

Real Time Embedded Systems – ECEN 5623

Summer 2019

Professor Sam Siewert

Report By: Poorn Mehta

Platform: Jetson Nano (Camera: Logitech C920s)

Homework 4 (git: <https://github.com/Poorn-Mehta/RTES/tree/master/HW3>)

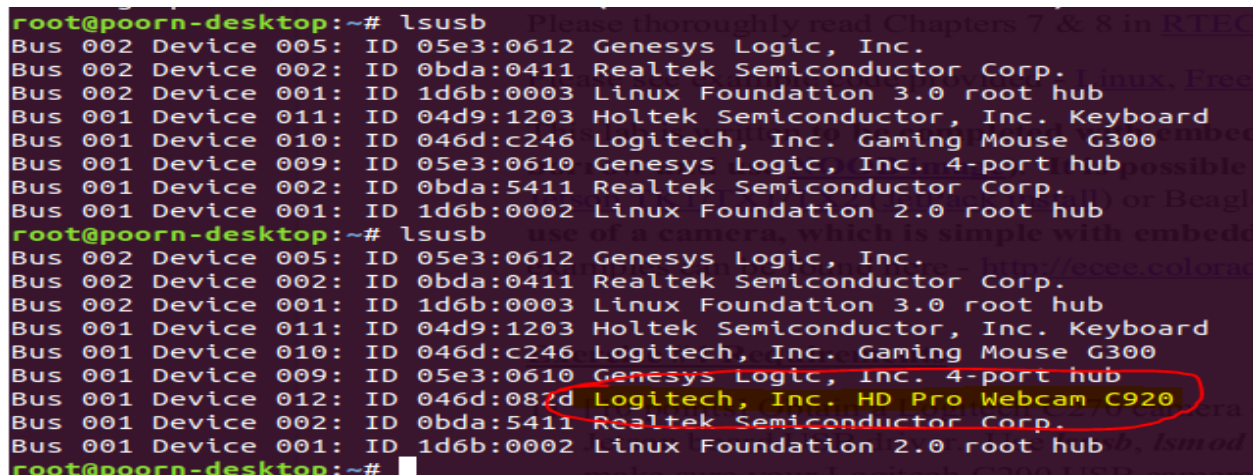
Capture Images: <https://drive.google.com/open?id=1WxHXocm0QxmOeRZy-ZEErzgXhJDdZniu>

Note: **I've used the simple capture code, provided by professor throughout the assignment.** All screenshots are uploaded on GitHub as well. Please refer to [that](#) in case higher quality is required. Also, due to time restrictions, I wasn't able to highlight important parts/key points. However, all the questions are addressed in here, and if it seems that that is not the case, please let me know. Thanks!

Q1: Obtain a Logitech C270 camera (or equivalent) and verify that is detected by the Jetson board USB driver. Use lsusb, lsmod and dmesg kernel driver configuration tool to make sure your Logitech C200 USB camera is plugged in and recognized by your Jetson. Do lsusb | grep C200 and prove to me (and more importantly yourself) with that output (screenshot) that your camera is recognized. Now, 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.

Solution:

lsusb



```
root@poorn-desktop:~# lsusb
Bus 002 Device 005: ID 05e3:0612 Genesys Logic, Inc.
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 011: ID 04d9:1203 Holtek Semiconductor, Inc. Keyboard
Bus 001 Device 010: ID 046d:c246 Logitech, Inc. Gaming Mouse G300
Bus 001 Device 009: ID 05e3:0610 Genesys Logic, Inc. 4-port hub
Bus 001 Device 002: ID 0bda:5411 Realtek Semiconductor Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
root@poorn-desktop:~# lsusb
Bus 002 Device 005: ID 05e3:0612 Genesys Logic, Inc.
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 011: ID 04d9:1203 Holtek Semiconductor, Inc. Keyboard
Bus 001 Device 010: ID 046d:c246 Logitech, Inc. Gaming Mouse G300
Bus 001 Device 009: ID 05e3:0610 Genesys Logic, Inc. 4-port hub
Bus 001 Device 012: ID 046d:082d Logitech, Inc. HD Pro Webcam C920
Bus 001 Device 002: ID 0bda:5411 Realtek Semiconductor Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
root@poorn-desktop:~#
```

```

root@poorn-desktop:~# lsusb | grep C920
Bus 001 Device 012: ID 046d:082d Logitech, Inc. HD Pro Webcam C920
root@poorn-desktop:~#

```

lsmod

```

root@poorn-desktop:~# lsmod
Module                  Size  Used by
uvccvideo               88565  0
fuse                   103841  2
bnep                    16562  2
overlay                48691  0
nvs                     54527  0
nvgpu                  1555053  56
bluetooth_pm           13912  0
ip_tables              19441  0
x_tables               28951  1 ip_tables
root@poorn-desktop:~#

```

```

root@poorn-desktop:~# lsmod | grep video
uvccvideo               88565  0

