Electrical, Computer & Energy Engineering
UNIVERSITY OF COLORADO BOULDER

# ECEN 5623 Exercise #4 Real-Time Continuous Media

Course Name: **Real-Time Embedded Systems**
Submission Date: **July 22nd 2019**
Student Name: **Rushi James Macwan**
**Board Used: Raspberry Pi 3B+**

Note: Correct answers are in Blue Font

***Credits:*** *All codes utilized for completing this assignment have been reused and modified accordingly. The codes are originally authored by **Dr. Siewert** and is **NOT** my original work. All code fragments have been reused and duly credited here for its reuse. Appropriate code explanations have been made wherever required with comments.*

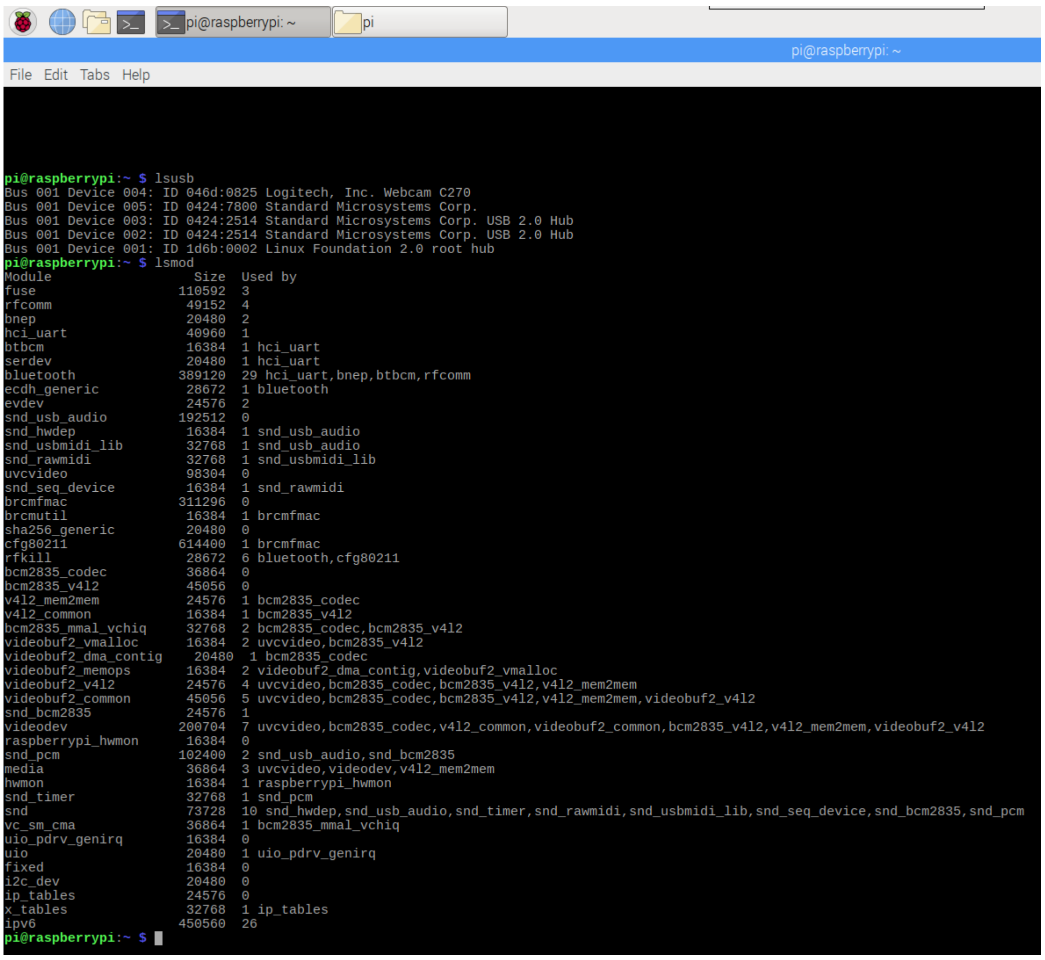
***Please Note:*** The images included in this report may not be very clear and as desired. Please, refer to the attached zipped file which contains all the images provided in this report.

1) [10 points] 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.

   **Sol.**

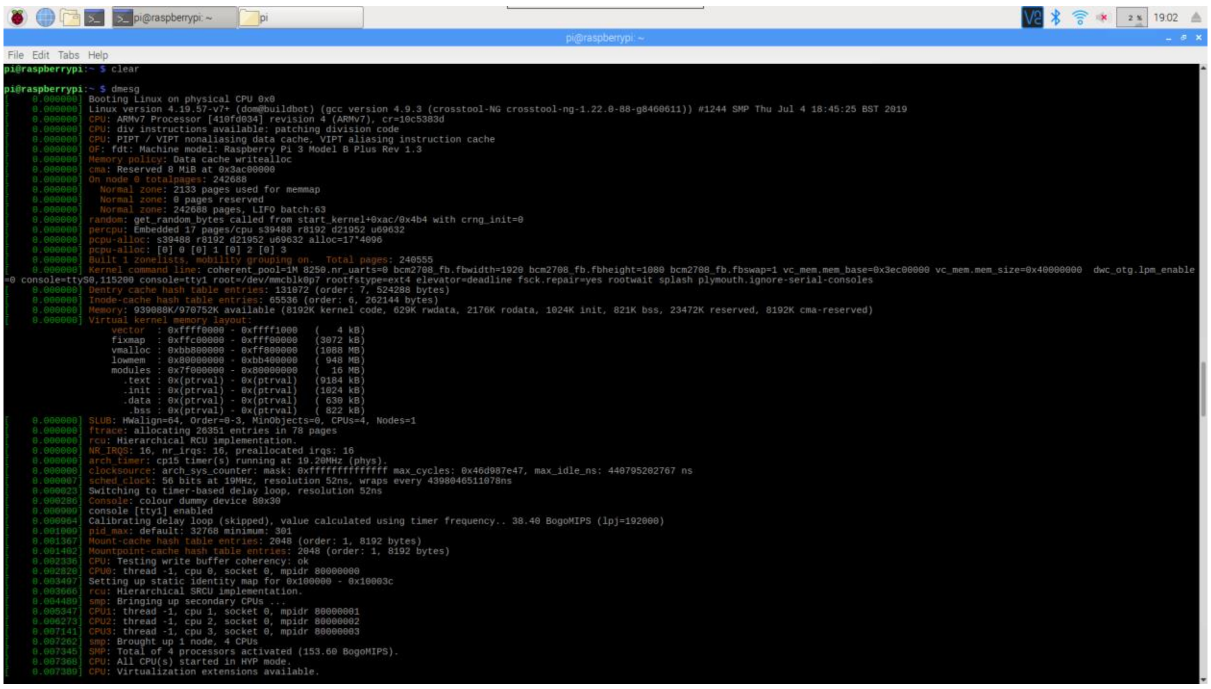
   **USB hot-plug (Detection of Logitech C270 Camera) through CLI:**

   I connected the Logitech C270 USB camera to my Raspberry Pi 3B+. I executed the commands mentioned in the problem on the CLI and the output screenshots pertaining to the commands (for testing the USB hot-plug connection) are as below:



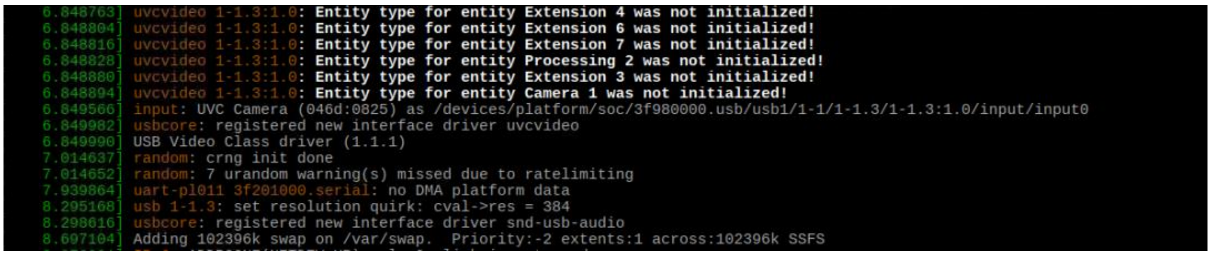
```
pi@raspberrypi:~ $ lsusb
Bus 001 Device 004: ID 046d:0825 Logitech, Inc. Webcam C270
Bus 001 Device 005: ID 0424:7800 Standard Microsystems Corp.
Bus 001 Device 003: ID 0424:2514 Standard Microsystems Corp. USB 2.0 Hub
Bus 001 Device 002: ID 0424:2514 Standard Microsystems Corp. USB 2.0 Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
pi@raspberrypi:~ $ lsmod
Module                  Size  Used by
fuse                  110592  3
rfcomm                 49152  4
bnep                   20480  2
hci_uart               40960  1
btbcm                  16384  1 hci_uart
serdev                 20480  1 hci_uart
bluetooth             389120  29 hci_uart,bnep,btbcm,rfcomm
ecdh_generic           28672  1 bluetooth
evdev                  24576  2
snd_usb_audio         192512  0
snd_hwdep              16384  1 snd_usb_audio
snd_usbmidi_lib        32768  1 snd_usb_audio
snd_rawmidi            32768  1 snd_usbmidi_lib
uvcvideo               98304  0
snd_seq_device         16384  1 snd_rawmidi
brcmfmac              311296  0
brcmutil               16384  1 brcmfmac
sha256_generic         20480  0
cfg80211              614400  1 brcmfmac
rfkill                 28672  6 bluetooth,cfg80211
bcm2835_codec          36864  0
bcm2835_v4l2           45056  0
v4l2_mem2mem           24576  1 bcm2835_codec
v4l2_common            16384  1 bcm2835_v4l2
bcm2835_mmal_vchiq     32768  2 bcm2835_codec,bcm2835_v4l2
videobuf2_vmalloc      16384  2 uvcvideo,bcm2835_v4l2
videobuf2_dma_contig    20480  1 bcm2835_codec
videobuf2_memops       16384  2 videobuf2_dma_contig,videobuf2_vmalloc
videobuf2_v4l2         24576  4 uvcvideo,bcm2835_codec,bcm2835_v4l2,v4l2_mem2mem
videobuf2_common       45056  5 uvcvideo,bcm2835_codec,bcm2835_v4l2,v4l2_mem2mem,videobuf2_v4l2
snd_bcm2835            24576  1
videodev              200704  7 uvcvideo,bcm2835_codec,v4l2_common,videobuf2_common,bcm2835_v4l2,v4l2_mem2mem,videobuf2_v4l2
raspberrypi_hwmon      16384  0
snd_pcm               102400  2 snd_usb_audio,snd_bcm2835
media                  36864  3 uvcvideo,videodev,v4l2_mem2mem
hwmon                  16384  1 raspberrypi_hwmon
snd_timer              32768  1 snd_pcm
snd                    73728  10 snd_hwdep,snd_usb_audio,snd_timer,snd_rawmidi,snd_usbmidi_lib,snd_seq_device,snd_bcm2835,snd_pcm
vc_sm_cma              36864  1 bcm2835_mmal_vchiq
uio_pdrv_genirq        16384  0
uio                    20480  1 uio_pdrv_genirq
fixed                  16384  0
i2c_dev                20480  0
ip_tables              24576  0
x_tables               32768  1 ip_tables
ipv6                  450560  26
pi@raspberrypi:~ $ 
```

