Exercise #4 – Real-Time Continuous Media

1) Obtain a Logitech C270 camera (or equivalent) and verify that is detected by the Raspberry Pi USB driver.

a) Show that it has been detected by using lsusb, lsmod and dmesg kernel driver configuration tool to make sure your Logitech C270 USB camera is plugged in and recognized. Do lsusb | grep -i logitech and prove to me (and more importantly yourself) with that output (screenshot) that your camera is recognized.

```
anuhya@raspberrypi:~ $ lsusb
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 003: ID 046d:0825 Logitech, Inc. Webcam C270
Bus 001 Device 002: ID 2109:3431 VIA Labs, Inc. Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
anuhya@raspberrypi:~ $
```

b) Now, do lsmod | grep video and verify that the UVC driver is loaded as well (http://www.ideasonboard.org/uvc/).

```
anuhya@raspberrypi:~ $ lsmod | grep video
uvcvideo              110592  0
videobuf2_vmalloc      16384  2 uvcvideo,bcm2835_v4l2
videobuf2_dma_contig   20480  3 bcm2835_codec,rpivid_hevc,bcm2835_isp
videobuf2_memops       16384  2 videobuf2_vmalloc,videobuf2_dma_contig
videobuf2_v4l2         32768  6 bcm2835_codec,uvcvideo,bcm2835_v4l2,rpivid_hevc,v4l2_mem2mem,bcm2835_isp
videobuf2_common       69632  10 bcm2835_codec,videobuf2_vmalloc,videobuf2_dma_contig,videobuf2_v4l2,uvcvideo,bcm2835_v4l2,rpivid_hevc,v4l2_mem2mem,videobuf2
memops,bcm2835_isp
videodev              274432  8 bcm2835_codec,videobuf2_v4l2,uvcvideo,bcm2835_v4l2,videobuf2_common,rpivid_hevc,v4l2_mem2mem,bcm2835_isp
mc                     61440  9 videodev,bcm2835_codec,snd_usb_audio,videobuf2_v4l2,uvcvideo,videobuf2_common,rpivid_hevc,v4l2_mem2mem,bcm2835_isp
```

c) 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.

```
anuhya@raspberrypi:~ $ dmesg | grep video
[    5.465407] videodev: Linux video capture interface: v2.00
[    5.786166] bcm2835-isp bcm2835-isp: Device node output[0] registered as /dev/video13
[    5.798334] rpivid feb10000.codec: Device registered as /dev/video19
[    5.803396] bcm2835-isp bcm2835-isp: Device node capture[0] registered as /dev/video14
[    5.806273] bcm2835-isp bcm2835-isp: Device node capture[1] registered as /dev/video15
[    5.811225] bcm2835-codec bcm2835-codec: Device registered as /dev/video10
[    5.821685] bcm2835-codec bcm2835-codec: Device registered as /dev/video11
[    5.831499] bcm2835-isp bcm2835-isp: Device node stats[2] registered as /dev/video16
[    5.837377] bcm2835-codec bcm2835-codec: Device registered as /dev/video12
[    5.873454] bcm2835-codec bcm2835-codec: Device registered as /dev/video18
[    5.896311] bcm2835-codec bcm2835-codec: Device registered as /dev/video31
[    5.922553] bcm2835-isp bcm2835-isp: Device node output[0] registered as /dev/video20
[    5.923728] bcm2835-isp bcm2835-isp: Device node capture[0] registered as /dev/video21
[    5.965947] bcm2835-isp bcm2835-isp: Device node capture[1] registered as /dev/video22
[    6.047661] bcm2835-isp bcm2835-isp: Device node stats[2] registered as /dev/video23
[    7.315236] usbcore: registered new interface driver uvcvideo
```

d) Capture all output and annotate what you see with descriptions to the best of your understanding.

