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

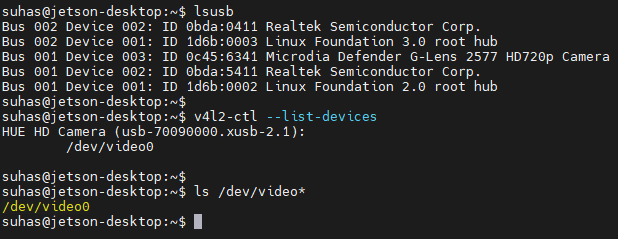
1) [10 points] Warning: START EARLY. Obtain a Logitech C200 or C270 camera or equivalent and verify that is detected by the DE1-SoC, Raspberry Pi or Jetson Board USB driver. You can check the camera out from eStore or purchase one of your own or use another camera that has a compliant UVC driver. Use 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:

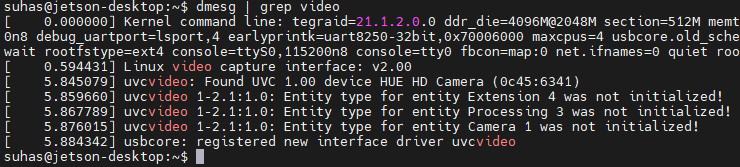
After connecting the USB Webcam to the Jetson Nano, the Jetson identifies the device class and its various pipes with the UVC (usb video class) driver, and makes the device video stream available as an attached device (at /dev/video0 in our case).

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

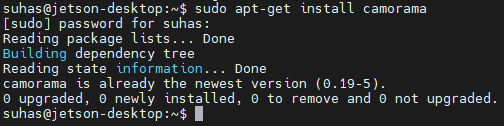




2) [10 points] Camera Tools – You only need to do either Option 1 or Option 2.

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:

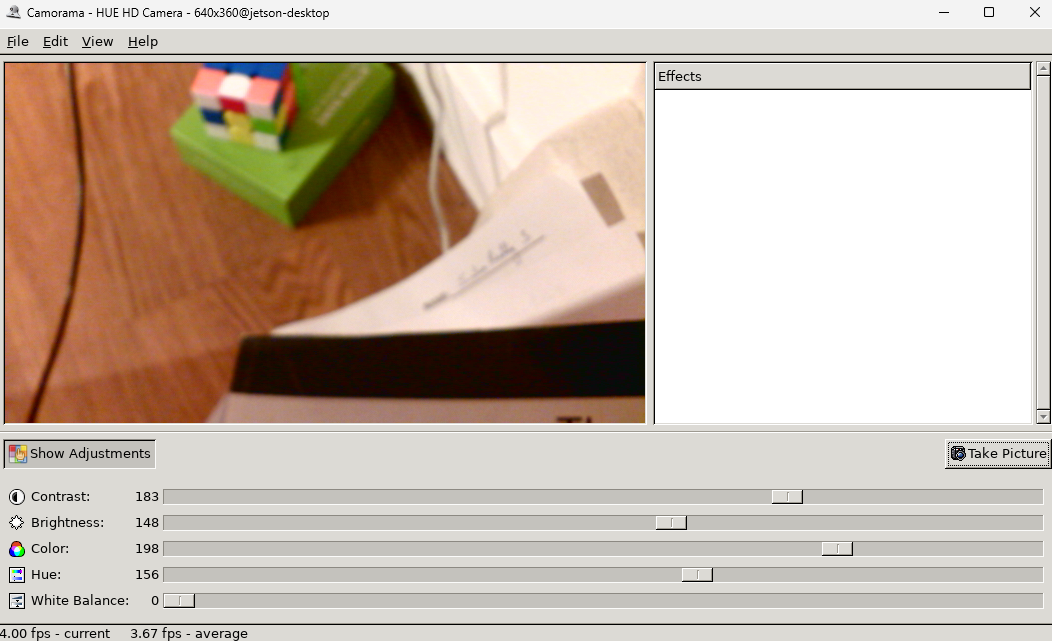
Camorama Console: 