*Screenshot-1*



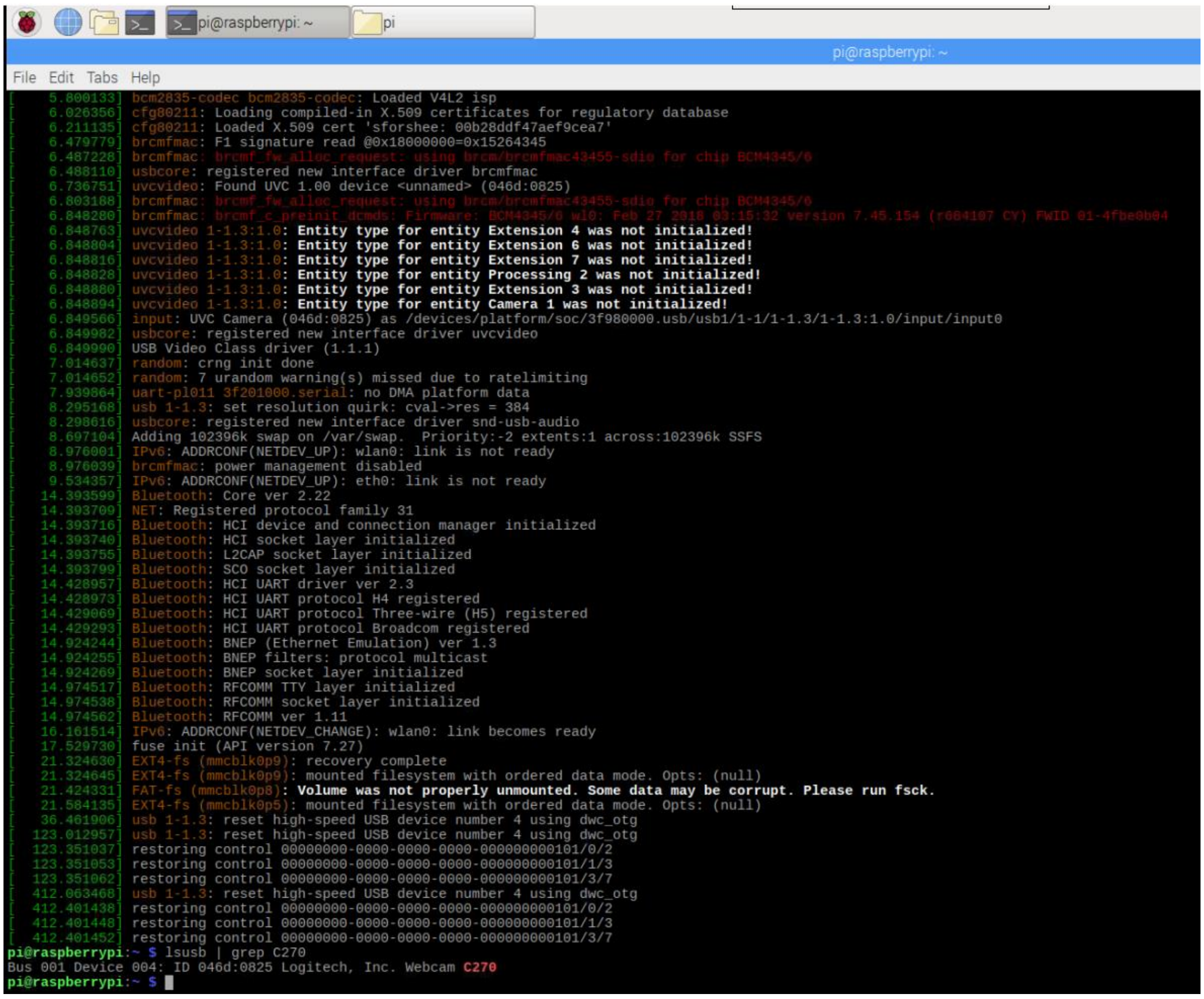
```
pi@raspberrypi:~ $ clear

pi@raspberrypi:~ $ dmesg
[    0.000000] Booting Linux on physical CPU 0x0
[    0.000000] Linux version 4.19.57-v7+ (dom@buildbot) (gcc version 4.9.3 (crosstool-NG crosstool-ng-1.22.0-88-g8460611)) #1244 SMP Thu Jul 4 18:45:25 BST 2019
[    0.000000] CPU: ARMv7 Processor [410fd034] revision 4 (ARMv7), cr=10c5383d
[    0.000000] CPU: div instructions available: patching division code
[    0.000000] CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
[    0.000000] OF: fdt: Machine model: Raspberry Pi 3 Model B Plus Rev 1.3
[    0.000000] Memory policy: Data cache writealloc
[    0.000000] cma: Reserved 8 MiB at 0x3ac00000
[    0.000000] On node 0 totalpages: 242688
[    0.000000]   Normal zone: 2133 pages used for memmap
[    0.000000]   Normal zone: 0 pages reserved
[    0.000000]   Normal zone: 242688 pages, LIFO batch:63
[    0.000000] random: get_random_bytes called from start_kernel+0xac/0x4b4 with crng_init=0
[    0.000000] percpu: Embedded 17 pages/cpu s39488 r8192 d21952 u69632
[    0.000000] pcpu-alloc: s39488 r8192 d21952 u69632 alloc=17*4096
[    0.000000] pcpu-alloc: [0] 0 [0] 1 [0] 2 [0] 3
[    0.000000] Built 1 zonelists, mobility grouping on.  Total pages: 240555
[    0.000000] Kernel command line: coherent_pool=1M 8250.nr_uarts=0 bcm2708_fb.fbwidth=1920 bcm2708_fb.fbheight=1080 bcm2708_fb.fbswap=1 vc_mem.mem_base=0x3ec00000 vc_mem.mem_size=0x40000000  dwc_otg.lpm_enable
=0 console=ttyS0,115200 console=tty1 root=/dev/mmcblk0p7 rootfstype=ext4 elevator=deadline fsck.repair=yes rootwait splash plymouth.ignore-serial-consoles
[    0.000000] Dentry cache hash table entries: 131072 (order: 7, 524288 bytes)
[    0.000000] Inode-cache hash table entries: 65536 (order: 6, 262144 bytes)
[    0.000000] Memory: 939088K/970752K available (8192K kernel code, 629K rwdata, 2176K rodata, 1024K init, 821K bss, 23472K reserved, 8192K cma-reserved)
[    0.000000] Virtual kernel memory layout:
                   vector  : 0xffff0000 - 0xffff1000   (   4 kB)
                   fixmap  : 0xffc00000 - 0xfff00000   (3072 kB)
                   vmalloc : 0xbb800000 - 0xff800000   (1088 MB)
                   lowmem  : 0x80000000 - 0xbb400000   ( 948 MB)
                   modules : 0x7f000000 - 0x80000000   (  16 MB)
                     .text : 0x(ptrval) - 0x(ptrval)   (9184 kB)
                     .init : 0x(ptrval) - 0x(ptrval)   (1024 kB)
                     .data : 0x(ptrval) - 0x(ptrval)   ( 630 kB)
                      .bss : 0x(ptrval) - 0x(ptrval)   ( 822 kB)
[    0.000000] SLUB: HWalign=64, Order=0-3, MinObjects=0, CPUs=4, Nodes=1
[    0.000000] ftrace: allocating 26351 entries in 78 pages
[    0.000000] rcu: Hierarchical RCU implementation.
[    0.000000] NR_IRQS: 16, nr_irqs: 16, preallocated irqs: 16
[    0.000006] arch_timer: cp15 timer(s) running at 19.20MHz (phys).
[    0.000008] clocksource: arch_sys_counter: mask: 0xffffffffffffff max_cycles: 0x46d987e47, max_idle_ns: 440795202767 ns
[    0.000007] sched_clock: 56 bits at 19MHz, resolution 52ns, wraps every 4398046511078ns
[    0.000023] Switching to timer-based delay loop, resolution 52ns
[    0.000286] Console: colour dummy device 80x30
[    0.000909] console [tty1] enabled
[    0.000964] Calibrating delay loop (skipped), value calculated using timer frequency.. 38.40 BogoMIPS (lpj=192000)
[    0.001009] pid_max: default: 32768 minimum: 301
[    0.001367] Mount-cache hash table entries: 2048 (order: 1, 8192 bytes)
[    0.001402] Mountpoint-cache hash table entries: 2048 (order: 1, 8192 bytes)
[    0.002336] CPU: Testing write buffer coherency: ok
[    0.002820] CPU0: thread -1, cpu 0, socket 0, mpidr 80000000
[    0.003497] Setting up static identity map for 0x100000 - 0x10003c
[    0.003666] rcu: Hierarchical SRCU implementation.
[    0.004489] smp: Bringing up secondary CPUs ...
[    0.005347] CPU1: thread -1, cpu 1, socket 0, mpidr 80000001
[    0.006273] CPU2: thread -1, cpu 2, socket 0, mpidr 80000002
[    0.007141] CPU3: thread -1, cpu 3, socket 0, mpidr 80000003
[    0.007262] smp: Brought up 1 node, 4 CPUs
[    0.007345] SMP: Total of 4 processors activated (153.60 BogoMIPS).
[    0.007368] CPU: All CPU(s) started in HYP mode.
[    0.007389] CPU: Virtualization extensions available.
```

*Screenshot-2*



```
  6.848763] uvcvideo 1-1.3:1.0: Entity type for entity Extension 4 was not initialized!
  6.848804] uvcvideo 1-1.3:1.0: Entity type for entity Extension 6 was not initialized!
  6.848816] uvcvideo 1-1.3:1.0: Entity type for entity Extension 7 was not initialized!
  6.848828] uvcvideo 1-1.3:1.0: Entity type for entity Processing 2 was not initialized!
  6.848880] uvcvideo 1-1.3:1.0: Entity type for entity Extension 3 was not initialized!
  6.848894] uvcvideo 1-1.3:1.0: Entity type for entity Camera 1 was not initialized!
  6.849566] input: UVC Camera (046d:0825) as /devices/platform/soc/3f980000.usb/usb1/1-1/1-1.3/1-1.3:1.0/input/input0
  6.849982] usbcore: registered new interface driver uvcvideo
  6.849990] USB Video Class driver (1.1.1)
  7.014637] random: crng init done
  7.014652] random: 7 urandom warning(s) missed due to ratelimiting
  7.939864] uart-pl011 3f201000.serial: no DMA platform data
  8.295168] usb 1-1.3: set resolution quirk: cval->res = 384
  8.298616] usbcore: registered new interface driver snd-usb-audio
  8.697104] Adding 102396k swap on /var/swap.  Priority:-2 extents:1 across:102396k SSFS
```

*Screenshot-3*



```
[    5.800133] bcm2835-codec bcm2835-codec: Loaded V4L2 isp
[    6.026356] cfg80211: Loading compiled-in X.509 certificates for regulatory database
[    6.211135] cfg80211: Loaded X.509 cert 'sforshee: 00b28ddf47aef9cea7'
[    6.479779] brcmfmac: F1 signature read @0x18000000=0x15264345
[    6.487228] brcmfmac: brcmf_fw_alloc_request: using brcm/brcmfmac43455-sdio for chip BCM4345/6
[    6.488110] usbcore: registered new interface driver brcmfmac
[    6.736751] uvcvideo: Found UVC 1.00 device <unnamed> (046d:0825)
[    6.803188] brcmfmac: brcmf_fw_alloc_request: using brcm/brcmfmac43455-sdio for chip BCM4345/6
[    6.848280] brcmfmac: brcmf_c_preinit_dcmds: Firmware: BCM4345/6 wl0: Feb 27 2018 03:15:32 version 7.45.154 (r684107 CY) FWID 01-4fbe0b04
[    6.848763] uvcvideo 1-1.3:1.0: Entity type for entity Extension 4 was not initialized!
[    6.848804] uvcvideo 1-1.3:1.0: Entity type for entity Extension 6 was not initialized!
[    6.848816] uvcvideo 1-1.3:1.0: Entity type for entity Extension 7 was not initialized!
[    6.848828] uvcvideo 1-1.3:1.0: Entity type for entity Processing 2 was not initialized!
[    6.848880] uvcvideo 1-1.3:1.0: Entity type for entity Extension 3 was not initialized!
[    6.848894] uvcvideo 1-1.3:1.0: Entity type for entity Camera 1 was not initialized!
[    6.849566] input: UVC Camera (046d:0825) as /devices/platform/soc/3f980000.usb/usb1/1-1/1-1.3/1-1.3:1.0/input/input0
[    6.849982] usbcore: registered new interface driver uvcvideo
[    6.849990] USB Video Class driver (1.1.1)
[    7.014637] random: crng init done
[    7.014652] random: 7 urandom warning(s) missed due to ratelimiting
[    7.939864] uart-pl011 3f201000.serial: no DMA platform data
[    8.295168] usb 1-1.3: set resolution quirk: cval->res = 384
[    8.298616] usbcore: registered new interface driver snd-usb-audio
[    8.697104] Adding 102396k swap on /var/swap.  Priority:-2 extents:1 across:102396k SSFS
[    8.976001] IPv6: ADDRCONF(NETDEV_UP): wlan0: link is not ready
[    8.976039] brcmfmac: power management disabled
[    9.534357] IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
[   14.393599] Bluetooth: Core ver 2.22
[   14.393709] NET: Registered protocol family 31
[   14.393716] Bluetooth: HCI device and connection manager initialized
[   14.393740] Bluetooth: HCI socket layer initialized
[   14.393755] Bluetooth: L2CAP socket layer initialized
[   14.393799] Bluetooth: SCO socket layer initialized
[   14.428957] Bluetooth: HCI UART driver ver 2.3
[   14.428973] Bluetooth: HCI UART protocol H4 registered
[   14.429069] Bluetooth: HCI UART protocol Three-wire (H5) registered
[   14.429293] Bluetooth: HCI UART protocol Broadcom registered
[   14.924244] Bluetooth: BNEP (Ethernet Emulation) ver 1.3
[   14.924255] Bluetooth: BNEP filters: protocol multicast
[   14.924269] Bluetooth: BNEP socket layer initialized
[   14.974517] Bluetooth: RFCOMM TTY layer initialized
[   14.974538] Bluetooth: RFCOMM socket layer initialized
[   14.974562] Bluetooth: RFCOMM ver 1.11
[   16.161514] IPv6: ADDRCONF(NETDEV_CHANGE): wlan0: link becomes ready
[   17.529730] fuse init (API version 7.27)
[   21.324630] EXT4-fs (mmcblk0p9): recovery complete
[   21.324645] EXT4-fs (mmcblk0p9): mounted filesystem with ordered data mode. Opts: (null)
[   21.424331] FAT-fs (mmcblk0p8): Volume was not properly unmounted. Some data may be corrupt. Please run fsck.
[   21.584135] EXT4-fs (mmcblk0p5): mounted filesystem with ordered data mode. Opts: (null)
[   36.461906] usb 1-1.3: reset high-speed USB device number 4 using dwc_otg
[  123.012957] usb 1-1.3: reset high-speed USB device number 4 using dwc_otg
[  123.351037] restoring control 00000000-0000-0000-0000-000000000101/0/2
[  123.351053] restoring control 00000000-0000-0000-0000-000000000101/1/3
[  123.351062] restoring control 00000000-0000-0000-0000-000000000101/3/7
[  412.063468] usb 1-1.3: reset high-speed USB device number 4 using dwc_otg
[  412.401438] restoring control 00000000-0000-0000-0000-000000000101/0/2
[  412.401448] restoring control 00000000-0000-0000-0000-000000000101/1/3
[  412.401452] restoring control 00000000-0000-0000-0000-000000000101/3/7
pi@raspberrypi:~ $ lsusb | grep C270
Bus 001 Device 004: ID 046d:0825 Logitech, Inc. Webcam C270
pi@raspberrypi:~ $ 
```

*Screenshot-4*

From the above command executions, the following observations are made:

1. By running the "**`lsusb`**" command, the CLI lists all the USB buses in the system and the devices connected to them. As Linux man page describes it, ""**`lsusb`**" is a utility for displaying information about buses in the system and the devices connected to these buses." After running the command, lists the Logitech Webcam 270 identified by its bus number (001), device number (004) and ID (046d:0825). This proves that the system acknowledges and detects the presence of the camera. ***Screenshot-1*** portrays the above mentioned information.