```

dmesg

```

root@poorn-desktop: ~
ing IP firewalling.)
[57810.921511] systemd[1]: Stopping Journal Service...
[57810.921862] systemd-journal[1925]: Received SIGTERM from PID 1 (systemd).
[57810.932153] systemd[1]: Stopped Journal Service.
[57810.935993] systemd[1]: Starting Journal Service...
[57811.010279] systemd[1]: Started Journal Service.
[58271.310127] usb 1-2.1: new high-speed USB device number 12 using tegra-xusb
[58273.568003] usb 1-2.1: New USB device found, idVendor=046d, idProduct=082d
[58273.568010] usb 1-2.1: New USB device strings: Mfr=0, Product=2, SerialNumber=1
[58273.568013] usb 1-2.1: Product: HD Pro Webcam C920
[58273.568016] usb 1-2.1: SerialNumber: FB63903F
[58273.833326] uvccvideo: Found UVC 1.00 device HD Pro Webcam C920 (046d:082d)
[58273.834551] uvccvideo 1-2.1:1.0: Entity type for entity Processing 3 was not initialized!
[58273.842717] uvccvideo 1-2.1:1.0: Entity type for entity Extension 6 was not initialized!
[58273.852332] uvccvideo 1-2.1:1.0: Entity type for entity Extension 12 was not initialized!
[58273.861661] uvccvideo 1-2.1:1.0: Entity type for entity Camera 1 was not initialized!
[58273.871047] uvccvideo 1-2.1:1.0: Entity type for entity Extension 8 was not initialized!
[58273.879281] uvccvideo 1-2.1:1.0: Entity type for entity Extension 9 was not initialized!
[58273.887553] uvccvideo 1-2.1:1.0: Entity type for entity Extension 10 was not initialized!
[58273.895789] uvccvideo 1-2.1:1.0: Entity type for entity Extension 11 was not initialized!
[58273.904889] input: HD Pro Webcam C920 as /devices/70090000.xusb/usb1/1-2/1-2.1/1-2.1:1.0/input/input14
[58273.904810] usbcore: registered new interface driver uvccvideo
[58273.904812] USB Video Class driver (1.1.1)
root@poorn-desktop:~#

```

```

root@poorn-desktop:~# dmesg | grep video
[58273.833326] uvcvideo: Found UVC 1.00 device HD Pro Webcam C920 (046d:082d)
[58273.834551] uvcvideo 1-2.1:1.0: Entity type for entity Processing 3 was not initialized!
[58273.842717] uvcvideo 1-2.1:1.0: Entity type for entity Extension 6 was not initialized!
[58273.852332] uvcvideo 1-2.1:1.0: Entity type for entity Extension 12 was not initialized!
[58273.861661] uvcvideo 1-2.1:1.0: Entity type for entity Camera 1 was not initialized!
[58273.871047] uvcvideo 1-2.1:1.0: Entity type for entity Extension 8 was not initialized!
[58273.879281] uvcvideo 1-2.1:1.0: Entity type for entity Extension 9 was not initialized!
[58273.887553] uvcvideo 1-2.1:1.0: Entity type for entity Extension 10 was not initialized!
[58273.895789] uvcvideo 1-2.1:1.0: Entity type for entity Extension 11 was not initialized!
[58273.904810] usbcore: registered new interface driver uvcvideo

```

Description

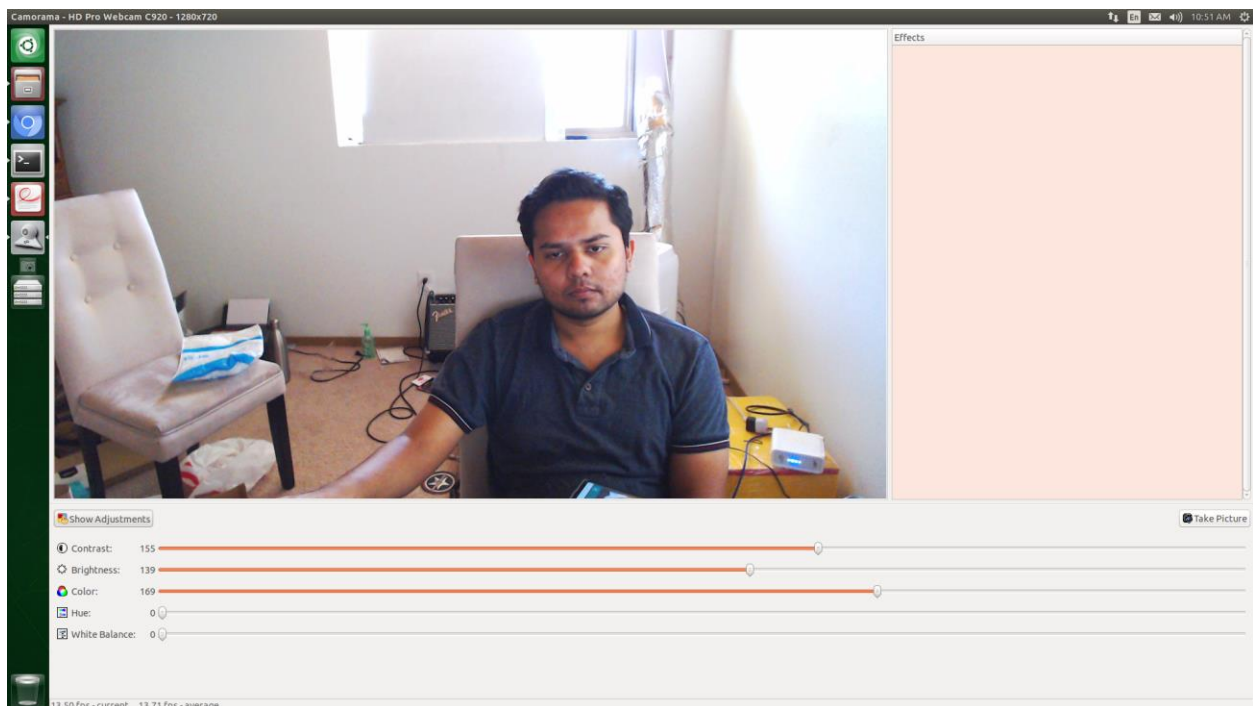
- The command `lsusb` is used to list all connected usb devices to the system
- It's clear from screenshots that as soon as the camera is plugged in, the device shows up in the list of usb devices
- The command `lsmod` is used to list all currently loaded kernel modules
- Most – if not all device drivers are implemented as kernel modules in linux.
- The driver for this camera is listed as 'uvcvideo' under the list of kernel modules
- The command `dmesg` basically prints out all kernel messages (that holds importance, which is determined by the kernel and kernel modules).
- Observing the output at the time of inserting camera shows that the kernel detected the device first while using usb driver (another kernel module), and determined that it can be driven by uvcvideo kernel module/driver.
- Therefore, the latter module is loaded, and initialized.

