frame: 3 Execution time: 3.2312 msec Jitter time:0.268802 msec
Frame: 4 Execution time: 3.4676 msec Jitter time:0.032396 msec
Frame: 5 Execution time: 3.29083 msec Jitter time:0.209166 msec
Frame: 6 Execution time: 3.26531 msec Jitter time:0.234687 msec
Frame: 7 Execution time: 3.29531 msec Jitter time:0.204687 msec
Frame: 8 Execution time: 3.26052 msec Jitter time:0.239479 msec
Frame: 9 Execution time: 3.31443 msec Jitter time:0.185573 msec
Frame: 10 Execution time: 3.22141 msec Jitter time:0.278593 msec
Frame: 11 Execution time: 3.18323 msec Jitter time:0.316771 msec
Frame: 12 Execution time: 3.28688 msec Jitter time:0.213125 msec
Frame: 13 Execution time: 3.27781 msec Jitter time:0.222187 msec
Frame: 14 Execution time: 3.25729 msec Jitter time:0.242708 msec
```



```
Excercise 4 question 5a:

Resolution set to: 320x240

Transform selected: houghline

Schedule Policy selected: SCHED_FIFO

********************Entered Hough Lines Thread********************
Frame: 0 Execution time: 3.43297 msec Jitter time:0.067031 msec
Frame: 1 Execution time: 3.34651 msec Jitter time:0.15349 msec
Frame: 2 Execution time: 3.26943 msec Jitter time:0.230573 msec
Frame: 3 Execution time: 3.2312 msec Jitter time:0.268802 msec