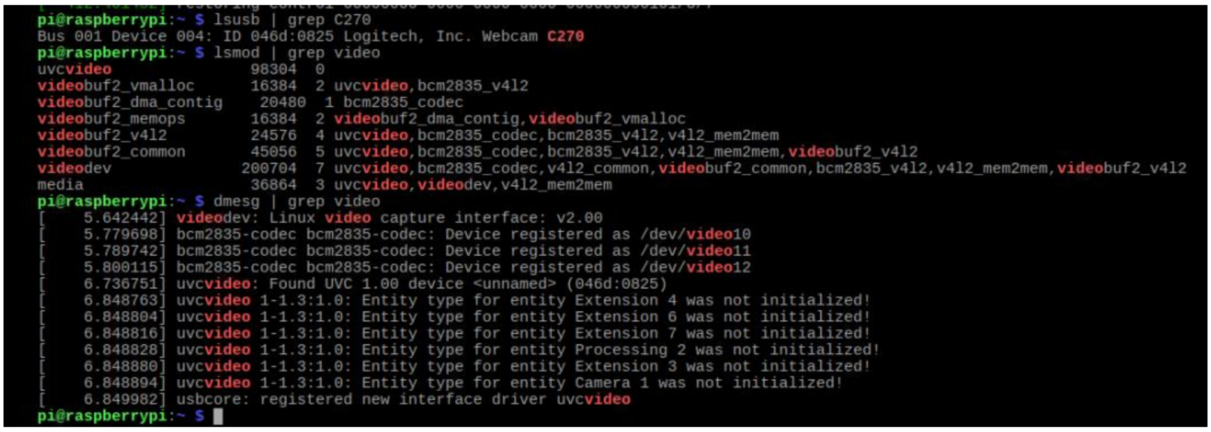
2. By running the "**`lsmod`**" command, the CLI lists the kernel modules that are loaded when the command was executed. As Linux man page describes it, ""**`lsmod`**" shows the status of modules in the Linux

Kernel. As can be read in the screenshot, some of the kernel modules are being used by the UVC video V4L2 API. It showcases that the system detects a device that provide the support of image and video acquisition. ***Screenshot-2*** portrays the execution and output of the command on the CLI.

3. By running the "**`dmesg`**" command, as Linux man page describes it, the CLI "prints the kernel ring buffer which displays all messages from the kernel ring buffer". ***Screenshot-3*** showcases that the UVC camera bearing the same ID (046d:0825) is detected and registered with a video driver (video0) and the UVC driver furnishes the requirements.
4. By running the "**`lsusb | grep C270`**" command, the CLI lists the USB buses in the system and devices connected to them while finding for the term "**`C270`**" in the results received. The CLI displays only the information that contains the term "**`C270`**". Based on what was just mentioned, ***Screenshot-4*** showcases that the Logitech Camera **C270** is detected by the system and appears on the CLI as a response to the command.

## UVC driver verify through CLI:

To verify that the UVC (USB Video Class) driver is loaded successfully in the system, the "**`lsmod | grep video`**" command is executed through the command line. The output shows that the V4L2 API along with the UVC driver has been successfully loaded. This is because the "**`lsmod`**" command grabs the kernels in progress that are associated with the V4L2 API of the UVC buffers required for capturing frames for image and video acquisition of the C270 camera. The screenshot shows the five buffers required for the execution of the simple-capture code and they are associated with the V4L2 API which implies that the driver is loaded into the system. ***Screenshot-5*** showcases that the UVC driver is loaded as it successfully appears as a response on the CLI. The "**`dmesg | grep video`**" searches for the term video in the kernel ring buffer and the screenshot showcases that the UVC device with the same ID (046d:0825) (i.e. Logitech C270 camera) is found (and therefore detected by the system).



```
pi@raspberrypi:~ $ lsusb | grep C270
Bus 001 Device 004: ID 046d:0825 Logitech, Inc. Webcam C270
pi@raspberrypi:~ $ lsmod | grep video
uvcvideo               98304  0
videobuf2_vmalloc      16384  2 uvcvideo,bcm2835_v4l2
videobuf2_dma_contig    20480  1 bcm2835_codec
videobuf2_memops       16384  2 videobuf2_dma_contig,videobuf2_vmalloc
videobuf2_v4l2         24576  4 uvcvideo,bcm2835_codec,bcm2835_v4l2,v4l2_mem2mem
videobuf2_common       45056  5 uvcvideo,bcm2835_codec,bcm2835_v4l2,v4l2_mem2mem,videobuf2_v4l2
videodev              200704  7 uvcvideo,bcm2835_codec,v4l2_common,videobuf2_common,bcm2835_v4l2,v4l2_mem2mem,videobuf2_v4l2
media                  36864  3 uvcvideo,videodev,v4l2_mem2mem
pi@raspberrypi:~ $ dmesg | grep video
[    5.642442] videodev: Linux video capture interface: v2.00
[    5.779698] bcm2835-codec bcm2835-codec: Device registered as /dev/video10
[    5.789742] bcm2835-codec bcm2835-codec: Device registered as /dev/video11
[    5.800115] bcm2835-codec bcm2835-codec: Device registered as /dev/video12
[    6.736751] uvcvideo: Found UVC 1.00 device <unnamed> (046d:0825)
[    6.848763] uvcvideo 1-1.3:1.0: Entity type for entity Extension 4 was not initialized!
[    6.848804] uvcvideo 1-1.3:1.0: Entity type for entity Extension 6 was not initialized!
[    6.848816] uvcvideo 1-1.3:1.0: Entity type for entity Extension 7 was not initialized!
[    6.848828] uvcvideo 1-1.3:1.0: Entity type for entity Processing 2 was not initialized!
[    6.848880] uvcvideo 1-1.3:1.0: Entity type for entity Extension 3 was not initialized!
[    6.848894] uvcvideo 1-1.3:1.0: Entity type for entity Camera 1 was not initialized!
[    6.849982] usbcore: registered new interface driver uvcvideo
pi@raspberrypi:~ $ 
```

***Screenshot-5***

**Courtesy:** Links – [1] [2] [3]

2) [10 points] 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.

**Sol.**

**Use of tools such as *Camorama*:**

Camorama is an interactive tool supported by the V4L2 (Video4Linux2) API. This tool is helpful in connecting a Linux system with a video and image acquisition system (e.g. the Logitech C270 camera). With the help of this tool, a Linux user can obtain frames of videos (and images) at a desired rate (provided the camera supports the desired frame rate with the set aspect-ratio and pixel resolution). The tool can provide soft-RT service of capturing frames of videos in RT. It also provides the user with the capability to process images in RT like – changing various constraints like color, contrast, brightness, hue, white balance, etc.

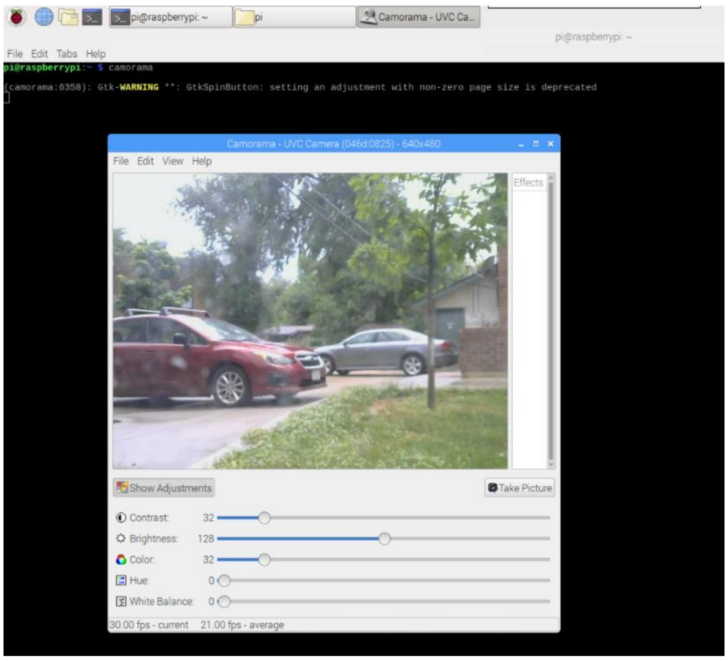
Tools like Camorama are very helpful in the following ways:

1. Provides a way to connect with image and video acquisition devices.
2. It allows the user to interact in SRT with the frames of videos that the system acquires from the device based on reliable constraints (i.e. frame rate, aspect ratio, pixel resolution, image pre-processing routines, etc.)
3. Such tools also provide a way to test an image or video acquisition device for its reliability and responsiveness.
4. It provides a video capture and output interface along with video overlay interface that directly fetches the video without potentially saving it on the system. This can be related to the effective communication tools used these days like Zoom & Skype that provide SRT video capture and output interfaces along with audio support.
5. Such SRT tools (like Camorama) allow the system to connect with other radio tuner devices during the blanking time of the video capture and output interface which ultimately provides services like Zoom & Skype.

**Verification:**

As can be observed from the screenshots below, with the use of the command **`"Camorama"`**, the CLI opens up the Camorama tool which looks as shown in the screenshots. The tool provides a GUI to the user to modify the frame pre-processing constraints like color level, brightness, hue level, white balance, contrast, etc. In addition to that, the tool provides a way to change other frame constraints like the pixel resolution.

***Screenshot-6*** shows the initial Camorama view. ***Screenshot-7*** shows the changes made to the parameters on the GUI. ***Screenshots 8 & 9*** simply contain the captured image files.

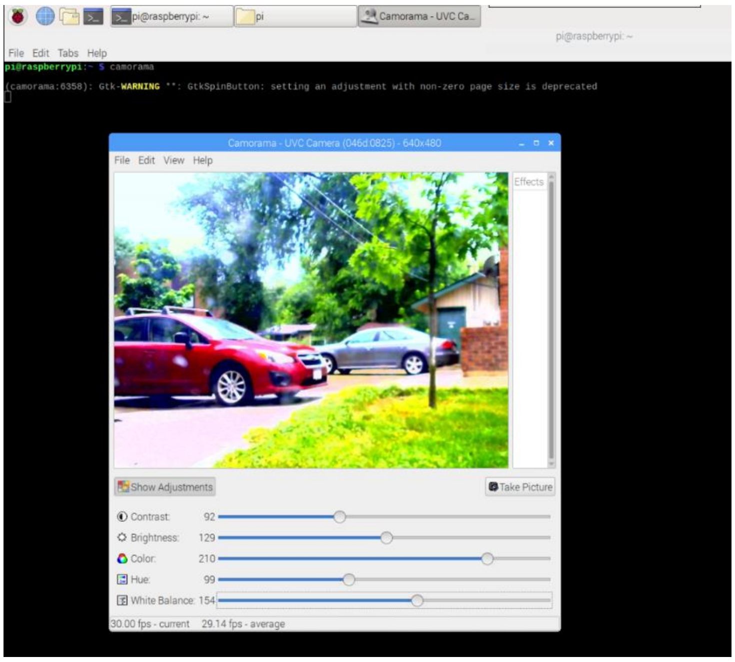


```
pi@raspberrypi:~ $ camorama

(camorama:6358): Gtk-WARNING **: GtkSpinButton: setting an adjustment with non-zero page size is deprecated
```

Camorama - UVC Camera (046d:0825) - 640x480

```
Contrast:       32
Brightness:    128
Color:          32
Hue:             0
White Balance:   0
30.00 fps - current   21.00 fps - average
```

*Screenshot-6*

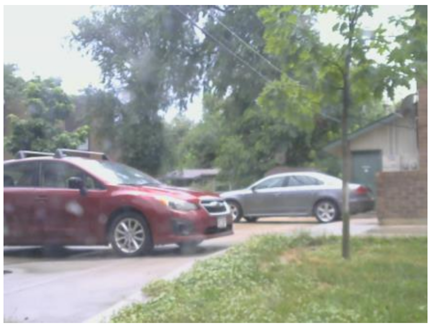


```
pi@raspberrypi:~ $ camorama

(camorama:6358): Gtk-WARNING **: GtkSpinButton: setting an adjustment with non-zero page size is deprecated
```

Camorama - UVC Camera (046d:0825) - 640x480

```
Contrast:       92
Brightness:    129
Color:         210
Hue:            99
White Balance: 154
30.00 fps - current   29.14 fps - average
```