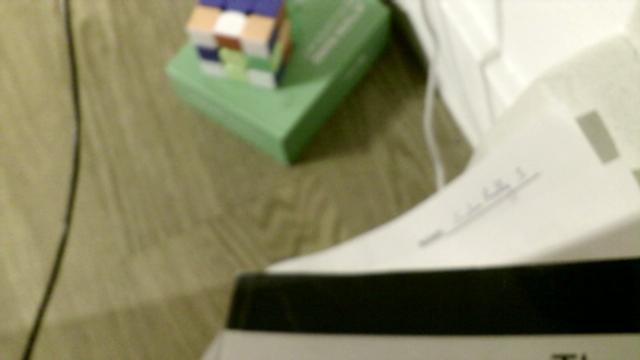
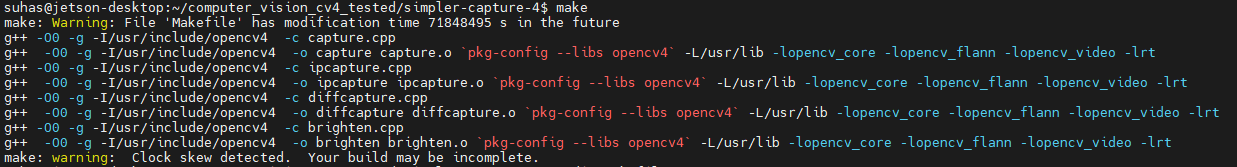
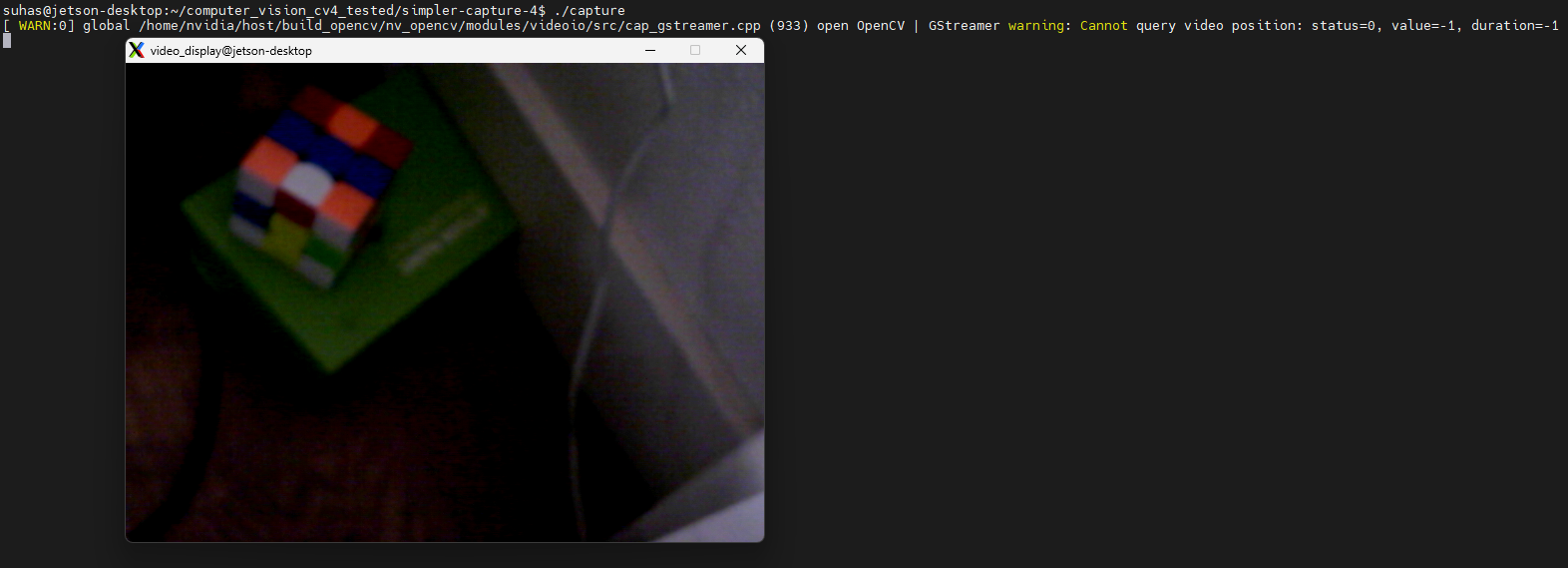
Image before adjustment: 