Frame: 4 Execution time: 3.4676 msec Jitter time:0.032396 msec
Frame: 5 Execution time: 3.29083 msec Jitter time:0.209166 msec
Frame: 6 Execution time: 3.26531 msec Jitter time:0.234687 msec
Frame: 7 Execution time: 3.29531 msec Jitter time:0.204687 msec
Frame: 8 Execution time: 3.26052 msec Jitter time:0.239479 msec
Frame: 9 Execution time: 3.31443 msec Jitter time:0.185573 msec
Frame: 10 Execution time: 3.22141 msec Jitter time:0.278593 msec
Frame: 11 Execution time: 3.18323 msec Jitter time:0.316771 msec
Frame: 12 Execution time: 3.28688 msec Jitter time:0.213125 msec
Frame: 13 Execution time: 3.27781 msec Jitter time:0.222187 msec
Frame: 14 Execution time: 3.25729 msec Jitter time:0.242708 msec
```

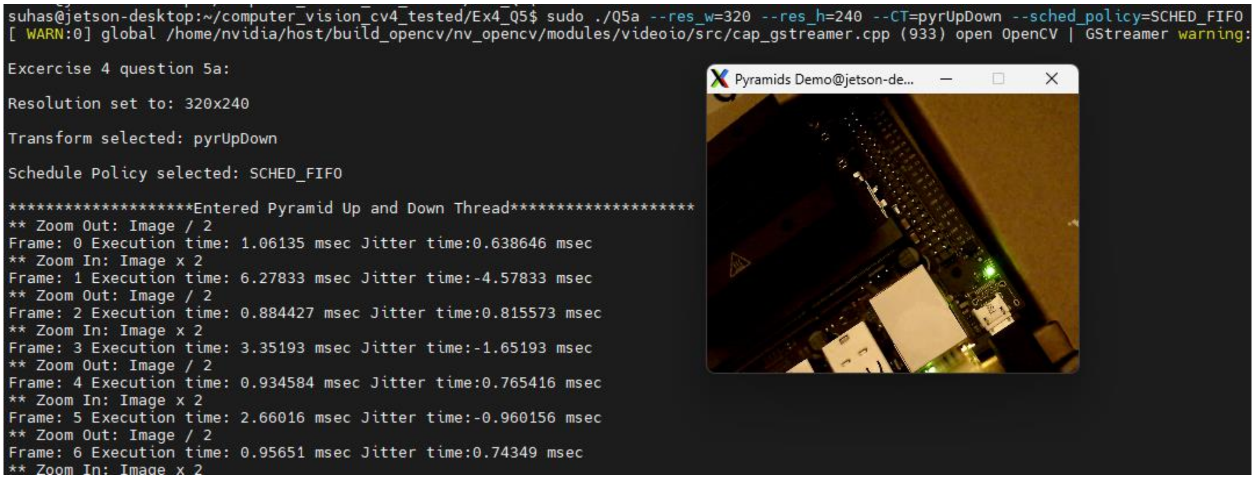


The plot takes 100 frames into consideration and a resolution of 320x240. The graph plots execution time vs number of frames (Blue) and jitter time vs number of frames (Orange). The steep dips in blue and steep spikes in red indicate frame drops therefore the jitter is high at those points.

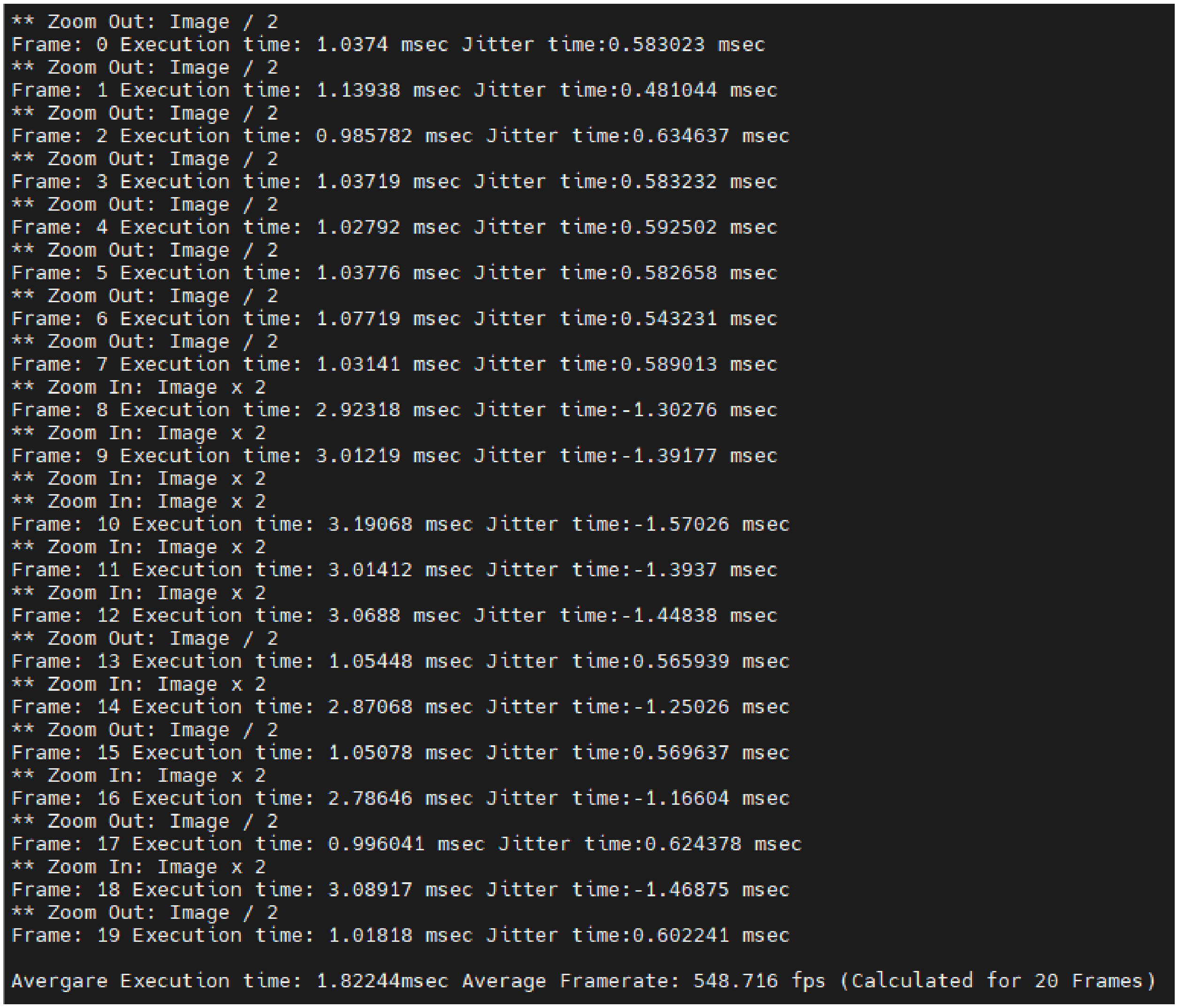
Pyramid Up and Down:

Command:

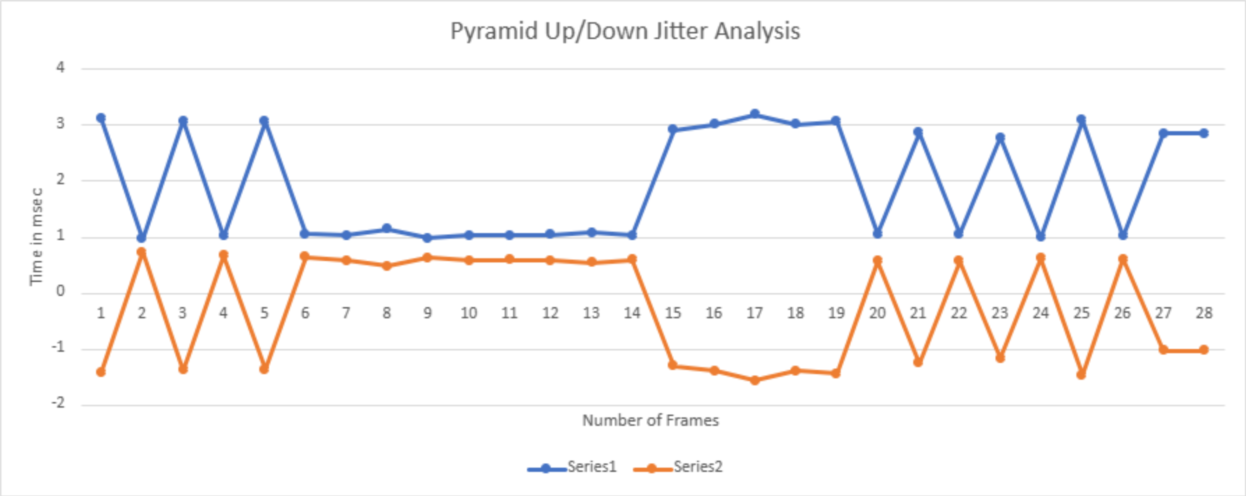
sudo ./Q5a --res_w=320 --res_h=240 --CT=pyrUpDown --sched_policy=SCHED_FIFO