*Screenshot-7*



***Screenshot-8***

***Screenshot-9***

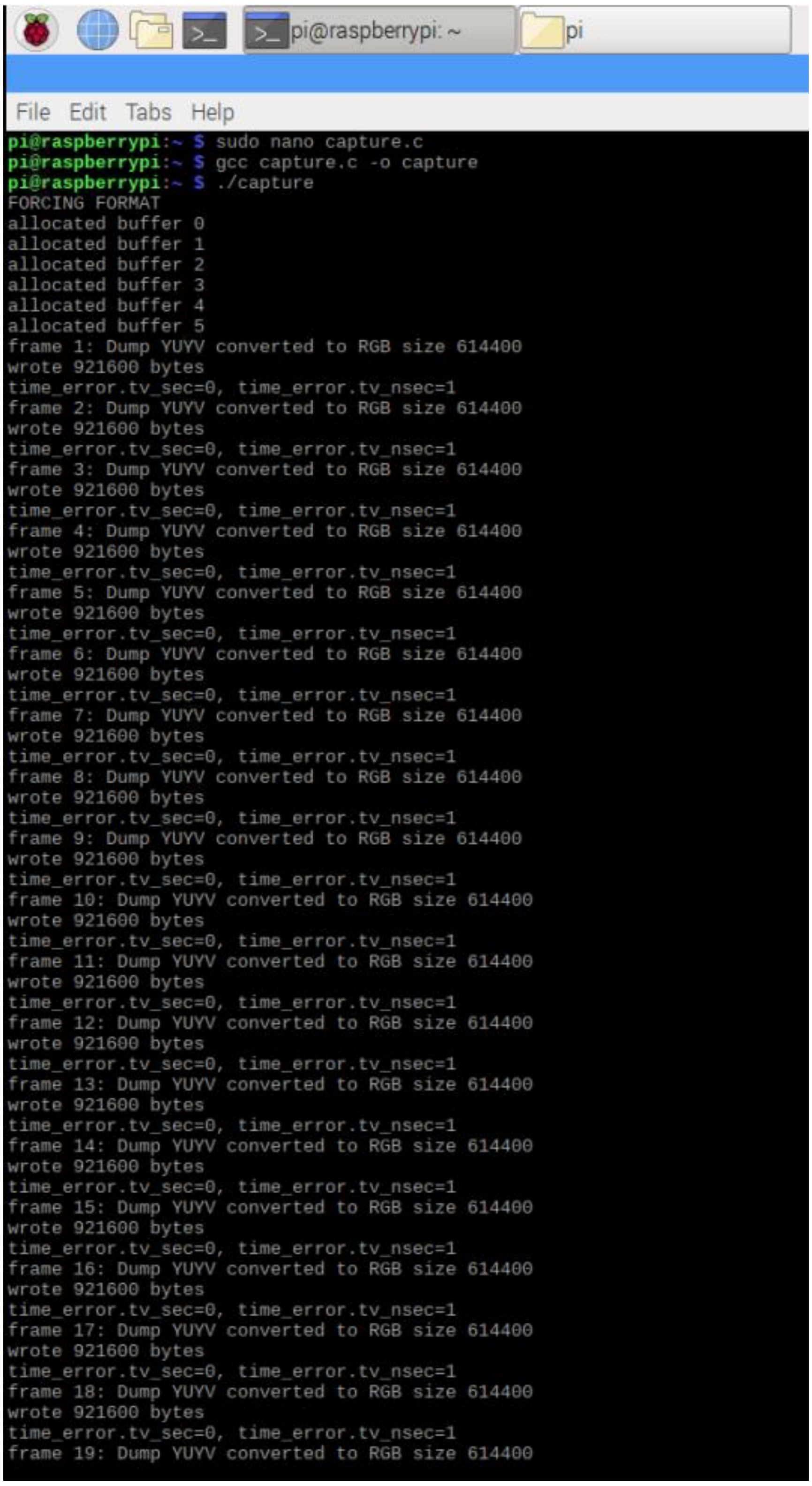
**Courtesy:** Links – [1]

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 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].

**Sol.**

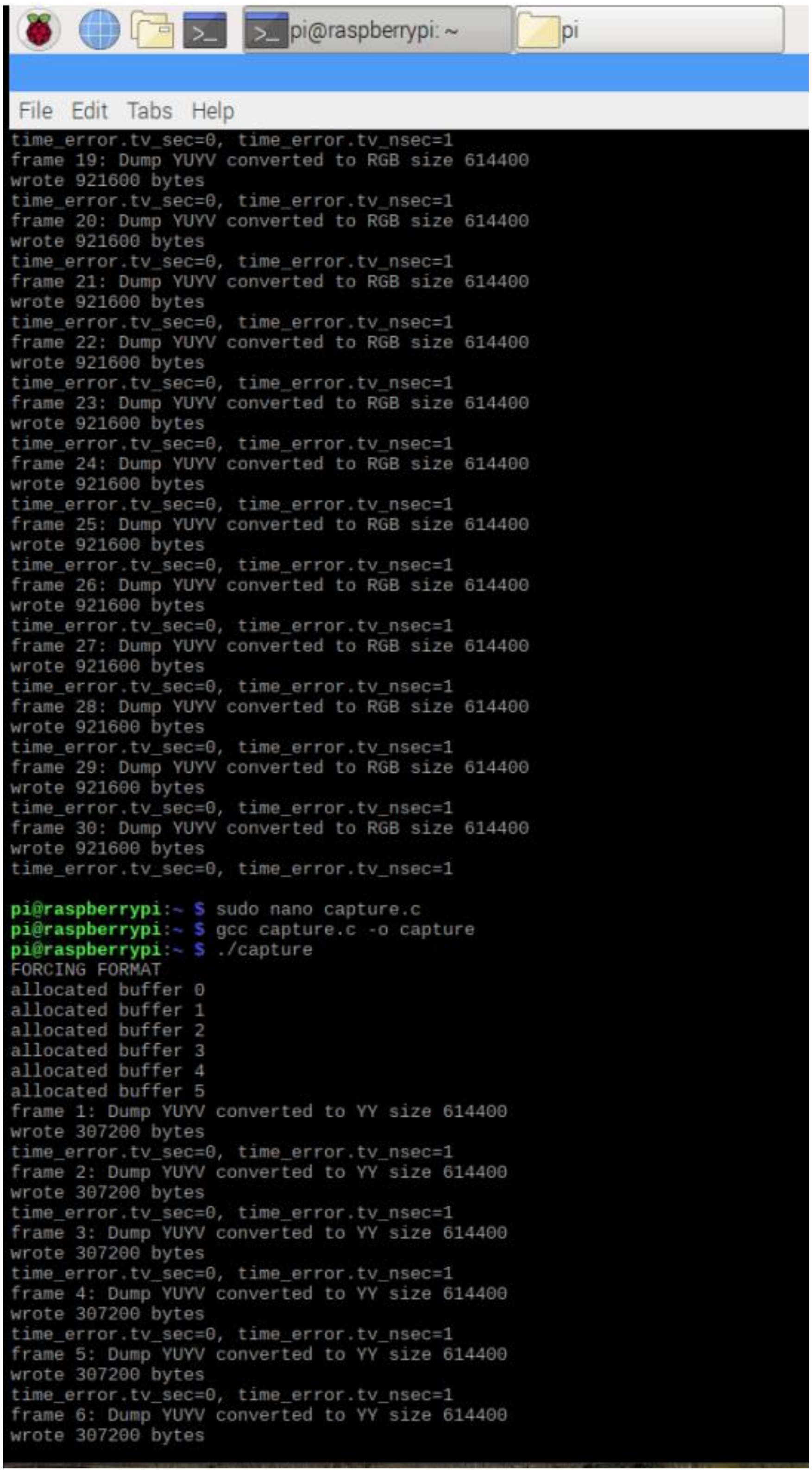
**Build and Test:**

Please, refer to the attached zipped folder for Prob #3 folder which contains the capture.c and capture (exe) files inside the codebase subfolder. The code execution screenshots are provided below:



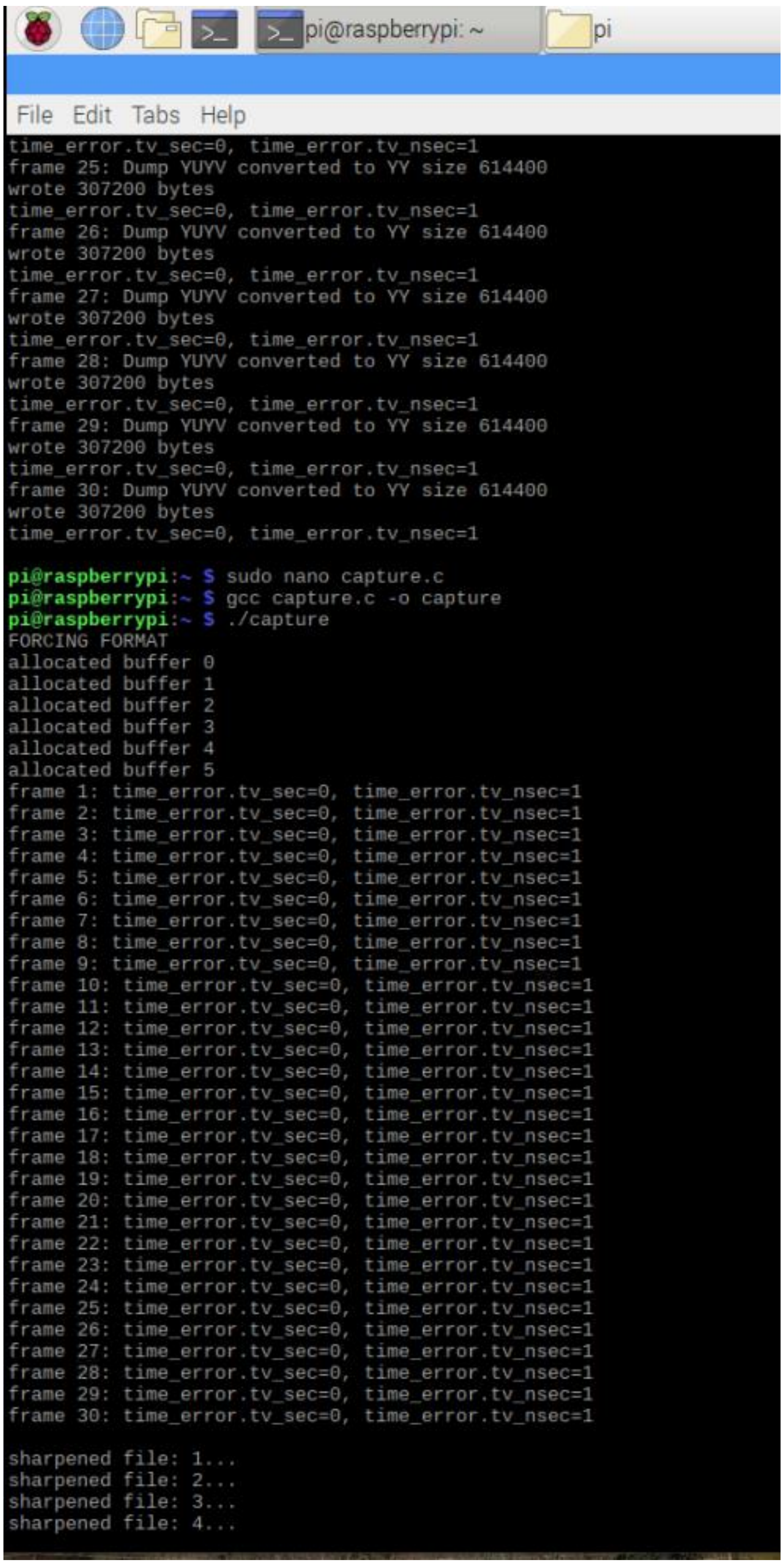
```
pi@raspberrypi:~ $ sudo nano capture.c
pi@raspberrypi:~ $ gcc capture.c -o capture
pi@raspberrypi:~ $ ./capture
FORCING FORMAT
allocated buffer 0
allocated buffer 1
allocated buffer 2
allocated buffer 3
allocated buffer 4
allocated buffer 5
frame 1: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 2: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 3: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 4: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 5: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 6: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 7: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 8: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 9: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 10: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 11: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 12: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 13: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 14: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 15: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 16: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 17: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 18: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 19: Dump YUYV converted to RGB size 614400
```

*Screenshot-10*



```
time_error.tv_sec=0, time_error.tv_nsec=1
frame 19: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 20: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 21: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 22: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 23: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 24: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 25: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 26: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 27: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 28: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 29: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 30: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1

pi@raspberrypi:~ $ sudo nano capture.c
pi@raspberrypi:~ $ gcc capture.c -o capture
pi@raspberrypi:~ $ ./capture
FORCING FORMAT
allocated buffer 0
allocated buffer 1
allocated buffer 2
allocated buffer 3
allocated buffer 4
allocated buffer 5
frame 1: Dump YUYV converted to YY size 614400
wrote 307200 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 2: Dump YUYV converted to YY size 614400
wrote 307200 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 3: Dump YUYV converted to YY size 614400
wrote 307200 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 4: Dump YUYV converted to YY size 614400
wrote 307200 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 5: Dump YUYV converted to YY size 614400
wrote 307200 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 6: Dump YUYV converted to YY size 614400
wrote 307200 bytes
```

*Screenshot-11*



```
time_error.tv_sec=0, time_error.tv_nsec=1
frame 25: Dump YUYV converted to YY size 614400
wrote 307200 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 26: Dump YUYV converted to YY size 614400
wrote 307200 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 27: Dump YUYV converted to YY size 614400
wrote 307200 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 28: Dump YUYV converted to YY size 614400
wrote 307200 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 29: Dump YUYV converted to YY size 614400
wrote 307200 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 30: Dump YUYV converted to YY size 614400
wrote 307200 bytes
time_error.tv_sec=0, time_error.tv_nsec=1

pi@raspberrypi:~ $ sudo nano capture.c
pi@raspberrypi:~ $ gcc capture.c -o capture
pi@raspberrypi:~ $ ./capture
FORCING FORMAT
allocated buffer 0
allocated buffer 1
allocated buffer 2
allocated buffer 3
allocated buffer 4
allocated buffer 5
frame 1: time_error.tv_sec=0, time_error.tv_nsec=1
frame 2: time_error.tv_sec=0, time_error.tv_nsec=1
frame 3: time_error.tv_sec=0, time_error.tv_nsec=1
frame 4: time_error.tv_sec=0, time_error.tv_nsec=1
frame 5: time_error.tv_sec=0, time_error.tv_nsec=1
frame 6: time_error.tv_sec=0, time_error.tv_nsec=1
frame 7: time_error.tv_sec=0, time_error.tv_nsec=1
frame 8: time_error.tv_sec=0, time_error.tv_nsec=1
frame 9: time_error.tv_sec=0, time_error.tv_nsec=1
frame 10: time_error.tv_sec=0, time_error.tv_nsec=1
frame 11: time_error.tv_sec=0, time_error.tv_nsec=1
frame 12: time_error.tv_sec=0, time_error.tv_nsec=1
frame 13: time_error.tv_sec=0, time_error.tv_nsec=1
frame 14: time_error.tv_sec=0, time_error.tv_nsec=1
frame 15: time_error.tv_sec=0, time_error.tv_nsec=1
frame 16: time_error.tv_sec=0, time_error.tv_nsec=1
frame 17: time_error.tv_sec=0, time_error.tv_nsec=1
frame 18: time_error.tv_sec=0, time_error.tv_nsec=1
frame 19: time_error.tv_sec=0, time_error.tv_nsec=1
frame 20: time_error.tv_sec=0, time_error.tv_nsec=1
frame 21: time_error.tv_sec=0, time_error.tv_nsec=1
frame 22: time_error.tv_sec=0, time_error.tv_nsec=1
frame 23: time_error.tv_sec=0, time_error.tv_nsec=1
frame 24: time_error.tv_sec=0, time_error.tv_nsec=1
frame 25: time_error.tv_sec=0, time_error.tv_nsec=1
frame 26: time_error.tv_sec=0, time_error.tv_nsec=1
frame 27: time_error.tv_sec=0, time_error.tv_nsec=1
frame 28: time_error.tv_sec=0, time_error.tv_nsec=1
frame 29: time_error.tv_sec=0, time_error.tv_nsec=1
frame 30: time_error.tv_sec=0, time_error.tv_nsec=1

sharpened file: 1...
sharpened file: 2...
sharpened file: 3...
sharpened file: 4...
```