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:

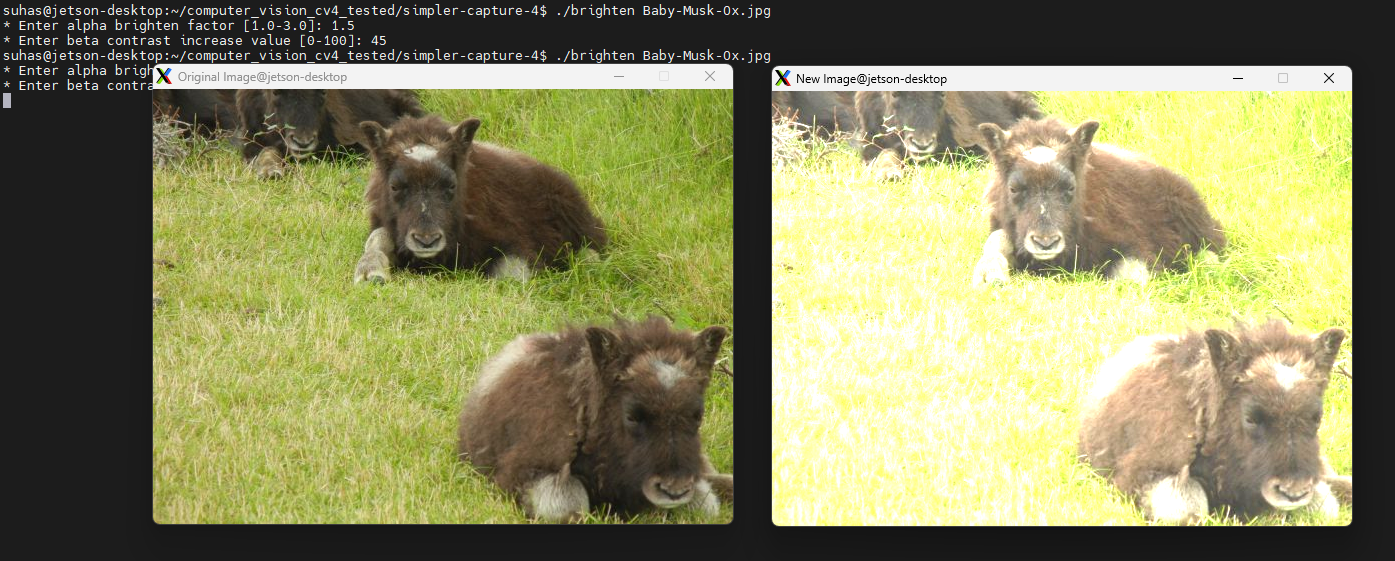
Simpler\_Capture-4 Built:



Capture.cpp

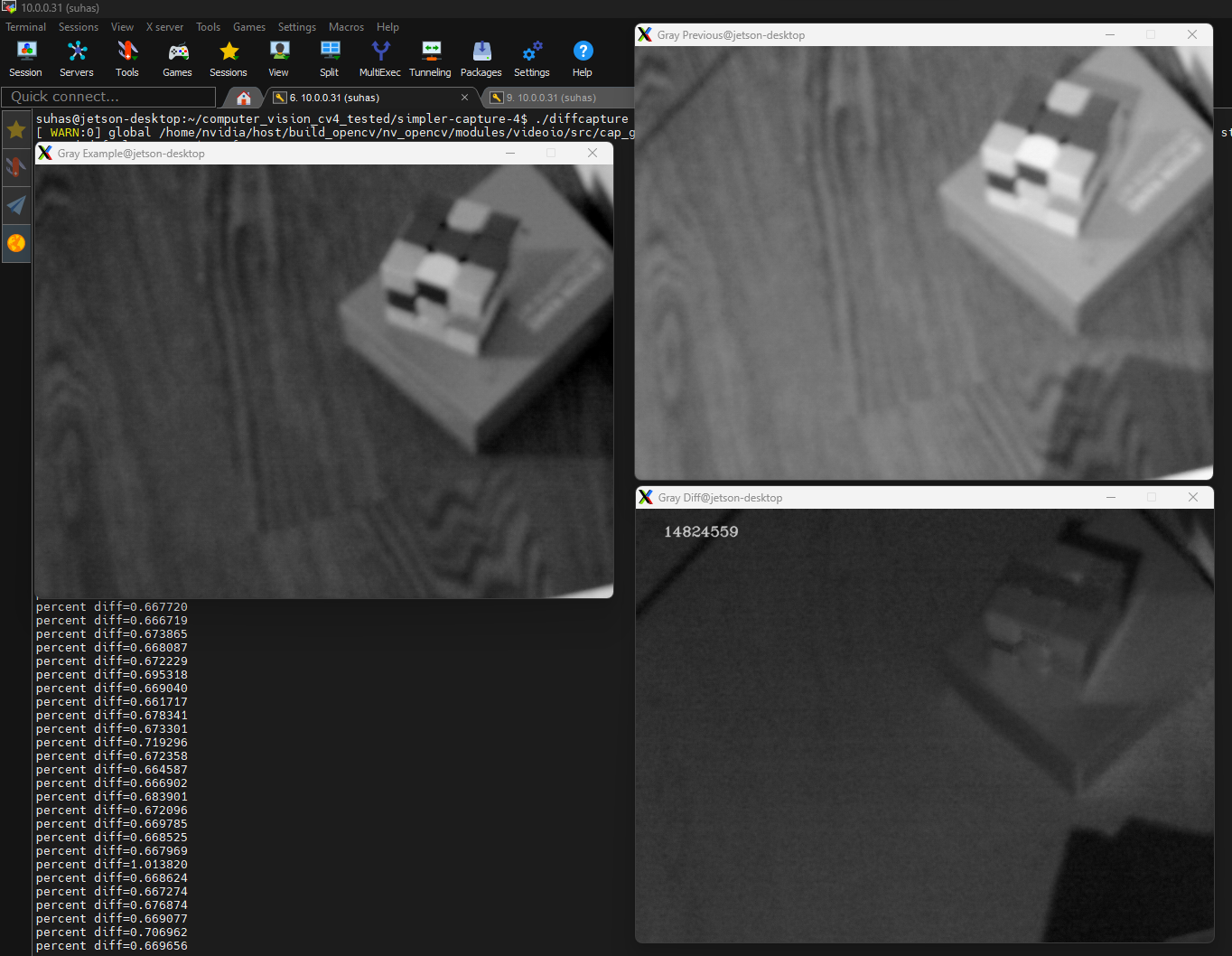
Code Brief:

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)  


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

Code

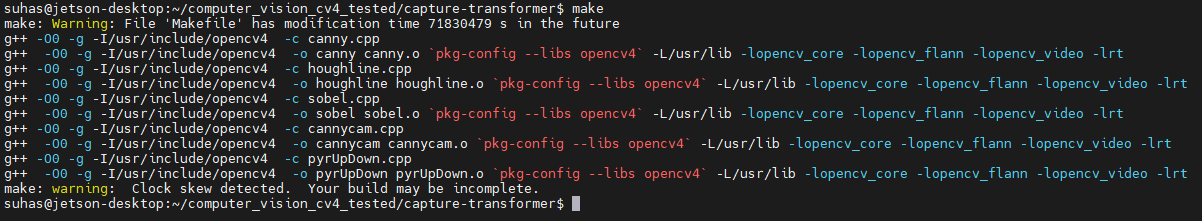
The DiffCapture algorithm continuously captures each image and compares it to the previous one. The comparison is made by measuring the greyscale brightness value, pixel by pixel, and calculating the absolute value of the difference between the previous image’s value. A new image is constructed with 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]