Q2: If you do not have camorama, do apt-get install camorama on your 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 you need to resolve issues with sudo and your account, research this and please do so.

Solution:

Camorama Screenshots

Playing with various settings



I tried changing various settings (brightness, contrast, color etc.) in Camorama – as well as a few filters.

Camorama captured sample Image



A sample image I captured using webcam and Camorama.

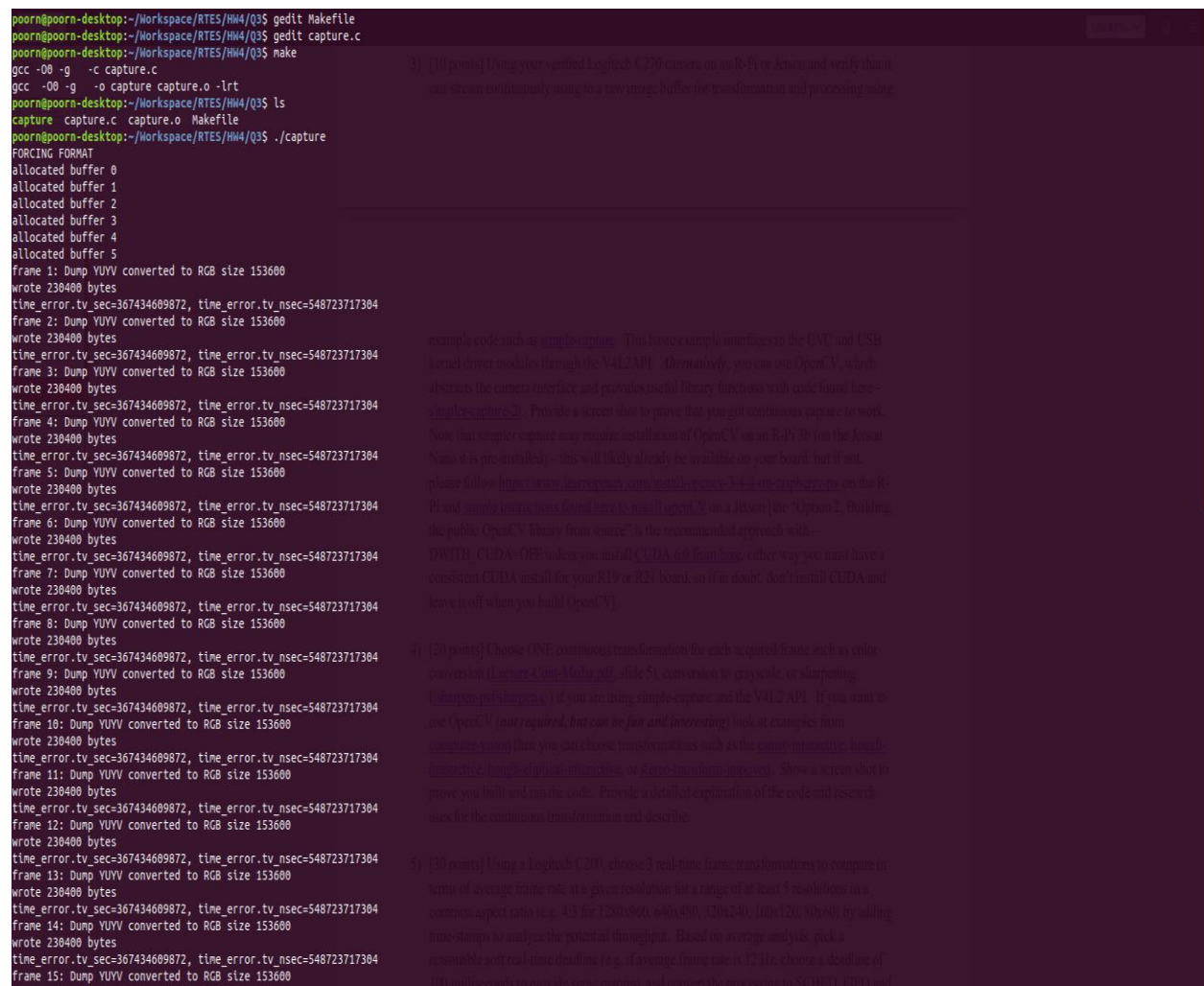
Use of Camorama

- This tool is using a simple library written in C – called V4L2 API.
- It operates by utilizing the UVC video driver
- It was very clear from the use of this tool, that it's quite efficient, and surprisingly light weight.
- The frame rates were quite good, and almost all the effects and filters worked instantly.
- Since this tool uses a clean C interface, many deterministic applications (with soft real time deadlines), can be used using this tool.
- Moreover, the filters and effects can be configured to meet a highly specific application.

Q3: Using your verified Logitech C270 camera on an R-Pi or Jetson and verify that it can stream continuously using to a raw image buffer for transformation and processing using example code such as simple-capture. This basic example interfaces to the UVC and USB kernel driver modules through the V4L2API. Alternatively, you can use OpenCV, which abstracts the camera interface and provides useful library functions with code found here - simpler-capture-2/. Provide a screen shot to prove that you got continuous capture to work. Note that simpler capture may require installation of OpenCV on an R-Pi 3b (on the Jetson Nano it is pre-installed) – this will likely already be available on your board, but if not, please follow <https://www.learnopencv.com/install-opencv-3-4-4-on-raspberry-pi/> on the RPi and simple instructions found here to install openCV on a Jetson [the “Option 2, Building the public OpenCV library from source” is the recommended approach with – DWITH_CUDA=OFF unless you install CUDA 6.0 from here, either way you must have a consistent CUDA install for your R19 or R21 board, so if in doubt, don’t install CUDA and leave it off when you build OpenCV].