*Screenshot-12*

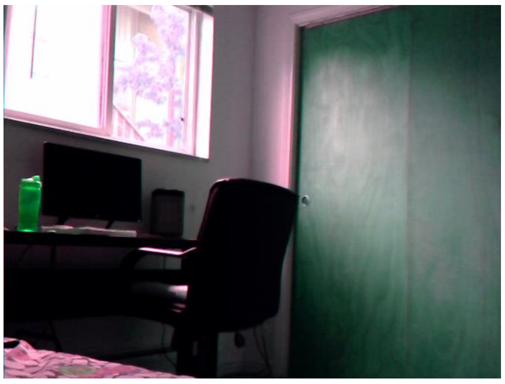
**Code Execution Explanation:**

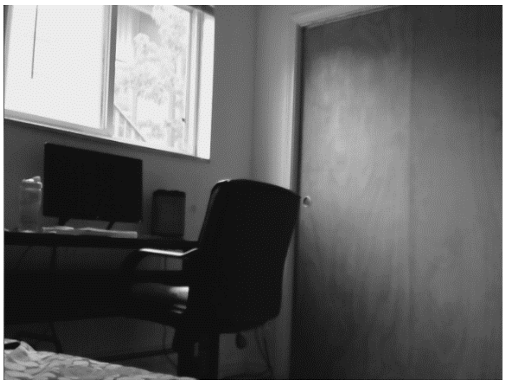
In the above three screenshots, I have executed the capture.c code with three different configurations defined by the #define statements in the code. The execution is in the order: Color conversion, Grayscale conversion and Image sharpening. The Image sharpening execution requires significantly more time for its execution as it reads the files stored by the Color conversion execution, later processes it for sharpening, and then stores to a file with the specified name. The code execution is based on a frame rate of 30fps. The images created by the code executions have been provided in the zipped subfolder named Processed Images. For the sharpened files, I have executed only 6 frames (because of the time it takes to process all 30 frames). But, the codes execute with any errors or warnings.

The code execution shows that initially the camera driver is initialized and the device is opened and memory buffers are allocated. Once that is successfully done, frames are read at the specified frame and resolution (of the specified aspect ratio). Acquired frames are processed according to the specified transformation and are later dumped into a file which can be a .ppm file for RGB (color) transformation and .pgm file for grayscale transformation. The sharpen transformation simply reads .ppm files, processes them and stores the sharpened (processed) images to a new .ppm file with a different name structure.

***The above execution was performed after enabling the "PRINTF" statements. By default, the code provided in the zipped folder, performs color transformation for 30 frames and ONLY uses "SYSLOG".***

Below, I have provided three screenshots (for three different transformations) as a proof that the code execution is successful. The same screenshots with 30 frames are provided in the zipped folder.


***Color Transformation***


***Grayscale Transformation***

***Sharpening Transformation***

**Streaming Verification:**

Most of the explanation pertaining to the streaming of continuous media is already explained in the previous section of this response. In addition to whatever was previously mentioned, streaming is verified through two ways: the successful representation of the code execution on the CLI (without any errors / warnings) and the successful acquisition of 30 frames for the transformations.

Once the camera device is initialized, video capturing (i.e. frame capturing) takes place as mentioned previously and after all the 30 frames have been acquired, the code stores all the frames with the prescribed image file format in the **/home/pi/** directory of the user. The code will then un-initialize the camera and terminate its connection with the video0 device (i.e. the camera C270).

***Please, refer to the set of processed images available in the zipped folder which showcase and verify the streaming feature.***

4) [20 points] 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-eliptical-interactive, or stereo-transform-impoved. 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.

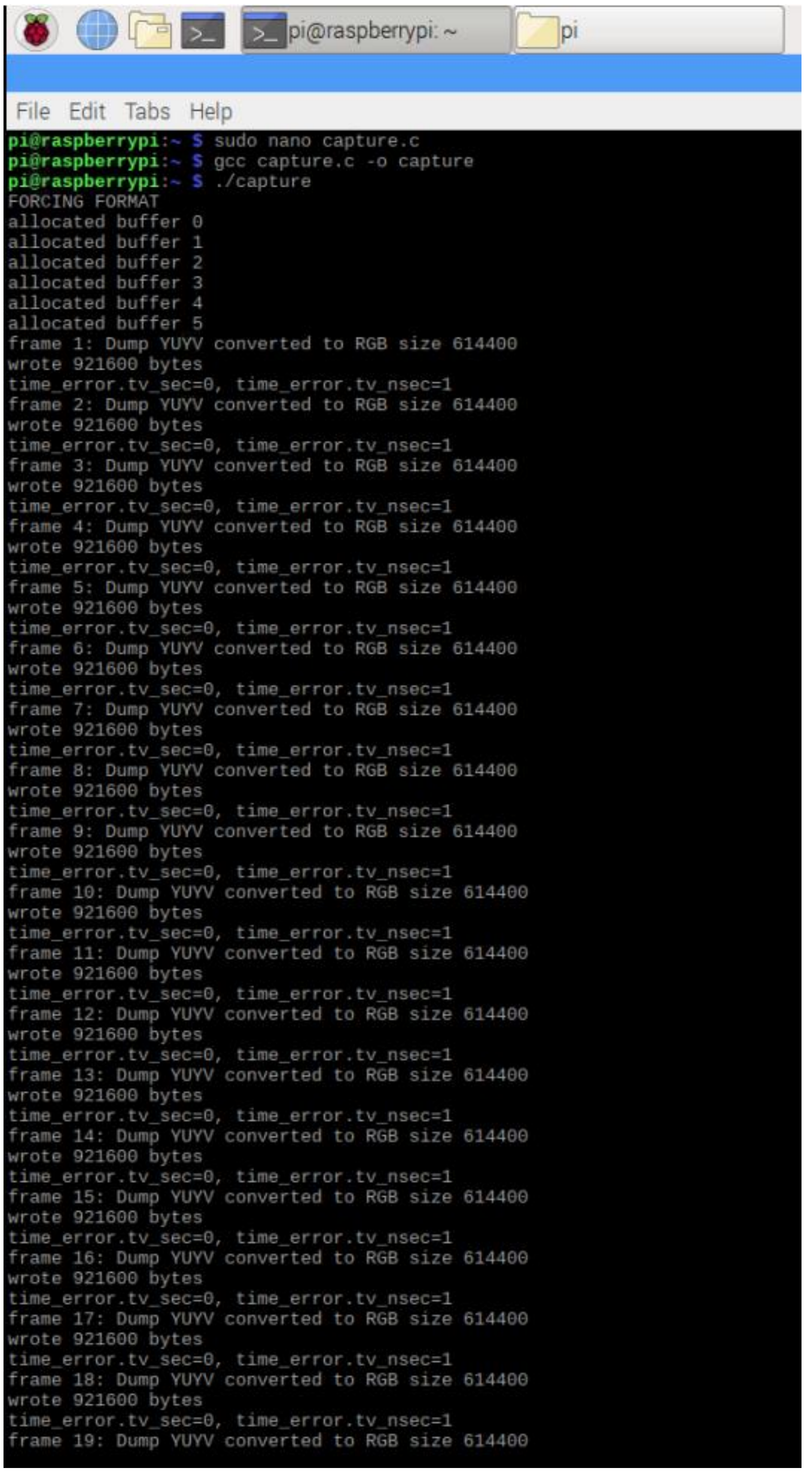
**Sol.**

**Demonstration:**

I have created one capture.c file using the resources provided by Dr. Siewert along with its capture (executable file). It is the same file that I created as a solution for this problem and I used the same file for explaining and demonstrating the solution to Problem #3.

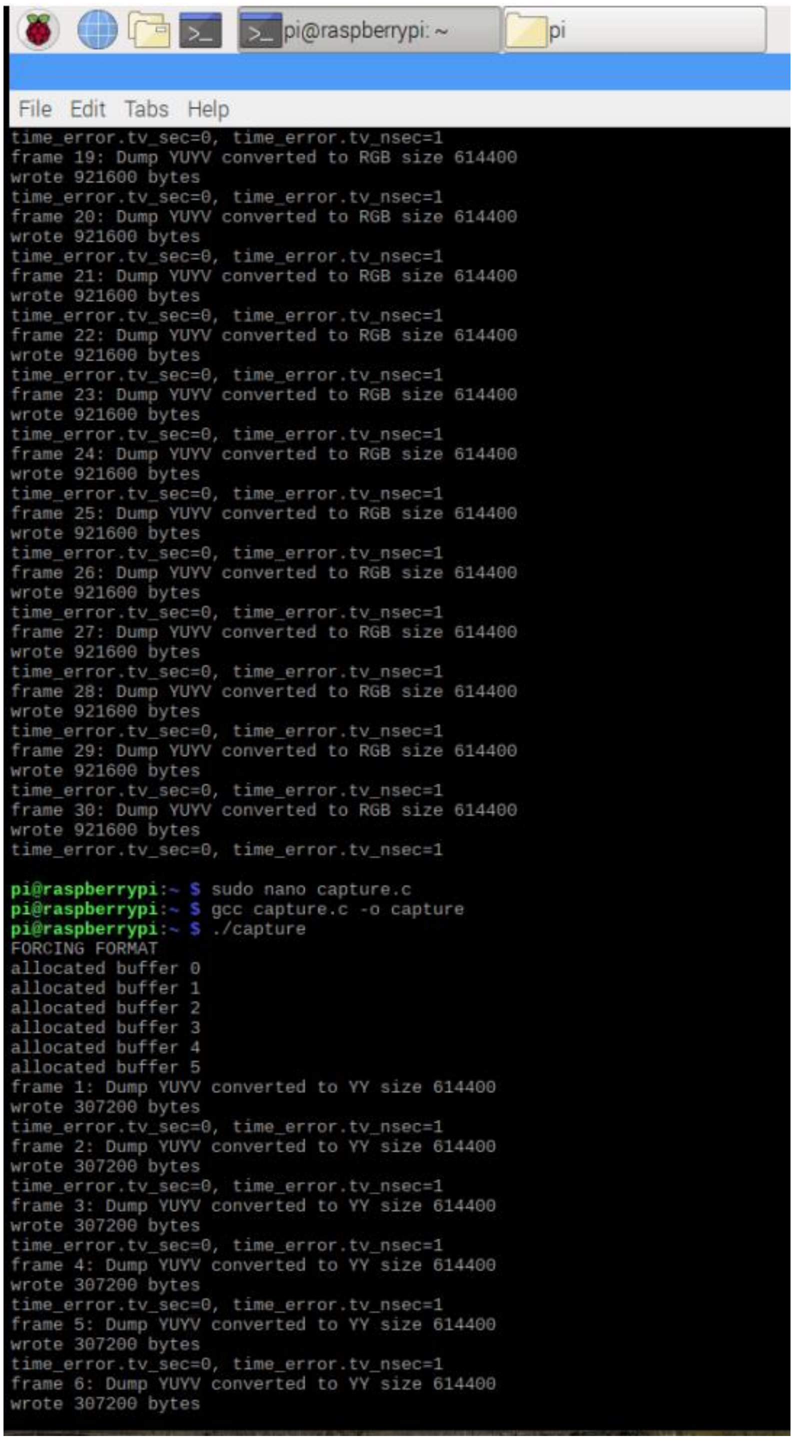
***Please, refer to the Prob #3 (subfolder) submitted for the previous problem in the zipped file which contains the source and executable files and the associated outputs. The screenshots and processed images will be the same as I have merged my demonstration for Problems #3 and #4 as they essentially built around the same concept. I have attached a detailed description of the code execution to provide better insights on how it works.***

I have attached the same execution screenshots (screenshots 10, 11 and 12) to showcase that I built and ran the code.