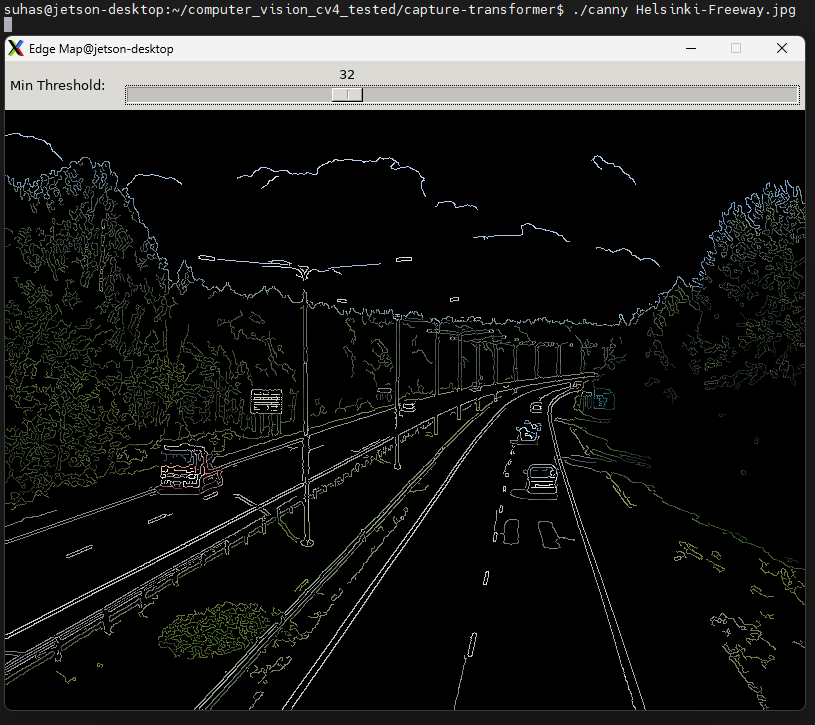
a) Choose a continuous transformation OpenCV example from computer vision cv3 tested such as the canny-interactive, hough-interactive, hough-eliptical-interactive, or stereo-transformimpoved . Show a screen shot to prove you built and ran the code. Provide a detailed explanation of the code and research 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:



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

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.

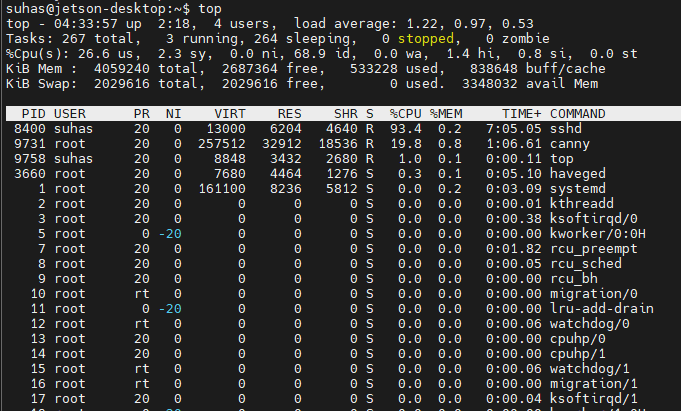
Second, the smoothed image is processed again with a gradient function. The gradient function calculates the local multivariate derivative with a Sobel kernel; this is a two-step matrix multiplication for a 3\*3 matrix which calculates both the slope and direction of gradient for each pixel.

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.

CPU Core Loading:

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.



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.



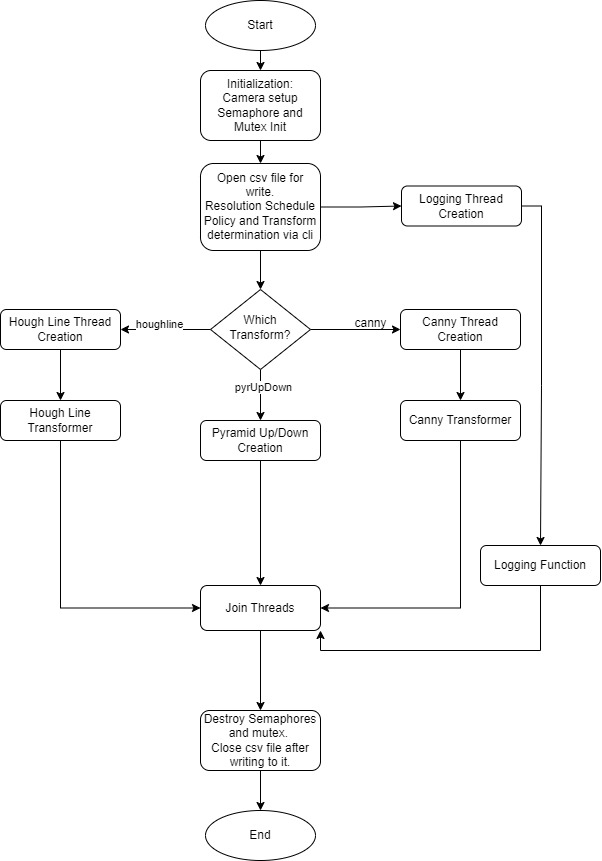
5) [40 points] a) Using a Logitech C2xx, choose 3 real-time interactive transformations to compare in terms of average frame rate at a given resolution for a range of at least 3 resolutions in a common aspect ratio (e.g. 4:3 for 1280x960, 640x480, 320x240, 160x120, 80x60) by adding time-stamps (there 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.