```
suhas@jetson-desktop:~/computer_vision_cv4_tested/Ex4_Q5$ sudo ./Q5a --res_w=320 --res_h=240 --CT=pyrUpDown --sched_policy=SCHED_FIFO
[ WARN:0] global /home/nvidia/host/build_opencv/nv_opencv/modules/videoio/src/cap_gstreamer.cpp (933) open OpenCV | GStreamer warning:

Excercise 4 question 5a:

Resolution set to: 320x240

Transform selected: pyrUpDown

Schedule Policy selected: SCHED_FIFO

*******************Entered Pyramid Up and Down Thread*******************
** Zoom Out: Image / 2
Frame: 0 Execution time: 1.06135 msec Jitter time:0.638646 msec
** Zoom In: Image x 2
Frame: 1 Execution time: 6.27833 msec Jitter time:-4.57833 msec
** Zoom Out: Image / 2
Frame: 2 Execution time: 0.884427 msec Jitter time:0.815573 msec
** Zoom In: Image x 2
Frame: 3 Execution time: 3.35193 msec Jitter time:-1.65193 msec
** Zoom Out: Image / 2
Frame: 4 Execution time: 0.934584 msec Jitter time:0.765416 msec
** Zoom In: Image x 2
Frame: 5 Execution time: 2.66016 msec Jitter time:-0.960156 msec
** Zoom Out: Image / 2
Frame: 6 Execution time: 0.95651 msec Jitter time:0.74349 msec
** Zoom In: Image x 2
```