Solution:

Screenshots



```
poorn@poorn-desktop:~/Workspace/RTES/HM4/Q3$ gedit Makefile
poorn@poorn-desktop:~/Workspace/RTES/HM4/Q3$ gedit capture.c
poorn@poorn-desktop:~/Workspace/RTES/HM4/Q3$ make
gcc -O0 -g -c capture.c
gcc -O0 -g -o capture capture.o -lrt
poorn@poorn-desktop:~/Workspace/RTES/HM4/Q3$ ls
capture  capture.c  capture.o  Makefile
poorn@poorn-desktop:~/Workspace/RTES/HM4/Q3$ ./capture
FORCING FORMAT
allocated buffer 0
allocated buffer 1
allocated buffer 2
allocated buffer 3
allocated buffer 4
allocated buffer 5
frame 1: Dump YUVV converted to RGB size 153600
wrote 230400 bytes
time_error.tv_sec=367434609872, time_error.tv_nsec=548723717304
frame 2: Dump YUVV converted to RGB size 153600
wrote 230400 bytes
time_error.tv_sec=367434609872, time_error.tv_nsec=548723717304
frame 3: Dump YUVV converted to RGB size 153600
wrote 230400 bytes
time_error.tv_sec=367434609872, time_error.tv_nsec=548723717304
frame 4: Dump YUVV converted to RGB size 153600
wrote 230400 bytes
time_error.tv_sec=367434609872, time_error.tv_nsec=548723717304
frame 5: Dump YUVV converted to RGB size 153600
wrote 230400 bytes
time_error.tv_sec=367434609872, time_error.tv_nsec=548723717304
frame 6: Dump YUVV converted to RGB size 153600
wrote 230400 bytes
time_error.tv_sec=367434609872, time_error.tv_nsec=548723717304
frame 7: Dump YUVV converted to RGB size 153600
wrote 230400 bytes
time_error.tv_sec=367434609872, time_error.tv_nsec=548723717304
frame 8: Dump YUVV converted to RGB size 153600
wrote 230400 bytes
time_error.tv_sec=367434609872, time_error.tv_nsec=548723717304
frame 9: Dump YUVV converted to RGB size 153600
wrote 230400 bytes
time_error.tv_sec=367434609872, time_error.tv_nsec=548723717304
frame 10: Dump YUVV converted to RGB size 153600
wrote 230400 bytes
time_error.tv_sec=367434609872, time_error.tv_nsec=548723717304
frame 11: Dump YUVV converted to RGB size 153600
wrote 230400 bytes
time_error.tv_sec=367434609872, time_error.tv_nsec=548723717304
frame 12: Dump YUVV converted to RGB size 153600
wrote 230400 bytes
time_error.tv_sec=367434609872, time_error.tv_nsec=548723717304
frame 13: Dump YUVV converted to RGB size 153600
wrote 230400 bytes
time_error.tv_sec=367434609872, time_error.tv_nsec=548723717304
frame 14: Dump YUVV converted to RGB size 153600
wrote 230400 bytes
time_error.tv_sec=367434609872, time_error.tv_nsec=548723717304
frame 15: Dump YUVV converted to RGB size 153600
wrote 230400 bytes
```

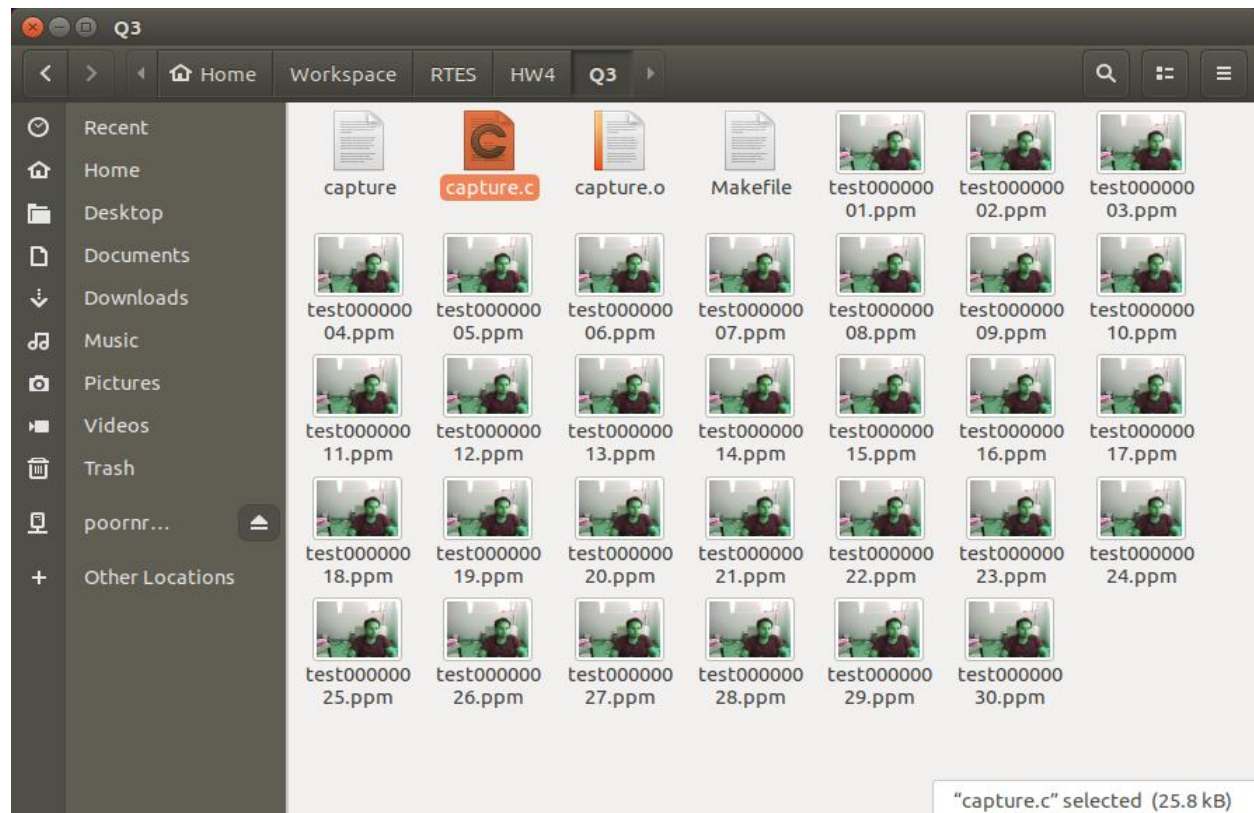
3) (10 points) Using your verified Logitech C270 camera on an R-Pi or Jetson and verify that it can stream continuously using to a raw image buffer for transformation and processing using