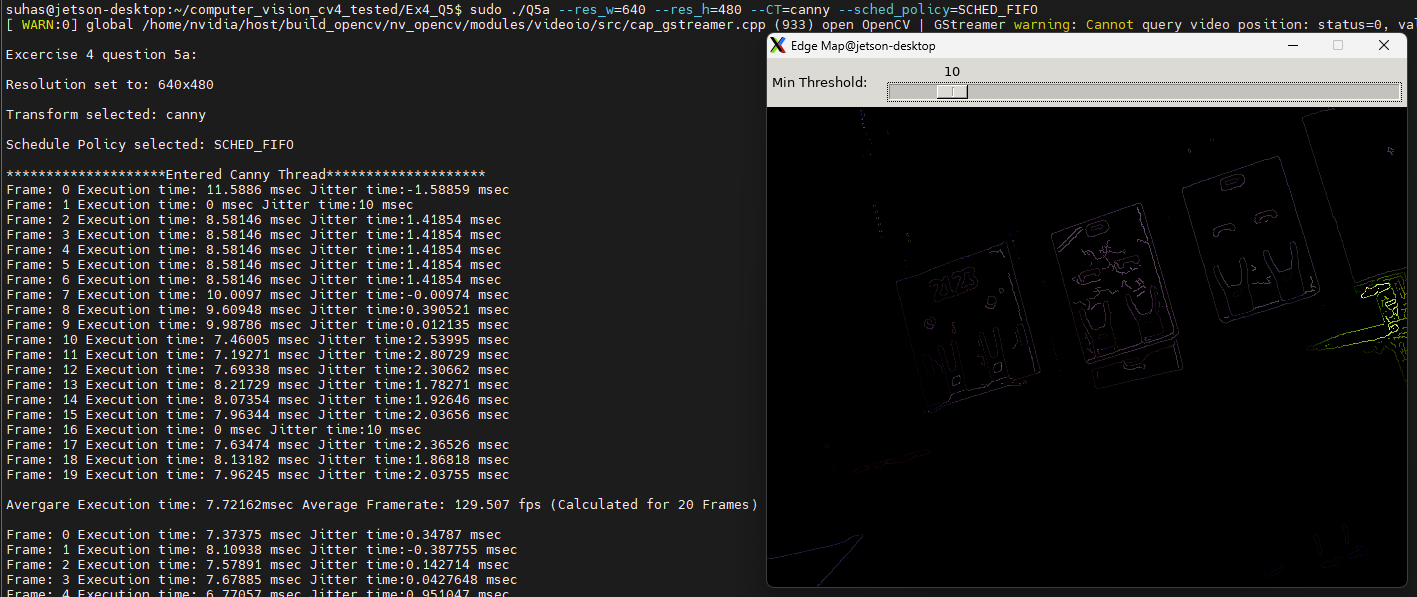
Flow chart

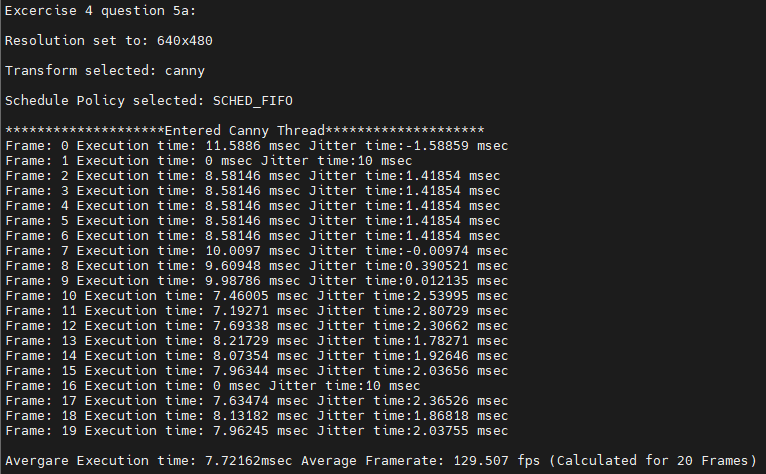
Table:

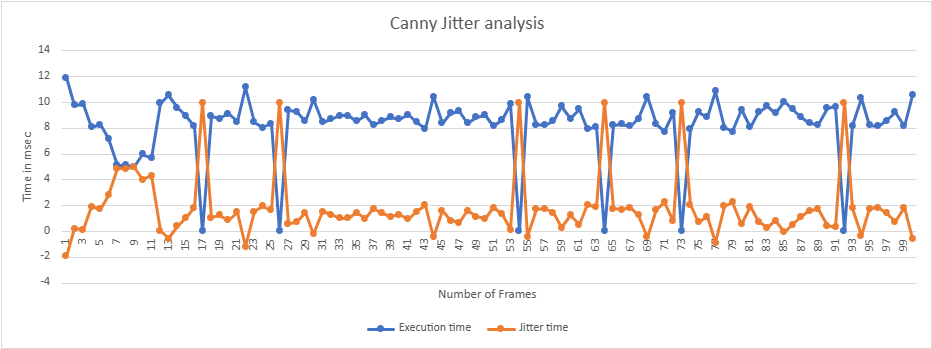
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Transform | Resolution | Average Execution Time | Frame Rate | No of frames |
| Canny | 640x480 | 9.81916 msec | 101.842 fps | 100 |
| Canny | 320x240 | 2.80111 msec | 357.001 fps | 100 |
| Canny | 160x120 | 0.861508 msec | 1160.75 fps | 100 |
| Hough 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