```
** Zoom Out: Image / 2
Frame: 0 Execution time: 1.0374 msec Jitter time:0.583023 msec
** Zoom Out: Image / 2
Frame: 1 Execution time: 1.13938 msec Jitter time:0.481044 msec
** Zoom Out: Image / 2
Frame: 2 Execution time: 0.985782 msec Jitter time:0.634637 msec
** Zoom Out: Image / 2
Frame: 3 Execution time: 1.03719 msec Jitter time:0.583232 msec
** Zoom Out: Image / 2
Frame: 4 Execution time: 1.02792 msec Jitter time:0.592502 msec
** Zoom Out: Image / 2
Frame: 5 Execution time: 1.03776 msec Jitter time:0.582658 msec
** Zoom Out: Image / 2
Frame: 6 Execution time: 1.07719 msec Jitter time:0.543231 msec
** Zoom Out: Image / 2
Frame: 7 Execution time: 1.03141 msec Jitter time:0.589013 msec
** Zoom In: Image x 2
Frame: 8 Execution time: 2.92318 msec Jitter time:-1.30276 msec
** Zoom In: Image x 2
Frame: 9 Execution time: 3.01219 msec Jitter time:-1.39177 msec
** Zoom In: Image x 2
** Zoom In: Image x 2
Frame: 10 Execution time: 3.19068 msec Jitter time:-1.57026 msec
** Zoom In: Image x 2
Frame: 11 Execution time: 3.01412 msec Jitter time:-1.3937 msec
** Zoom In: Image x 2
Frame: 12 Execution time: 3.0688 msec Jitter time:-1.44838 msec
** Zoom Out: Image / 2
Frame: 13 Execution time: 1.05448 msec Jitter time:0.565939 msec
** Zoom In: Image x 2
Frame: 14 Execution time: 2.87068 msec Jitter time:-1.25026 msec
** Zoom Out: Image / 2
Frame: 15 Execution time: 1.05078 msec Jitter time:0.569637 msec
** Zoom In: Image x 2
Frame: 16 Execution time: 2.78646 msec Jitter time:-1.16604 msec
** Zoom Out: Image / 2
Frame: 17 Execution time: 0.996041 msec Jitter time:0.624378 msec
** Zoom In: Image x 2
Frame: 18 Execution time: 3.08917 msec Jitter time:-1.46875 msec
** Zoom Out: Image / 2
Frame: 19 Execution time: 1.01818 msec Jitter time:0.602241 msec

Avergare Execution time: 1.82244msec Average Framerate: 548.716 fps (Calculated for 20 Frames)
```



The plot takes about 30 frames into consideration and a resolution of 320x240. The graph plots execution time vs number of frames (Blue) and jitter time vs number of frames

(Orange). The steep dips in blue and steep spikes in red indicate frame drops and inconsistent user inputs therefore the jitter is high at those points.

Appendix:

Excercise 4 Question 5a

```
#include <iostream>
#include <pthread.h>
#include <opencv2/opencv.hpp>
#include <semaphore.h>
#include <sched.h>


using namespace cv;
using namespace std;


void end_delay_test(void);


sem_t logSem, stopSem;
struct timespec startTime, endTime;
bool stop_logging = false;


long long totalFrameTime_nsec = 0; // Total time for frames in nanoseconds
int frameCount = 0;                // Count of frames processed
int numFramesForAvg = 100;         // Number of frames to average over
long double avgExecTime_msec = 1.0;


FILE *fpt;