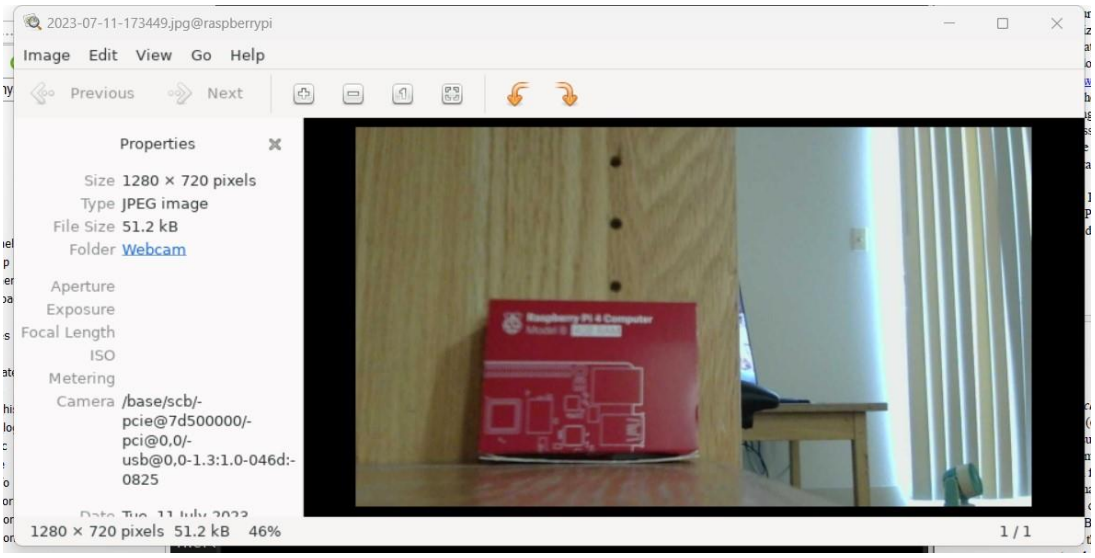
The 'lsusb' command is used to list the USB devices connected to the Raspberry Pi. It prints out the details like- vendor ID, product ID, and company associated with the device.

The 'lsmod' command is used to list the loaded kernel modules (drivers). The 'grep' command searches for the term passed as the argument in the information listed out in lsmod. The UVC driver is USB video class device driver which needs to be used for USB cameras like Logitech C270 in this case.

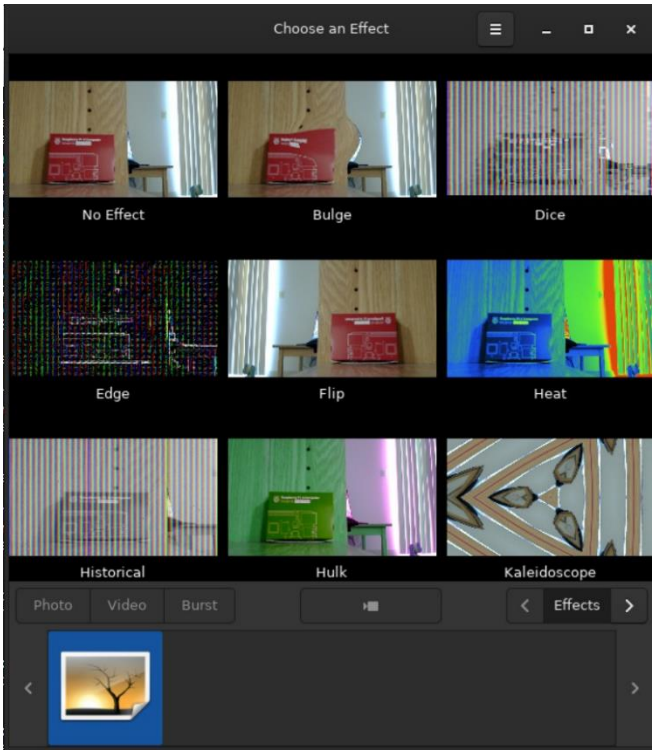
The 'dmesg' command lists out the kernel ring buffer messages. With the 'grep' command, all the video-related messages are printed out- such as device detection, and driver information.

2) If you do not have camorama, do "sudo apt-get install camorama" on your Raspberry Pi board [you may need to first do sudo apt-get update]. If for some reason camorama does not work for you, an alternative application is cheese (install with "sudo apt- get install cheese"). 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/). Provide a summary log of your installation.

a) Run camorama with no arguments and it should provide an interactive camera control session for your Logitech C2xx camera – provide a screen dump of the GUI.



b) 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 your network, USB, or sudo and your account, research this and please do so.