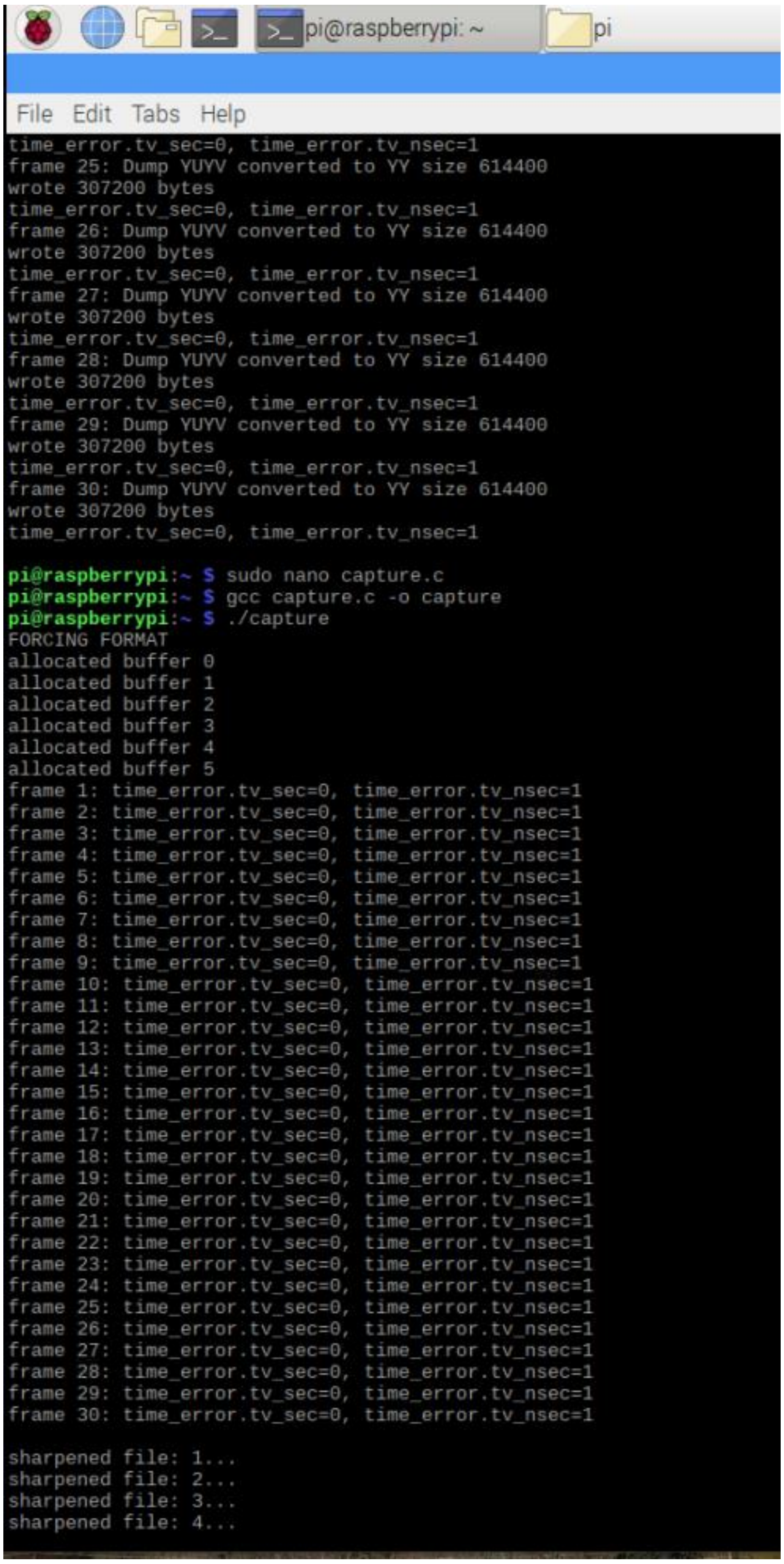
```
pi@raspberrypi:~ $ sudo nano capture.c
pi@raspberrypi:~ $ gcc capture.c -o capture
pi@raspberrypi:~ $ ./capture
FORCING FORMAT
allocated buffer 0
allocated buffer 1
allocated buffer 2
allocated buffer 3
allocated buffer 4
allocated buffer 5
frame 1: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 2: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 3: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 4: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 5: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 6: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 7: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 8: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 9: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 10: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 11: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 12: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 13: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 14: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 15: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 16: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 17: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 18: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 19: Dump YUYV converted to RGB size 614400
```

***Screenshot-10***



```
time_error.tv_sec=0, time_error.tv_nsec=1
frame 19: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 20: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 21: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 22: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 23: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 24: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 25: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 26: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 27: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 28: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 29: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 30: Dump YUYV converted to RGB size 614400
wrote 921600 bytes
time_error.tv_sec=0, time_error.tv_nsec=1

pi@raspberrypi:~ $ sudo nano capture.c
pi@raspberrypi:~ $ gcc capture.c -o capture
pi@raspberrypi:~ $ ./capture
FORCING FORMAT
allocated buffer 0
allocated buffer 1
allocated buffer 2
allocated buffer 3
allocated buffer 4
allocated buffer 5
frame 1: Dump YUYV converted to YY size 614400
wrote 307200 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 2: Dump YUYV converted to YY size 614400
wrote 307200 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 3: Dump YUYV converted to YY size 614400
wrote 307200 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 4: Dump YUYV converted to YY size 614400
wrote 307200 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 5: Dump YUYV converted to YY size 614400
wrote 307200 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 6: Dump YUYV converted to YY size 614400
wrote 307200 bytes
```

***Screenshot-11***



```
time_error.tv_sec=0, time_error.tv_nsec=1
frame 25: Dump YUYV converted to YY size 614400
wrote 307200 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 26: Dump YUYV converted to YY size 614400
wrote 307200 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 27: Dump YUYV converted to YY size 614400
wrote 307200 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 28: Dump YUYV converted to YY size 614400
wrote 307200 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 29: Dump YUYV converted to YY size 614400
wrote 307200 bytes
time_error.tv_sec=0, time_error.tv_nsec=1
frame 30: Dump YUYV converted to YY size 614400
wrote 307200 bytes
time_error.tv_sec=0, time_error.tv_nsec=1

pi@raspberrypi:~ $ sudo nano capture.c
pi@raspberrypi:~ $ gcc capture.c -o capture
pi@raspberrypi:~ $ ./capture
FORCING FORMAT
allocated buffer 0
allocated buffer 1
allocated buffer 2
allocated buffer 3
allocated buffer 4
allocated buffer 5
frame 1: time_error.tv_sec=0, time_error.tv_nsec=1
frame 2: time_error.tv_sec=0, time_error.tv_nsec=1
frame 3: time_error.tv_sec=0, time_error.tv_nsec=1
frame 4: time_error.tv_sec=0, time_error.tv_nsec=1
frame 5: time_error.tv_sec=0, time_error.tv_nsec=1
frame 6: time_error.tv_sec=0, time_error.tv_nsec=1
frame 7: time_error.tv_sec=0, time_error.tv_nsec=1
frame 8: time_error.tv_sec=0, time_error.tv_nsec=1
frame 9: time_error.tv_sec=0, time_error.tv_nsec=1
frame 10: time_error.tv_sec=0, time_error.tv_nsec=1
frame 11: time_error.tv_sec=0, time_error.tv_nsec=1
frame 12: time_error.tv_sec=0, time_error.tv_nsec=1
frame 13: time_error.tv_sec=0, time_error.tv_nsec=1
frame 14: time_error.tv_sec=0, time_error.tv_nsec=1
frame 15: time_error.tv_sec=0, time_error.tv_nsec=1
frame 16: time_error.tv_sec=0, time_error.tv_nsec=1
frame 17: time_error.tv_sec=0, time_error.tv_nsec=1
frame 18: time_error.tv_sec=0, time_error.tv_nsec=1
frame 19: time_error.tv_sec=0, time_error.tv_nsec=1
frame 20: time_error.tv_sec=0, time_error.tv_nsec=1
frame 21: time_error.tv_sec=0, time_error.tv_nsec=1
frame 22: time_error.tv_sec=0, time_error.tv_nsec=1
frame 23: time_error.tv_sec=0, time_error.tv_nsec=1
frame 24: time_error.tv_sec=0, time_error.tv_nsec=1
frame 25: time_error.tv_sec=0, time_error.tv_nsec=1
frame 26: time_error.tv_sec=0, time_error.tv_nsec=1
frame 27: time_error.tv_sec=0, time_error.tv_nsec=1
frame 28: time_error.tv_sec=0, time_error.tv_nsec=1
frame 29: time_error.tv_sec=0, time_error.tv_nsec=1
frame 30: time_error.tv_sec=0, time_error.tv_nsec=1

sharpened file: 1...
sharpened file: 2...
sharpened file: 3...
sharpened file: 4...
```

***Screenshot-12***

**Description:**

The **Code Execution Explanation** and **Streaming Verification** provided in the previous problem explains how the code execution handles the camera device for acquiring video frames at a given frame rate (e.g. 30 fps) and for a given transformation (e.g. RGB transformation) with the images dumped into the system and the outputs provided onto the logger. The below description dives deeper into the code structure of the **capture.c** file and briefly explains the execution flow of the code based on the high-level functions supported by the V4L2 API calls:

1. The open_device() function opens the video0 device located in the /dev directory. The video0 device is actually the USB camera (C270) which the system detects and interfaces the block device named video0. The open_device() function sets up video0 for communication and throws error if it fails to respond. This task is performed using the V4L2 API.

2. The init_device() function checks the camera for its capabilities to capture video and its ability to read memory types as per the need of the system. Successful execution of this function, sets up the system for reliable and efficient communication with the camera.

3. The start_capturing() function based on the IO control method specified in the system, communicates with the Memory Mapped IO. The function V4L2 API to request the system to allocate memory buffers to the camera for video capturing. Once that is successfully done, the camera is ready in the following ways:

    a. The camera is detected using its video0 block device interface provided by the V4L2 API in the system.
    b. The camera is checked against its capabilities to communicate reliably for video capturing and memory access.
    c. The camera is allocated memory to start capturing video frames as per the need of the user.

4. The mainloop() function reads frames (using the read_frame() function) and processes it according to the specified #define preprocessor directives. The continuous transformation is done using the process_image() function.

5. The processed image is dumped into a file in the specified .pgm or .ppm file format in the specified YUYV, YY, RGB, etc. image formats. Once this is successfully done, the code execution begins to act in the reverse direction.

6. The stop_capturing() function instructs the system to stop requesting and acquiring frames from the camera.

7. The uninit_device() function frees the allocated buffers for communication with the device0 camera interface.

8. Finally, the close_device() breaks off the connection with the video0 block device and the code is exited thereafter.

The above execution flow represents the following points in a nutshell:

1. Camera block device (video0) detection
2. Camera check for its capabilities to communicate with the system
3. Memory allocation to the Camera device to store information onto the system
4. Capturing and processing and image frame acquired by the system from the camera with the help of V4L2 API

**<u>Applications of Continuous Transformation:</u>**

There are innumerous applications after continuous transformation (Image processing) and image processing. Some of the key points as emphasized in the referenced document are as under:

- Image processing (IP) allows for image restoration and enhancement especially when the acquired images have a lot of noise.
- IP facilitates recognition systems to provide highly automated tasks (e.g. self-driving cars).
- IP has applications in the aerospace industry where constant efforts are made to acquire better images to study the space. Use of image processing is significantly important to process massive information in the aerospace industry, especially in remote sensing applications.
- IP has applications in the biomedical industry where images are filtered and studied to find meaningful information at the microscopic scale.
- IP is useful in recognition and machine intelligence (e.g. finger-print sensors, self-driving cars, etc.).
- IP has a lot of applications in the industry where product verification for faults is a very critical task which can be huge problem if unaddressed.

**Courtesy:** Links – [1]

5) [30 points] 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.

**Sol.**

***Please Note: For the solution, I have attached two source files: one used for performing the testing of the timing analysis which is not a RT-threaded execution of the system (capture2.c). The other file (capture3.c) provides the RT-threaded execution based on the reasonably selected SRT deadlines based on the analysis carried out in the solution below.***

**Design Concepts:**

For this problem, some design considerations were taken into account which are as under:

- Proper timeout selection for frame acquisitions for different resolutions of the same aspect ratio.
- Iterations for jitter & timing testing for execution of each transformation for each of the five resolutions.
- Design of accurate time-stamping routines so that accurate deadline decisions can be made.
- Elimination of "printf" statements and solely the use of "syslog" logger system for reducing system latencies that add to the error in the SRT response of the system.
- Shifting of the transformation routines to POSIX SRT threads for scheduling and testing for RM.

**Algorithm Analysis:**

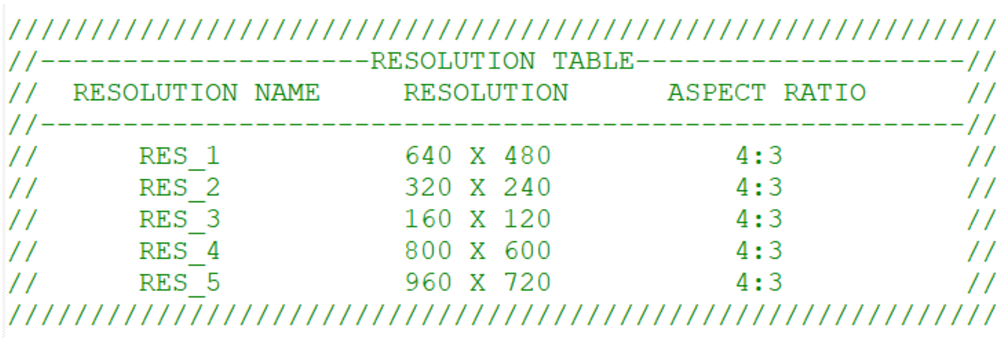
As part of the algorithm analysis, the following things can be observed:

- Keeping the non-RT processing elements outside of the SRT execution – like device (C270) setup and initialization. Once that is done, the system enters the SRT execution mode where the service is non-periodic in nature and only ends at the termination of the program.
- Using semaphores to protect unintended preemption while the threads are executing critical sections (i.e. global data that is shared across the threads).
- Using proper frequency allocation for each thread so that a high amount of reliability and predictable response is acquired. The analysis performed in the later section of this solution provides more information on the same.