#define ESCAPE_KEY 27
#define FRAME_DELAY 33  // Frame delay in milliseconds (about 30 frames per second)
```

```
#define NSEC_PER_SEC (1000000000)
#define DELAY_TICKS (1)
#define ERROR (-1)
#define OK (0)

pthread_mutex_t cameraMutex; // Mutex for camera access

struct ThreadData {
    VideoCapture* cam;
};

int delta_t(struct timespec *stop, struct timespec *start, struct timespec *delta_t)
{
    // Ensure start and stop times are not NULL
    if (start == NULL || stop == NULL || delta_t == NULL) {
        return ERROR; // return error if any input is NULL
    }

    // Calculate the time difference
    delta_t->tv_sec = stop->tv_sec - start->tv_sec;
    delta_t->tv_nsec = stop->tv_nsec - start->tv_nsec;

    // Normalize the time so that tv_nsec is less than 1 second
    if (delta_t->tv_nsec < 0) {
        --delta_t->tv_sec;
        delta_t->tv_nsec += NSEC_PER_SEC;
    }
    // Ensure that the time difference is not negative
    if (delta_t->tv_sec < 0 || (delta_t->tv_sec == 0 && delta_t->tv_nsec < 0)) {
        // The stop time is before the start time, which should not happen
        delta_t->tv_sec = 0;
        delta_t->tv_nsec = 0;
        return ERROR; // return an error code
    }

    return OK; // return OK for successful execution
}

void* LoggingThread(void* threadp) {
    struct timespec ts;
    fprintf(fpt,"Frame no, Execution time, Jitter time\n");
    while (true) {
        clock_gettime(CLOCK_REALTIME, &ts);
        ts.tv_sec += 1; // Wait for up to 1 second
        if (sem_timedwait(&stopSem, &ts) == -1) {
            if (errno != ETIMEDOUT) {
                break;
            }
        } else {
            break;
        }
        if (sem_trywait(&logSem) == 0) {
            struct timespec elapsedTime;
            delta_t(&endTime, &startTime, &elapsedTime);


            long long frameTime_nsec = elapsedTime.tv_sec * NSEC_PER_SEC +
elapsedTime.tv_nsec;
            long double frameTime_msec = (long double)frameTime_nsec / (1000000);


            cout <<"Frame: " << frameCount << " Execution time: "<< frameTime_msec << " msec
Jitter time:" << avgExecTime_msec - frameTime_msec << " msec" << endl;
            fprintf(fpt,"%d, %Lf, %Lf\n",frameCount, frameTime_msec, avgExecTime_msec -
frameTime_msec);


            totalFrameTime_nsec += frameTime_nsec;
            frameCount++;


            // Calculate and display average framerate every numFramesForAvg frames
            if (frameCount == numFramesForAvg) {
                if (totalFrameTime_nsec > 0) {
                  long double avgTimePerFrame_msec = (long double)totalFrameTime_nsec /
(numFramesForAvg * 1000000);
                  long double avgFramerate = 1000 / avgTimePerFrame_msec;
                  cout << endl << "Avergare Execution time: "<< avgTimePerFrame_msec <<"msec
Average Framerate: " << avgFramerate << " fps (Calculated for " << numFramesForAvg << "
Frames)" << endl << endl;
                  //Update avg excution time to calulate jitter
                  avgExecTime_msec = avgTimePerFrame_msec;
                }


                // Reset counters
                totalFrameTime_nsec = 0;
```

```
                frameCount = 0;
            }
        }
    }
    return nullptr;
}

void* CannyThread(void* arg) {
    ThreadData* data = static_cast<ThreadData*>(arg);
    int PreiodMS = 100;
    cout << "********************Entered Canny Thread********************" <<endl;
    Mat frame, src_gray, detected_edges, dst;
    namedWindow("Edge Map", WINDOW_AUTOSIZE);