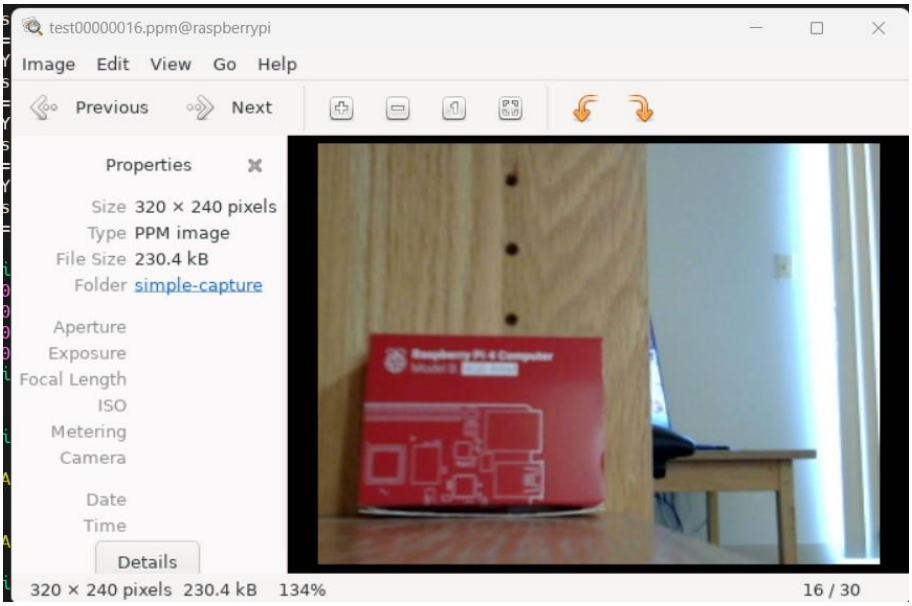
3) Using your verified Logitech C270 camera on your Raspberry Pi, now verify that it can stream continuously.

a) Using starter code, capture 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 V4L2 API. Provide a screen shot to prove that you got continuous capture to work with V4L2. To display images you have captured, you will find "eom" to be quite useful, which can be installed with "sudo apt-get install eom". Optional: Alternatively, if you were to use OpenCV, which abstracts the camera interface and provides useful library functions with code found here - simpler-capture-2/, you can see the code is simpler, but please do not use OpenCV. Note that simpler-capture-2 will require installation of OpenCV on an R-Pi 3b (if you have extra time to explore, here's some instructions to install it - please follow https://www.learnopencv.com/installopencv-

3-4-4-on-raspberry-pi/ - to use simpler-capture-2 as is with no modifications). You can also install the latest version of OpenCV, 4.x, which has only a C++ API, then you will need to use the most up to date code (simpler-capture-4), but bear in mind that you must use V4L2 for 5623, this code is for the follow-on course, 5763.

The simple capture code output is shown below. With use of 'eom', the captured image can be displayed.

```
anuhya@raspberrypi:~/Desktop/RTES-ECEE-5623-main/simple-capture $ ls
capture     test00000001.ppm  test00000005.ppm  test00000009.ppm  test00000013.ppm  test00000017.ppm  test00000021.ppm  test00000025.ppm  test00000029.ppm
capture.c   test00000002.ppm  test00000006.ppm  test00000010.ppm  test00000014.ppm  test00000018.ppm  test00000022.ppm  test00000026.ppm  test00000030.ppm
capture.o   test00000003.ppm  test00000007.ppm  test00000011.ppm  test00000015.ppm  test00000019.ppm  test00000023.ppm  test00000027.ppm
Makefile    test00000004.ppm  test00000008.ppm  test00000012.ppm  test00000016.ppm  test00000020.ppm  test00000024.ppm  test00000028.ppm
```



b) Download starter code (simple-capture-1800) that captures 1800 frames from your camera using V4L2 and run it as is first to test the code (use "make clean", "make", and run – note that make clean will remove test frames). Modify the code so that it uses syslog rather than printf to the console for tracing. Then, compare frame rate for PGM (graymap) and PPM (color) image write-back to the frame sub-directory on your flash filesystem and provide analysis of average and worst-case (lowest) frame rate seen for each.

```
anuhya@raspberrypi:~/Desktop/question_3 $ sudo ./capture
FORCING FORMAT
allocated buffer 0
allocated buffer 1
allocated buffer 2
allocated buffer 3
allocated buffer 4
allocated buffer 5

 Total capture time=12.082541, for 182 frames, 15.063057 FPS
```