example code such as simple-capture. This basic example interfaces to the UVC and USB kernel driver modules through the V4L2API. Alternatively, you can use OpenCV, which abstracts the camera interface and provides useful library functions with code found here - simpler-capture-2/. Provide a screen shot to prove that you got continuous capture to work. Note that simpler capture may require installation of OpenCV on an R-Pi 3b (on the Jetson Nano it is pre-installed) – this will likely already be available on your board, but if not, please follow <https://www.learnopencv.com/install-opencv-3-4-4-on-raspberry-pi/> on the R-Pi and simple instructions found here to install openCV on a Jetson [the “Option 2, Building the public OpenCV library from source” is the recommended approach with – DWITH_CUDA=OFF unless you install CUDA 6.0 from here, either way you must have a consistent CUDA install for your R19 or R21 board, so if in doubt, don’t install CUDA and leave it off when you build OpenCV].

4) (20 points) Choose ONE continuous transformation for each input frame such as color conversion (Luminance/Chroma/Alpha only), alpha 5% conversion to grayscale, or sharpening (Laplace or Gaussian). If you are using simple-capture and the V4L2 API. If you want to use OpenCV (not required, but can be fun and interesting), look at examples from <https://github.com/opencv/opencv/blob/master/samples/cpp/transformations.cpp> that you can choose transformations such as the color-conversion, though, remember, though, alpha-conversion is not stream-transfered. 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 and describe.

5) (30 points) Using a Logitech C270, choose 3 real-time frame transformations to compare in terms of average frame rate at a given resolution for a range of at least 5 resolutions in a common aspect ratio (e.g. 4:3 for 1280x960, 640x480, 320x240, 160x120, 80x60) by adding time-stamps to analyze the potential throughput. 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 ms) and each frame of your program, and log on the screen average FPS (FPS) and

```
poorn@poorn-desktop:~/Workspace/RTES/HM4/Q35 ls
capture test00000001.ppn test00000005.ppn test00000009.ppn test00000013.ppn test00000017.ppn test00000021.ppn test00000025.ppn test00000029.ppn
capture_c test00000002.ppn test00000006.ppn test00000010.ppn test00000014.ppn test00000018.ppn test00000022.ppn test00000026.ppn test00000030.ppn
capture.o test00000003.ppn test00000007.ppn test00000011.ppn test00000015.ppn test00000019.ppn test00000023.ppn test00000027.ppn test00000031.ppn
Makefile test00000004.ppn test00000008.ppn test00000012.ppn test00000016.ppn test00000020.ppn test00000024.ppn test00000028.ppn test00000032.ppn
poorn@poorn-desktop:~/Workspace/RTES/HM4/Q35 gedit test00000001.ppn
Killed
poorn@poorn-desktop:~/Workspace/RTES/HM4/Q35
```



As it is evident from the above screenshots, I've built and ran the simple capture program, without any issues.

Code Description

Please visit [this](#) – to read my comments on the code that I've used as a base in this assignment. Essentially, it's just capture.c program with lots of comments to clearly describe what's going on in what function, and how. Since I have spent a lot of time for clear comments, along with reference links, I won't be describing the same with much detail here.

- From the code snippets provided in reference manual of V4L2 API, the capture.c program was developed
- The basic flow is quite simple: setup, capture, process, and stop.
- Setup contains multiple functionalities to properly initialize the driver, and device – and also to configure it as required. This includes choosing the appropriate settings (for examples, memory mapping, user pointer, no of frames to be buffered etc.) which makes the code quite flexible.
- The capture and processing, is executed a few times in loop – to verify the streaming mode of device.
- For capturing, the basic steps are straightforward: dequeue a filled buffer from driver, pass the pointer of the same to image processing function, and finally to queue back the empty buffer.
- It should be noted that this process, doesn't include exchanging of real frame data – rather only the pointers to buffers are exchanged between the driver and application code.
- This is only valid for memory mapping and user pointer method.
- The user pointer method uses dynamic allocation, but reduces load on driver.
- After the required number of frames have been captured, the device and driver are properly closed. The interface between driver and application is terminated, and the resources are freed.

Streaming Verification

- The driver is tested in streaming mode, along with memory mapping mode.
- It is evident from the captured frames, that the driver was indeed operating under the streaming mode of capture.