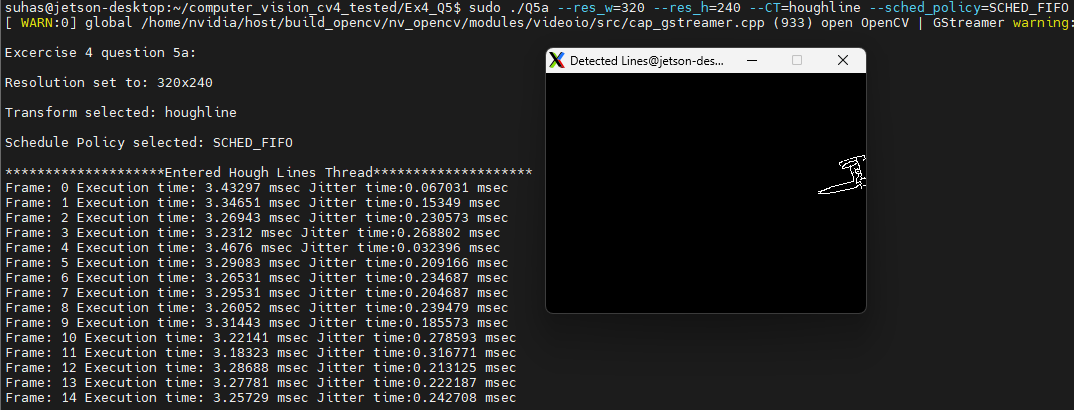


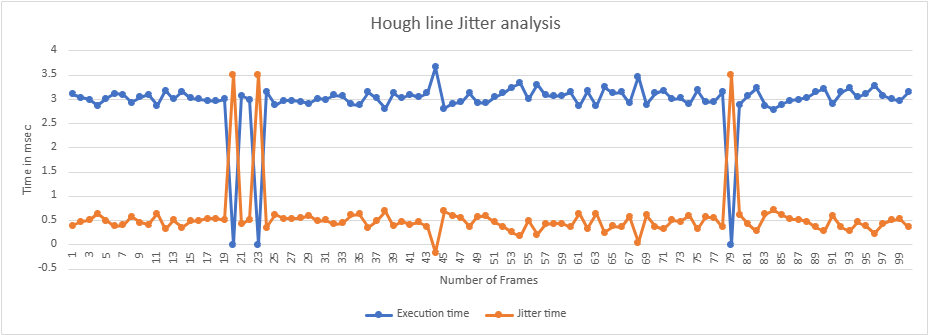
The plot takes 100 frames into consideration and has a resolution of 640x480. The graph plots execution time vs number of frames (Blue) and jitter time vs number of frames (Orange). The steep dips in blue and steep spikes in red indicate frame drops therefore the jitter is high at those points.

Hough Line:

Command:

sudo ./Q5a --res\_w=320 –res\_h=240 --CT=houghline --sched\_policy=SCHED\_FIFO



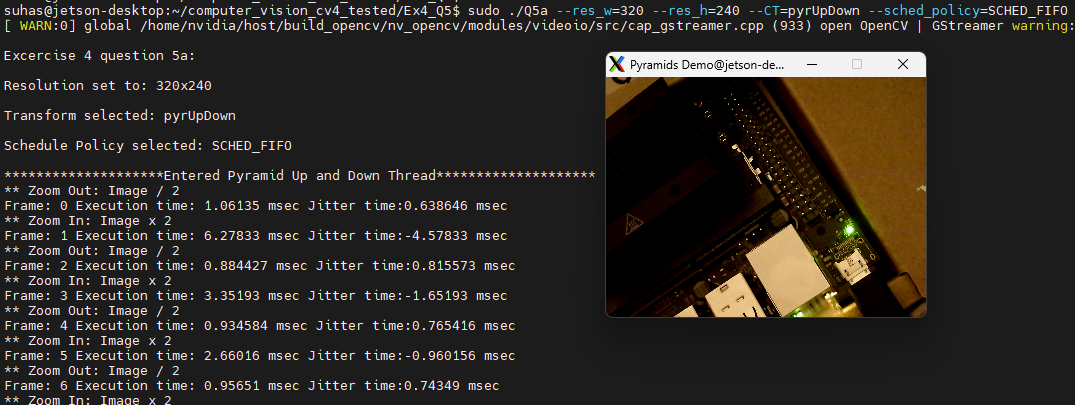