- The algorithm uses a lot of macros to increase the simplicity in the execution of the code. However, run-time features for adding arguments to simply test the system for different transformations has not been added.
- The algorithm does not consider the dumping of files as part of the RT execution and jitter analysis. However, the processing of images performed (i.e. the `mainloop()` function) is considered as a RT execution thread.
- For the image sharpening algorithm, it was originally designed to read the RGB .ppm frames and then convert it to sharpened files and store it to a file in the .ppm format. Although, the writing (dumping) of sharpened images to a .ppm file is contained in a macro safe-guarding that part from RT-execution, the thread requires reading the RGB image files which consistently adds to a significant overhead. As a result of this, the image sharpening jitter analysis shows that it requires a lot of execution time (in the 1500-1600 milliseconds range). However, the same is not true for other transformations as they are not reading any image file but are directly reading frames inside the `mainloop()` and therefore do not add overhead. They consume about 30-40 milliseconds which is way less than what it takes for the sharpening. I could reduce that overhead but it required some modification to my existing algorithm which could potentially disrupt other sections and require more time to fix everything.

## <u>Jitter Analysis:</u>

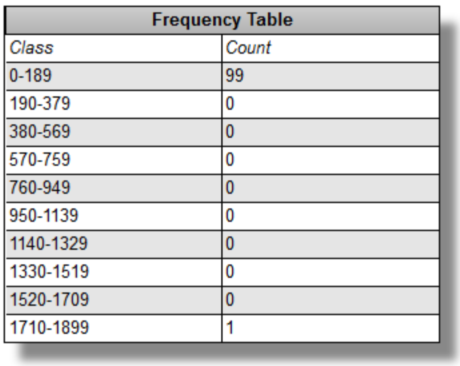
With the use of 5 resolutions with a common aspect ratio as provided below, and with a feature to timestamp every frame processing, I performed a jitter analysis of the system and how the system should be scheduled in order to have a predictable SRT response.



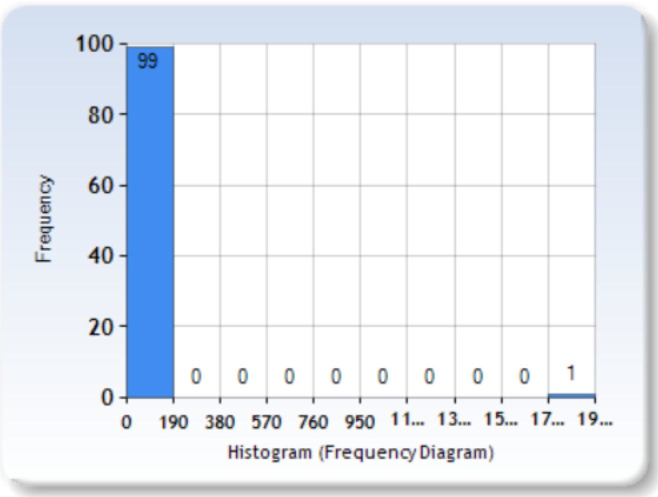
```
////////////////////////////////////////////////////////////
//--------------------RESOLUTION TABLE--------------------//
//  RESOLUTION NAME     RESOLUTION      ASPECT RATIO      //
//--------------------------------------------------------//
//      RES_1           640 X 480           4:3           //
//      RES_2           320 X 240           4:3           //
//      RES_3           160 X 120           4:3           //
//      RES_4           800 X 600           4:3           //
//      RES_5           960 X 720           4:3           //
////////////////////////////////////////////////////////////
```

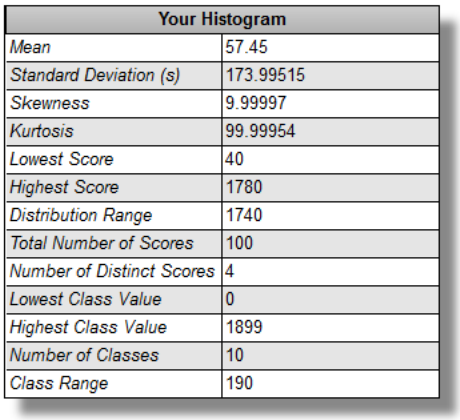
The files containing the timing in **milli-seconds** for each image transformation (i.e. RGB (color), Grayscale and Sharpening) has been added to the **Jitter Test** subfolder (**in the Prob #5 subfolder**) provided in the zipped file. The below information focuses on the jitter analysis for each transformation (per frame for a set of 100 frames) for 5 different 4:3 resolutions as mentioned above. This leaves 15 different jitter analysis elements and the screenshots for each jitter analysis (5 for each transformation) has been provided in here and in the zipped file.

*Please Note: The jitter analysis histogram study was done with the help of the resources [1] and [2] mentioned in the courtesy section below. I duly credit the resources provided by the website to create histograms for a set of large data.*

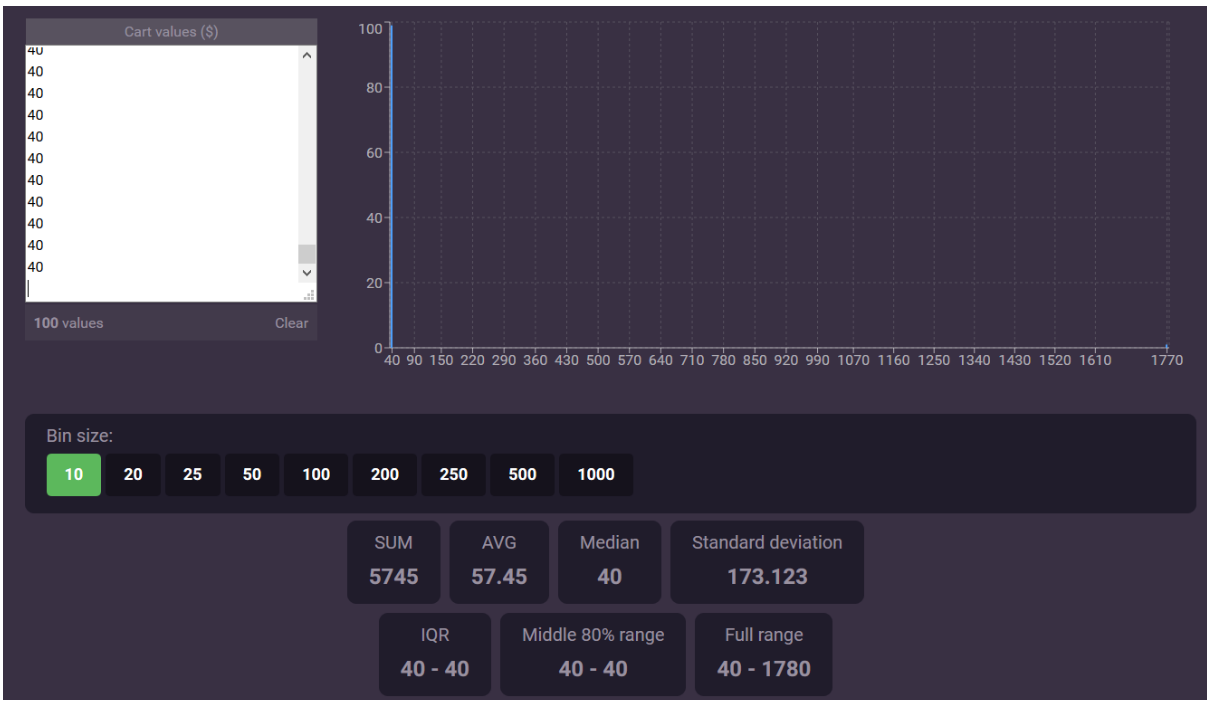


| Frequency Table | |
|---|---|
| Class | Count |
| 0-189 | 99 |
| 190-379 | 0 |
| 380-569 | 0 |
| 570-759 | 0 |
| 760-949 | 0 |
| 950-1139 | 0 |
| 1140-1329 | 0 |
| 1330-1519 | 0 |
| 1520-1709 | 0 |
| 1710-1899 | 1 |





| Your Histogram | |
|---|---|
| Mean | 57.45 |
| Standard Deviation (s) | 173.99515 |
| Skewness | 9.99997 |
| Kurtosis | 99.99954 |
| Lowest Score | 40 |
| Highest Score | 1780 |
| Distribution Range | 1740 |
| Total Number of Scores | 100 |
| Number of Distinct Scores | 4 |
| Lowest Class Value | 0 |
| Highest Class Value | 1899 |
| Number of Classes | 10 |
| Class Range | 190 |



Cart values ($)

```
40
40
40
40
40
40
40
40
40
40
40
```

100 values

Bin size: **10** 20 25 50 100 200 250 500 1000

| SUM | AVG | Median | Standard deviation |
|---|---|---|---|
| 5745 | 57.45 | 40 | 173.123 |

| IQR | Middle 80% range | Full range |
|---|---|---|
| 40 - 40 | 40 - 40 | 40 - 1780 |

***RGB (Color) Transformation Jitter Analysis (using histogram) for RES_1 (640 x 480)***

The above jitter analysis shows that for the execution of the code capture2.c (with capture2 executable-file), the RGB transformation for RES_1 resolution is tested for 100 frames. For a test consisting of 100 frames showcased in the histogram with the **x-axis showing the time taken for ONE frame transformation in milli-seconds** and **y-axis showing the number of frame transformations corresponding to a given ONE frame transformation time**. From the above analysis, it can be observed that 99% of the frames consume an execution time of 40 milli-seconds (which is middle range 80%) based on the above histograms. There is only 1% frame (i.e. just 1 frame) that consumes 1780 milliseconds. On an average, the execution time is 57.45 milli-seconds which includes the 1780 milliseconds execution time consumed by 1% of the frames. However, for a soft RT system that does not have catastrophic impacts upon missing a deadline, it can be feasible to provide a deadline period of **80 milliseconds** (with a potential margin of about 40 milliseconds based on the resolution of the histogram above since 99% of the frames will execute in about 40 milliseconds) for a service that performs the above execution.

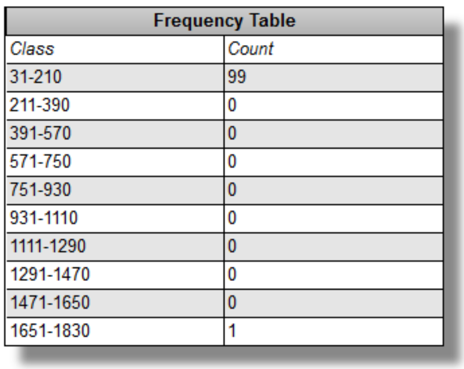
Analysis Conclusion (based on the information provided in the above screenshot):

Potential Average Execution Time: **40 milli-seconds (approx.)**

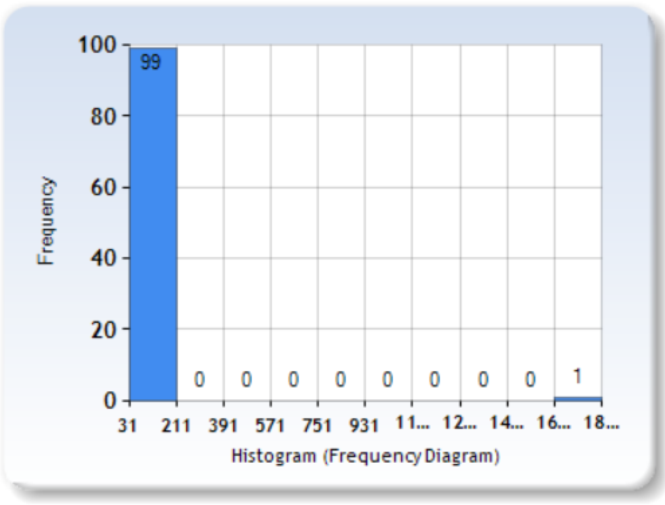
Worst Case Execution Time: **1780 milli-seconds (approx.) – Not ideal since it occurs only for 1% of the time**

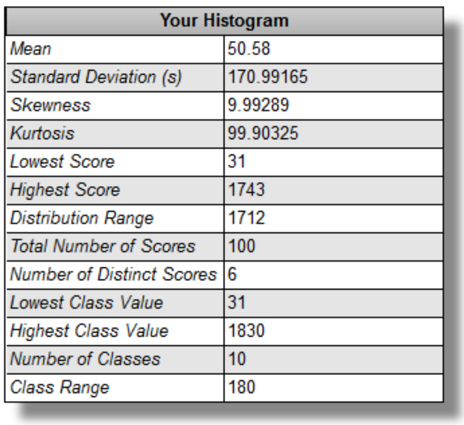
SRT Execution Deadline: **80 milli-seconds (approx.)**