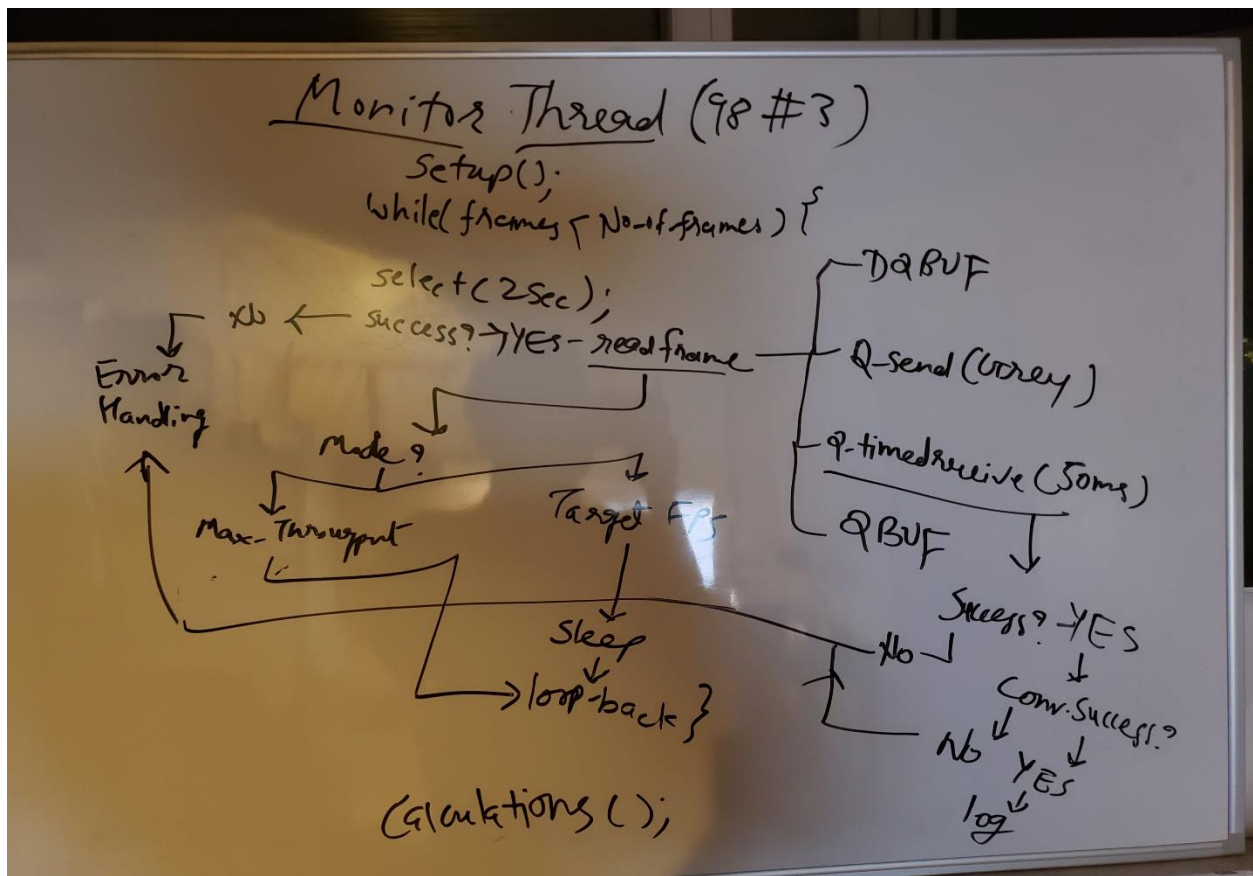
Q4: Choose ONE continuous transformation for each acquired frame such as color conversion (Lecture-Cont-Media.pdf, slide 5), conversion to grayscale, or sharpening (/sharpen-psf/sharpen.c) if you are using simple-capture and the V4L2 API. If you want to use OpenCV (not required, but can be fun and interesting) look at examples from computer-vision then you can choose transformations such as the canny-interactive, houghinteractive, hough-elliptical-interactive, or stereo-transform-improved. 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 and describe.

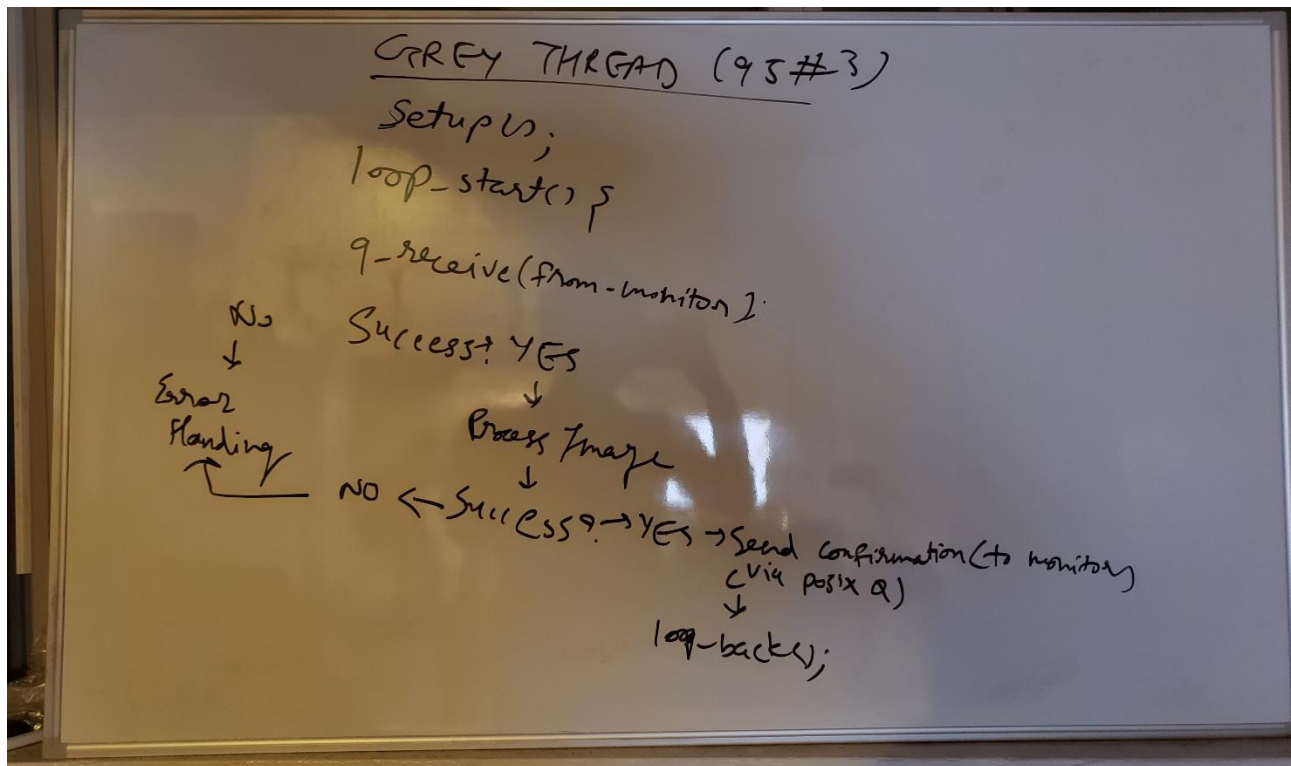
Solution:

Code Description (Greyscale Conversion)

The code I have written, utilizes the base of capture.c and then, extends further to have a specific program architecture (overview in photos provided below). Please note, that this is quite a complex program, especially without comments included in the source files. It was left out because at the time of developing, the architecture was changing rapidly, and it wasn't added at the last since the architecture fundamentals were proven to be incorrect – according to the final project requirement. The overview provided here however, along with code description given in previous question, and the detailed comments in the original code – should be more than enough to understand it. Please find the captured images [here](#).

Architecture Photos





- There are two primary threads of importance here: Monitor, and Grey.
- The monitor thread, executes repetitively (either as fast as possible, or with some sleep to lock on a framerate) – to capture a raw frame.
- It first blocks on select() call, waiting for device to be ready. Once it succeeds, it reads the frame, sends it to Grey thread using POSIX queue, and then waits for a fixed amount of time, to hear back from it.
- Grey thread, block indefinitely on the queue receive on the other hand, waiting for a frame to be made available to it – via Monitor thread.
- Once it receives a frame, it processes on it as fast as it can, and reports back the success/failure to the monitor.
- Based on the data received from Grey thread/timeout in queue receive, the Monitor thread takes appropriate actions (logging, error handling etc).
- After the required number of frames have been captured, the threads exit, and a FPS calculations are made to be logged before terminating the program.

Continuous Transformation Applications

- There is a very huge number of practical uses, of image processing streaming mode/real time. This is done via continuous transformations.
- The raw images captured and provided by camera, are uncompressed - and very large in size. Transferring the same over various communication mediums thus, becomes very difficult.
- For video streaming applications, this is made possible via compressions – which are incorporated in form of continuous image transformations.
- Here, a greyscale conversion is used – which is a very simple transformation. In this, YUYV captured frames are converted to YY frames, removing the color information.
- This is done on each frame, as soon as it is captured, and therefore – is continuous in nature.
- Apart from video streaming, there are plenty of other usages as well: such as object detection, face recognition, image quality enhancer, etc.
- In multitude of research projects, continuous transformations are widely used, to observe the cause and effect in real time – rather than to analyze events, long after they've taken place.
- Clearly, continuous transformation is quite useful.

Q5: Using a Logitech C200, choose 3 real-time frame transformations to compare in terms of average frame rate at a given resolution for a range of at least 5 resolutions in a common aspect ratio (e.g. 4:3 for 1280x960, 640x480, 320x240, 160x120, 80x60) by adding time-stamps to analyze the potential throughput. 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 jitter in the frame rate relative to your deadline

Solution:

Part1 Folder:

https://github.com/Poorn-Mehta/RTES/tree/master/HW4/Q5/My_Sol

Part1 Analysis (output):

https://github.com/Poorn-Mehta/RTES/blob/master/HW4/Q5/My_Sol/src/analysis.txt

Part1 Captured Images:

<https://drive.google.com/open?id=1XGfzxUBldjCvJqAO1gVqzA8oxFPdmE>

Part2 Folder:

https://github.com/Poorn-Mehta/RTES/tree/master/HW4/Q5/My_Sol_2

Part2 Analysis (output):

https://github.com/Poorn-Mehta/RTES/blob/master/HW4/Q5/My_Sol_2/src/analysis.txt

https://github.com/Poorn-Mehta/RTES/blob/master/HW4/Q5/My_Sol_2/src/analysis2.txt

Part2 Jitter Variation Graphs:

https://github.com/Poorn-Mehta/RTES/tree/master/HW4/Q5/My_Sol_2/src/analysis_graphs

Part2 Jitter Variation Files (syslog output, csv, bash script for generating csv):

https://github.com/Poorn-Mehta/RTES/tree/master/HW4/Q5/My_Sol_2/src/analysis_files