The plot takes 100 frames into consideration and a resolution of 320x240. The graph plots execution time vs number of frames (Blue) and jitter time vs number of frames (Orange). The steep dips in blue and steep spikes in red indicate frame drops therefore the jitter is high at those points.

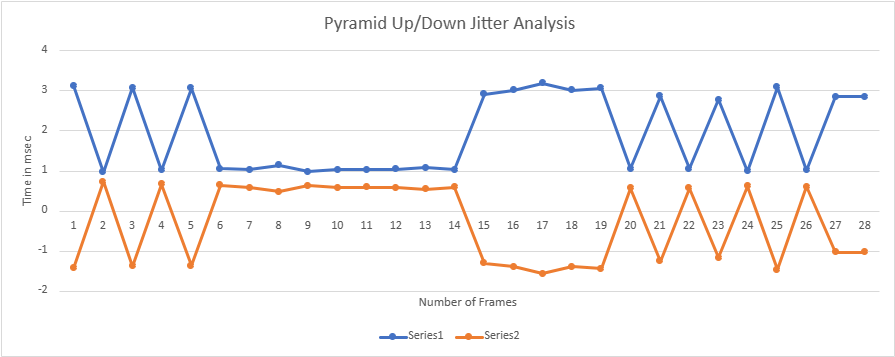
Pyramid Up and Down:

Command:

sudo ./Q5a --res\_w=320 --res\_h=240 --CT=pyrUpDown --sched\_policy=SCHED\_FIFO







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;

}