SRT Margin: **40 milli-seconds (approx.)**

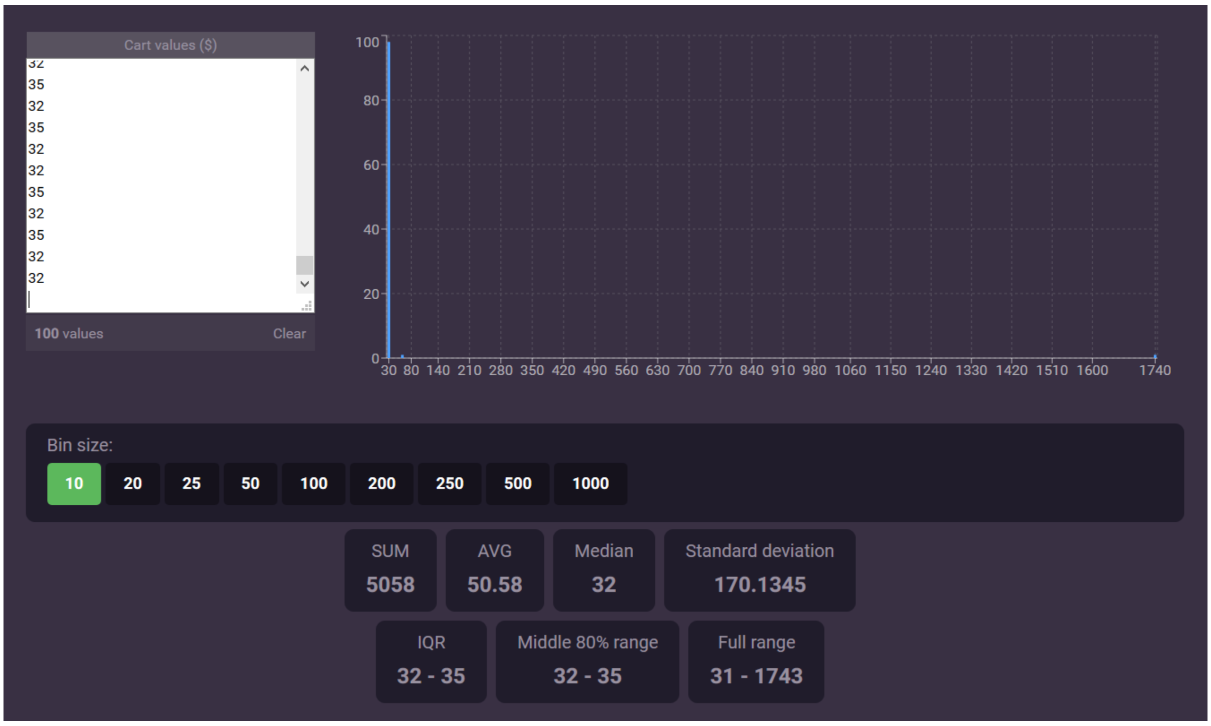


| Frequency Table | |
|---|---|
| *Class* | *Count* |
| 31-210 | 99 |
| 211-390 | 0 |
| 391-570 | 0 |
| 571-750 | 0 |
| 751-930 | 0 |
| 931-1110 | 0 |
| 1111-1290 | 0 |
| 1291-1470 | 0 |
| 1471-1650 | 0 |
| 1651-1830 | 1 |





| Your Histogram | |
|---|---|
| *Mean* | 50.58 |
| *Standard Deviation (s)* | 170.99165 |
| *Skewness* | 9.99289 |
| *Kurtosis* | 99.90325 |
| *Lowest Score* | 31 |
| *Highest Score* | 1743 |
| *Distribution Range* | 1712 |
| *Total Number of Scores* | 100 |
| *Number of Distinct Scores* | 6 |
| *Lowest Class Value* | 31 |
| *Highest Class Value* | 1830 |
| *Number of Classes* | 10 |
| *Class Range* | 180 |



SUM 5058 | AVG 50.58 | Median 32 | Standard deviation 170.1345

IQR 32 - 35 | Middle 80% range 32 - 35 | Full range 31 - 1743

*Grayscale Transformation Jitter Analysis (using histogram) for RES_1 (640 x 480)*

The above jitter analysis shows that for the execution of the code capture2.c (with capture2 executable-file), the RGB transformation for RES_1 resolution is tested for 100 frames. For a test consisting of 100 frames showcased in the histogram with the **x-axis showing the time taken for ONE frame transformation in milli-seconds** and **y-axis showing the number of frame transformations corresponding to a given ONE frame transformation time**. From the above analysis, it can be observed that around 80% of the frames consume an execution time of 32-35 milli-seconds (middle range) based on the above histograms. There is only 1% frame (i.e. just 1 frame) that consumes 1743 milliseconds. On an average, the execution time is 50.58 milli-seconds which includes the 1743 milliseconds execution time consumed by 1% of the frames. However, for a soft RT system that does not have catastrophic impacts upon missing a deadline, it can be feasible to provide a deadline period of **80 milliseconds** (with a potential margin of about 45 milliseconds based on the resolution of the histogram above since 99% of the frames will execute in about 40 milliseconds) for a service that performs the above execution.

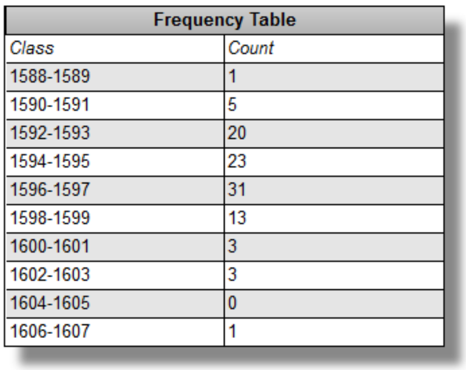
Analysis Conclusion (based on the information provided in the above screenshot):

Potential Average Execution Time: **32-35 milli-seconds (approx.)**

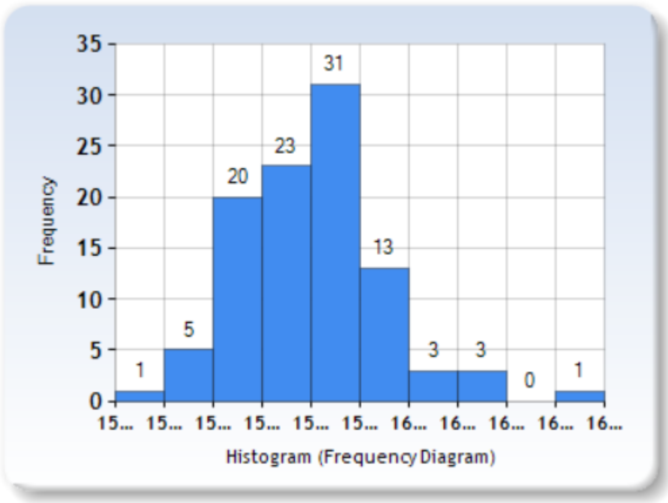
Worst Case Execution Time: **1743 milli-seconds (approx.) – Not ideal since it occurs only for 1% of the time**

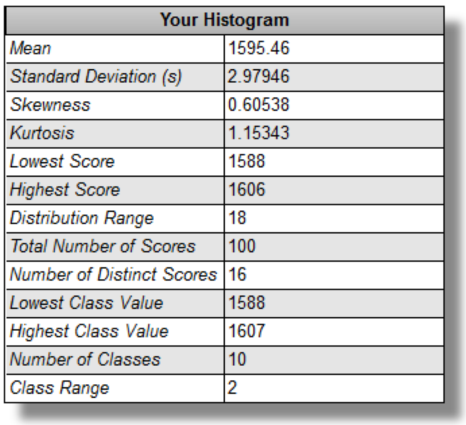
SRT Execution Deadline: **80 milli-seconds (approx.)**

SRT Margin: **45 milli-seconds (approx.)**

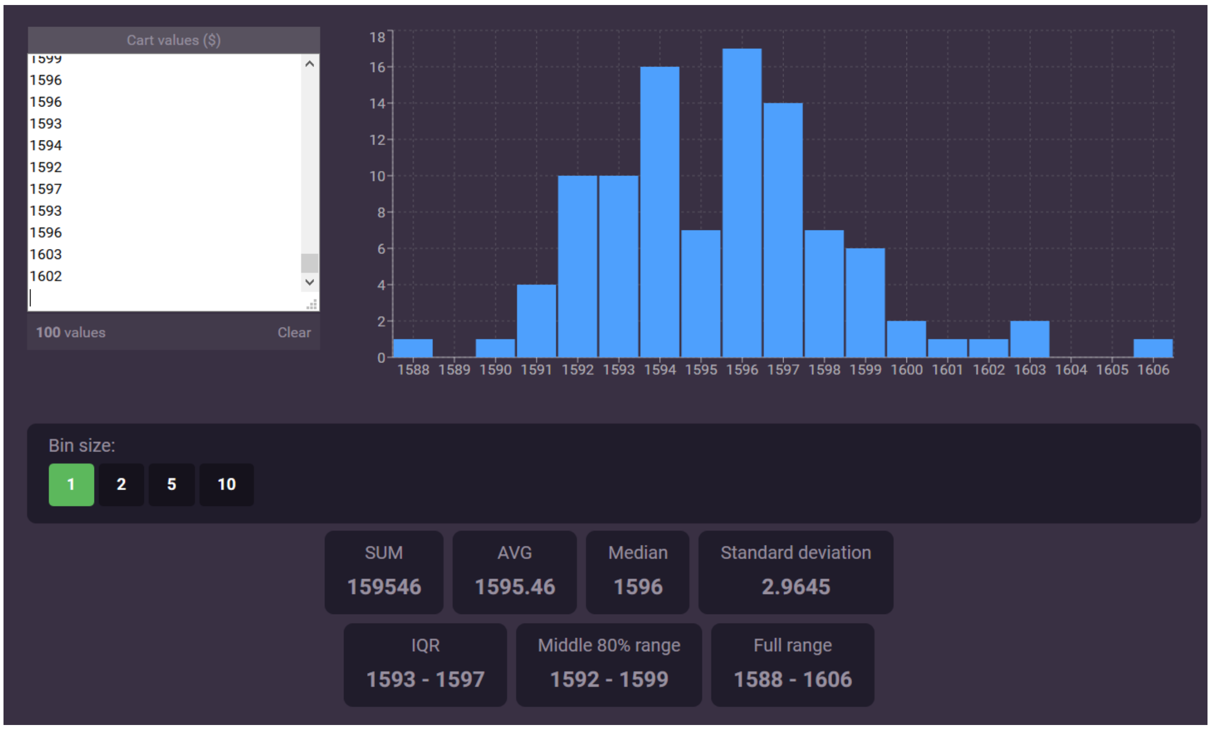


| Frequency Table | |
|---|---|
| Class | Count |
| 1588-1589 | 1 |
| 1590-1591 | 5 |
| 1592-1593 | 20 |
| 1594-1595 | 23 |
| 1596-1597 | 31 |
| 1598-1599 | 13 |
| 1600-1601 | 3 |
| 1602-1603 | 3 |
| 1604-1605 | 0 |
| 1606-1607 | 1 |





| Your Histogram | |
|---|---|
| Mean | 1595.46 |
| Standard Deviation (s) | 2.97946 |
| Skewness | 0.60538 |
| Kurtosis | 1.15343 |
| Lowest Score | 1588 |
| Highest Score | 1606 |
| Distribution Range | 18 |
| Total Number of Scores | 100 |
| Number of Distinct Scores | 16 |
| Lowest Class Value | 1588 |
| Highest Class Value | 1607 |
| Number of Classes | 10 |
| Class Range | 2 |



Cart values ($)

```
1599
1596
1596
1593
1594
1592
1597
1593
1596
1603
1602
```

100 values Clear

Bin size: [✓ 1] 2 5 10

| SUM | AVG | Median | Standard deviation |
|---|---|---|---|
| 159546 | 1595.46 | 1596 | 2.9645 |

| IQR | Middle 80% range | Full range |
|---|---|---|
| 1593 - 1597 | 1592 - 1599 | 1588 - 1606 |

***Image Sharpening Transformation Jitter Analysis (using histogram) for RES_1 (640 x 480)***

The above jitter analysis shows that for the execution of the code capture2.c (with capture2 executable-file), the Grayscale transformation for RES_1 resolution is tested for 100 frames. For a test consisting of 100 frames showcased in the histogram with the **x-axis showing the time taken for ONE frame transformation in milli-seconds** and **y-axis showing the number of frame transformations corresponding to a given ONE frame transformation time**. From the above analysis, it can be observed that the histogram has a bell curve based on its resolution. A significant amount (80 % of frames) have their execution time between 1592 and 1599 milli seconds. While it lies between 1588 and 1607 milliseconds, the average execution time is around 1595.5 milliseconds. To generalize, the execution is spread across a range of 19 milliseconds (i.e. 1607 – 1507 milliseconds) and expanding that with a 50 millisecond margin would be suitable. That means, the execution may take around 1650 milliseconds in the worst case, assuming that there is a 50 milliseconds margin still saves the system from missing the deadline. Based on the above information, it can be assumed that for a soft RT system that does not have catastrophic impacts upon missing a deadline, it can be feasible to provide a deadline period of **1650 milliseconds** (with a potential margin of 50 milliseconds based on the resolution of the histogram above since 1607 milliseconds was the worst execution time) for a service that performs the above execution.

Analysis Conclusion (based on the information provided in the above screenshot):

Potential Average Execution Time: **1595.5 milli-seconds (approx.)**

Worst Case Execution Time: **1607 milli-seconds (approx.)**

SRT Execution Deadline: **1650 milli-seconds (approx.)**

SRT Margin: **45-50 milli-seconds (approx.)**

***Assumptions:***

*In the RGB and Grayscale prototype analysis, it was assumed that the system will not ever cross a worst case execution time that is beyond the decided margin. On an another note, since it is a soft real-time deadline, if the worst case execution time occurs only for 1% of the time and is way away from the mean and the median then it is ignored. Being a SRT system, the impacts of missing a deadline are not catastrophic and therefore, if missing 1% of the deadlines increases system efficiency and schedulability, then it is worth the effort to ignore the 1% failure chances. If the 1% failure chances are not ignored, the margin will be more than 100% of the mean and median execution times. For a HRT, the system should not miss that around 1% of the frames but for SRT, it is more important to service as much as possible with the system resources that are available.*

*On the other hand, for the analysis as seen in the Image sharpening transformation above, the bell-curved histogram has almost a uniformly varying spread of execution times. In such case, since the worst case execution is not too far apart from the middle 80% frequency range, it is acceptable and care is taken that the worst case execution time is considered for the allocation of margin.*

***For the remaining 12 image transformation analysis as above, the same approach to judge the jitter response and scheduling is applied and implemented. All the screenshots for the histograms are provided in the zipped file. It is assumed that having a margin of 40-50 milliseconds above the execution time in the middle range (i.e. for 80% of the frames) would be very safe for a SRT system. Ignoring a very minor amount of frequency (i.e. the worst execution time of 1% of frames) which is way too far from the middle range would not harm much if missed for a SRT but would actually allow for better schedulability as services may be serviced faster.***

**Courtesy:** Links – [1] [2]