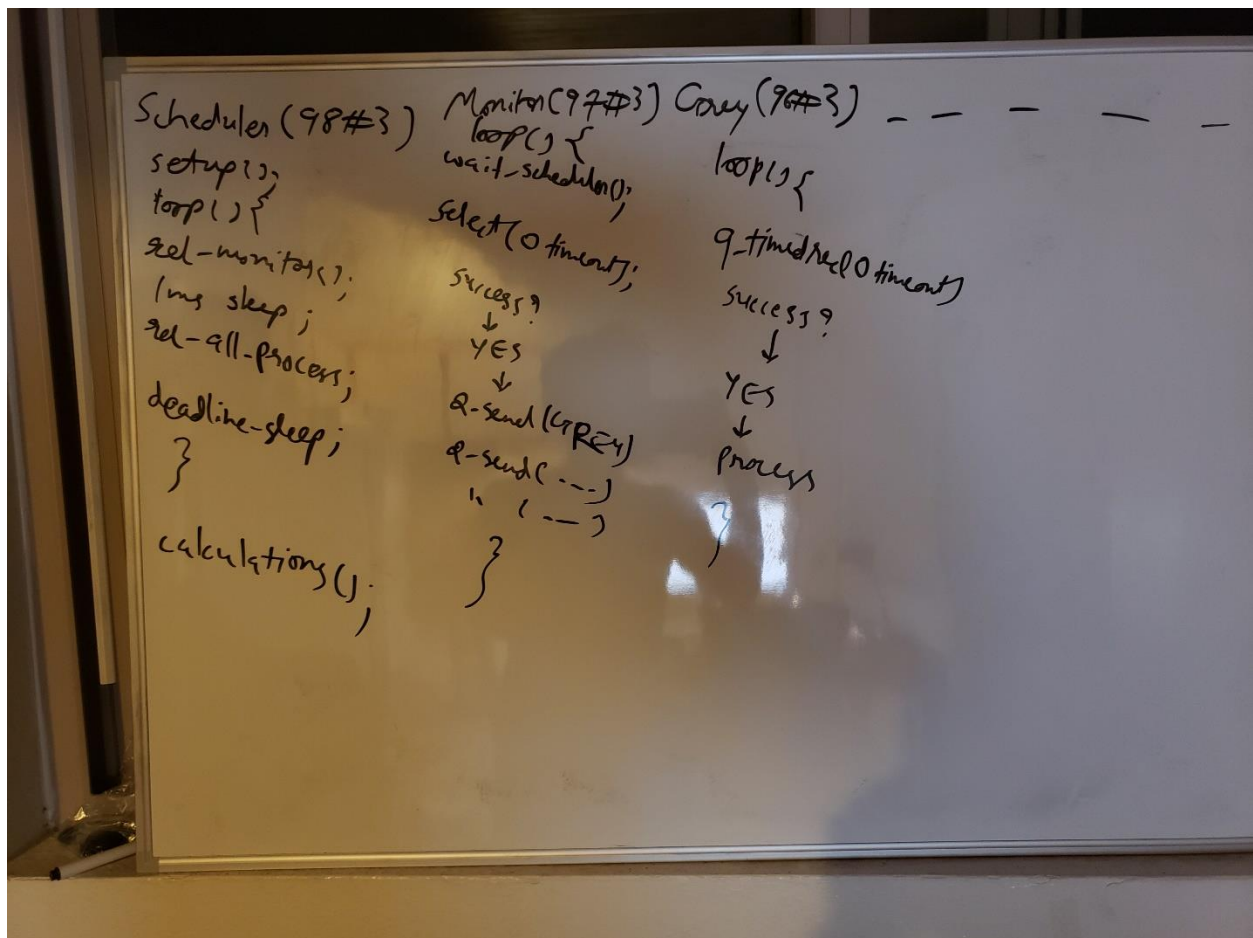
Part2 Captured Images (all methods):

https://drive.google.com/open?id=1b1-sk8NpepI-Q7_mYOcmNZtWCldPJNy-

Part 1:

- The code used here, is simply an extension to that developed in Q4.
- Please go through the analysis.txt file (link above), for the code results.
- It calculates Average FPS, Average Execution Time, and Worst-Case Execution Time for all transforms, under all resolutions.
- These calculations are made for 1000 frames captured, for all cases.
- Keeping in mind the camera's max frame rate (30fps), and the code result (please note that the camera was kept in relatively dark environment, so the analysis here is pessimistic), the soft deadline was chosen to be of 100 ms – for part 2.
- The exact reason for some extremely long execution times is unknown, but the flawed architecture design is suspected. Due to lack of time, I'm discouraged to dig deeper in this obsolete design.

Part 2: Architecture and Code Description



- This architecture is developed after consulting Steve Rizor, and brainstorming the requirements of the final project.
- Currently, it is not working out as per the final project goals – in that, it is missing some deadlines, and the jitter accumulation is there as well. However, the basic functionality is

there, and once the timing issues are fixed with not so major modifications in the architecture, it should be ready for the final project.

- There are 4 threads being managed under RM sequencer/scheduler, all of these running on a single CPU with different priorities.
- The scheduler has the highest priority, which is followed by monitor, grey, brightness, and contrast threads.
- The scheduler is configured to launch all services at fixed interval (directly related to determined deadline), by either using `clock_nanosleep()` – or timer and signals.
- The first service released is monitor – which executes `select()` call with 0 timeout. In case of success, it reads a frame, and sends it to the rest 3 threads using POSIX queues.
- The other 3 services, when released, executes `queue_timedreceive()` with 0 timeout, and in case of a valid message reception, they start their individual processing. They also keep storing each execution times in an array, as well as calculating ‘no frame’ instances as well.
- Once the required number of frames have been captured (1000 in this case), the scheduler performs detailed analysis on jitter, missed deadlines, average fps etc.
- I have created jitter variation graphs for each case, and have stored code output for each case as well. Please refer to the links posted above, to go through the same.

Analysis

- After capturing thousands of frames – of a smartphone stopwatch, and trying out many different ways to deal with the jitter accumulation problems over a long period of time, I’ve reached to following conclusions.
- High resolutions are more prone to missed deadlines and large jitter, than lower resolutions. Naturally, the amount of processing required is playing a key role here, however, there is still something fundamentally wrong with my code – that’s allowing it to happen to a great extent.
- The stopwatch frames aren’t going out of sync for a few tens of captures, however, in thousand frames – the error becomes significant, and easily observable.
- The reason behind this is not particularly known, since I have tried a multitude of things which randomly failed.
- In some cases, I surprisingly got very accurate results – but I was unable to faithfully reproduce the same.
- I will keep on trying to make this much more deterministic, by playing around with following concepts:
 - Removing monitor thread from scheduler, and let it run freely at a higher rate
 - Removing queues, and using shared resources – protected with mutex
 - Trying out various ways to get accurate sleep. Timers, `clock_nanosleep()`, referring to a time stamp taken at start, etc. are in this list.