    int lowThreshold = 0;
    int max_lowThreshold = 100;
    int kernel_size = 3;
    createTrackbar("Min Threshold:", "Edge Map", &lowThreshold, max_lowThreshold);
    while(true) {
        pthread_mutex_lock(&cameraMutex);
        bool success = data->cam->read(frame);
        pthread_mutex_unlock(&cameraMutex);

        clock_gettime(CLOCK_REALTIME, &startTime);
        if (!success) {
            cerr << "Error: Could not grab a frame" << endl;
            break;
        }


        cvtColor(frame, src_gray, COLOR_BGR2GRAY);
        blur(src_gray, detected_edges, Size(3,3));
        Canny(detected_edges, detected_edges, lowThreshold, lowThreshold * 3, kernel_size);
        dst = Scalar::all(0);
        frame.copyTo(dst, detected_edges);
        imshow("Edge Map", dst);
        clock_gettime(CLOCK_REALTIME, &endTime);
        sem_post(&logSem);
        char c = (char)waitKey(PreiodMS);
        if (c == ESCAPE_KEY) {
            stop_logging = true;
            break;
        }
    }


    return nullptr;
}


void* HoughLinesThread(void* arg) {
    ThreadData* data = static_cast<ThreadData*>(arg);
    int PreiodMS = 60;
    cout << "********************Entered Hough Lines Thread********************"
<<endl;
    Mat frame, dst, cdst, cdstP;
    namedWindow("Detected Lines", WINDOW_AUTOSIZE);
```

```
while(true) {
    pthread_mutex_lock(&cameraMutex);
    bool success = data->cam->read(frame);
    pthread_mutex_unlock(&cameraMutex);

    if (!success) {
        cerr << "Error: Could not grab a frame" << endl;
        break;
    }

    clock_gettime(CLOCK_REALTIME, &startTime);
    // Convert to grayscale
    cvtColor(frame, frame, COLOR_BGR2GRAY);

    // Edge detection
    Canny(frame, dst, 80, 240, 3);

    // Copy edges to the images that will display the results in BGR
    cvtColor(dst, cdst, COLOR_GRAY2BGR);
    cdstP = cdst.clone();

    // Probabilistic Line Transform
    vector<Vec4i> linesP; // will hold the results of the detection
    HoughLinesP(dst, linesP, 1, CV_PI/180, 50, 50, 10); // runs the actual detection

    // Draw the lines
    for(size_t i = 0; i < linesP.size(); i++) {
            Vec4i l = linesP[i];
            line(cdstP, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(0,0,255), 3, LINE_AA);
        }

        // Show results
        imshow("Detected Lines", cdstP);
        clock_gettime(CLOCK_REALTIME, &endTime);
        sem_post(&logSem);
        char c = (char)waitKey(PreiodMS);
        if (c == ESCAPE_KEY) {
            stop_logging = true;
            break;
        }
    }
    return nullptr;
}

void* PyrUpDownThread(void* arg) {
    ThreadData* data = static_cast<ThreadData*>(arg);
    int PreiodMS = 500;
    cout << "********************Entered Pyramid Up and Down
Thread********************" <<endl;
    Mat frame;
    namedWindow("Pyramids Demo", WINDOW_AUTOSIZE);

    while(true) {
        pthread_mutex_lock(&cameraMutex);
```

```
bool success = data->cam->read(frame);
pthread_mutex_unlock(&cameraMutex);

if (!success) {
    cerr << "Error: Could not grab a frame" << endl;
    break;
}

imshow("Pyramids Demo", frame);

char c = (char)waitKey(0); // Wait for a key press
clock_gettime(CLOCK_REALTIME, &startTime);
if (c == ESCAPE_KEY) {
    break;
} else if (c == 'i') {
    Mat temp;
    pyrUp(frame, temp, Size(frame.cols * 2, frame.rows * 2));
    frame = temp;
    printf("** Zoom In: Image x 2 \n");
} else if (c == 'o') {
    Mat temp;
    pyrDown(frame, temp, Size(frame.cols / 2, frame.rows / 2));
    frame = temp;
    printf("** Zoom Out: Image / 2 \n");
}

imshow("Pyramids Demo", frame); // Redisplay the frame after zoom operation
```

```
        clock_gettime(CLOCK_REALTIME, &endTime);
        sem_post(&logSem);
        waitKey(PreiodMS); // Add delay to observe the zoom effect
    }
    return nullptr;
}

int setRealTimeScheduling(int priority, String sched_policy) {
    struct sched_param sch_params;
    sch_params.sched_priority = (sched_policy == "SCHED_FIFO") ? priority : 0;
    if(sched_policy == "SCHED_FIFO") {
      cout << endl << "Schedule Policy selected: SCHED_FIFO" << endl << endl;
      return sched_setscheduler(0, SCHED_FIFO, &sch_params);
    } else {
      cout << endl << "Schedule Policy selected: SCHED_OTHER" << endl << endl;
      return sched_setscheduler(0, SCHED_OTHER, &sch_params);
    }
}

int main(int argc, char *argv[]) {
    CommandLineParser parser(argc, argv,
                             "{res_w w|640|camera resolution width}"
                             "{res_h h|480|camera resolution height}"
                             "{CT   |  |continuous transform type}"
                             "{sched_policy   |  |scheduling policy type}");

    VideoCapture cam0(0);
```

```
if (!cam0.isOpened()) {
    cerr << "Error: Could not open camera" << endl;
    return -1;
}

fpt = fopen("ContinuousTransform.csv", "w+");

// Initialize the camera mutex
pthread_mutex_init(&cameraMutex, NULL);
sem_init(&logSem, 0, 0);

cout << endl << "Excercise 4 question 5a:" << endl;

int width = 640;
width = parser.get<int>("res_w");
int height = 480;
height = parser.get<int>("res_h");

cout << endl << "Resolution set to: " << width << "x"<< height << endl;

cam0.set(CAP_PROP_FRAME_WIDTH, width);
cam0.set(CAP_PROP_FRAME_HEIGHT, height);

String cmd = parser.get<String>("CT");
if (cmd.empty()) {
    cout << "No command provided. Please provide --CT=<command>." << endl;
    return -1;
```

```
}

cout << endl << "Transform selected: " << cmd << endl;

String sched_policy = "SCHED_OTHER";
sched_policy = parser.get<String>("sched_policy");

if (setRealTimeScheduling(99, sched_policy) != 0) { // 99 is a high priority
    cerr << "Failed to set real-time scheduling policy." << endl;
    return -1;
}

ThreadData data;
data.cam = &cam0;

pthread_t logThread;
pthread_create(&logThread, NULL, LoggingThread, NULL);

pthread_t thread;
int rc;
cpu_set_t cpuset;

// Initialize CPU set to the desired CPU(s)
CPU_ZERO(&cpuset);   // Clear the CPU set
CPU_SET(0, &cpuset);

if (cmd == "canny") {
    rc = pthread_create(&thread, NULL, CannyThread, &data);
    avgExecTime_msec *= 10;
} else if (cmd == "houghline") {
    rc = pthread_create(&thread, NULL, HoughLinesThread, &data);
    avgExecTime_msec *= 50;
} else if (cmd == "pyrUpDown") {
    rc = pthread_create(&thread, NULL, PyrUpDownThread, &data);
    avgExecTime_msec *= 10;
} else {
    cout << "Invalid command provided: " << cmd << endl;
    return -1;
}

if (rc == 0) {
        pthread_setaffinity_np(thread, sizeof(cpu_set_t), &cpuset);
        pthread_setaffinity_np(logThread, sizeof(cpu_set_t), &cpuset);
    }

if (rc) {
    cout << "Error: unable to create thread," << rc << endl;
    return -1;
}

pthread_join(thread, NULL); // Wait for the thread to finish
sem_post(&stopSem);
pthread_join(logThread, NULL);
```

```
    pthread_mutex_destroy(&cameraMutex);
    sem_destroy(&logSem);
    fclose(fpt);

    return 0;
}
```