Figure 1: Frame rate for PPM images

```
anuhya@raspberrypi:~/Desktop/question_3 $ sudo ./capture
FORCING FORMAT
allocated buffer 0
allocated buffer 1
allocated buffer 2
allocated buffer 3
allocated buffer 4
allocated buffer 5

 Total capture time=12.088069, for 182 frames, 15.056168 FPS
```

Figure 1: Frame rate for PGM images

Average FPS:	19.64241063
Least FPS:	13.638807

Table 1: Average and least FPS for PPM images

Average FPS	24.9247
Min. FPS	23.75196

Table 2: Average and least FPS for PGM images

The code is run for 180 frames with 8 starter frames. When the camera pixel format is checked to be YUYV format, depending on whether the COLOR_CONVERT_RGB is set, the frame is converted to a ppm (color) format or pgm (graymap) format. The write-back for the PGM format takes longer as the 4 bytes of YUYV need to be converted to 6 bytes of RGB format using the function yuv2rgb(). The conversion used is called YUV422. As shown in the table 1, for PPM images, the average frame rate seen is 19.64 and the worst-case frame rate observed is 13.638. In the table 2, the average and worst-case frame rates for PGM images are shown to be 24.92 and 23.75. The frames rate for each frame for both ppm and pgm images are stored in the excel sheet- 'fps for ppm and pgm images'.

4) Choose ONE continuous transformation for each acquired frame such as color conversion (Lecture-Cont-Media.pdf, slide 5), conversion to grayscale (slide 6), negative image (new_pixel = saturation – pixel), brightness/contrast adjustment (e.g., /c- brighten/brighten.c), or sharpening (/sharpen-psf/sharpen.c) and consider how to apply this to a real-time stream of image frames acquired from your camera rather than just one image file (PPM in the examples). Optional: If you want to use OpenCV (not required, but can be fun and interesting if you have extra time) look at examples from computer_vision_cv3_tested/ then you can choose transformations such as the diff-interactive/, simple-canny-interactive/, simple-houghinteractive/ , etc. Note that these examples were written for OpenCV 3.x, not OpenCV 4.x. Newer versions of this code can be found in computer_vision_cv4_tested. This is not required, but noted for students interested in real-time computer vision, which is covered in depth in ECEN 5763, Embedded Machine Vision and Intelligent Automation.

a) Show a screen shot to prove you built and ran the code and a before and after transform example image.

The continuous transformation used is sharpening the frame. The image before transformation is shown in Figure 1. The image after transformation is shown in Figure 2.

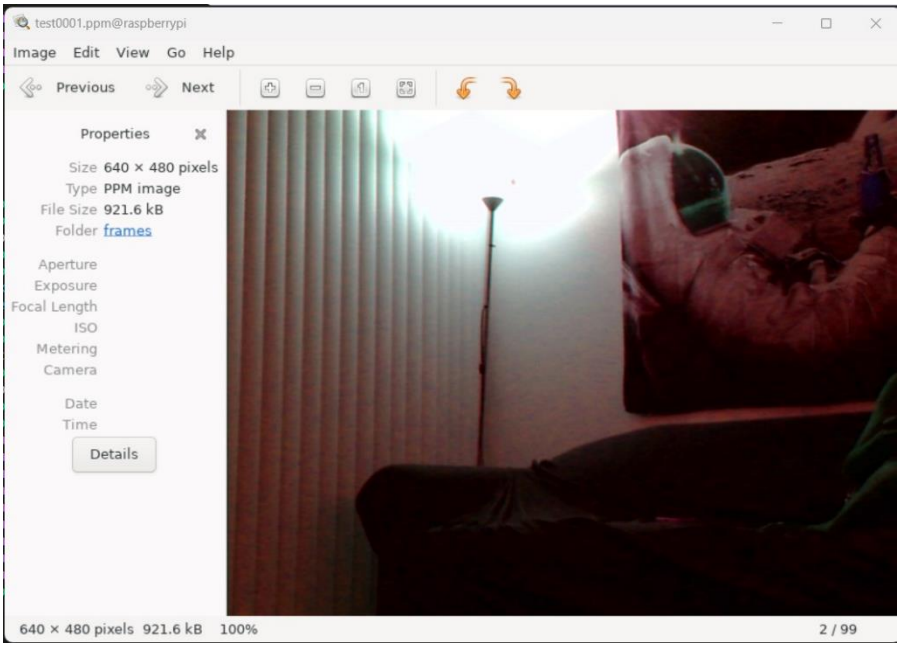


Figure 1: Before transformation

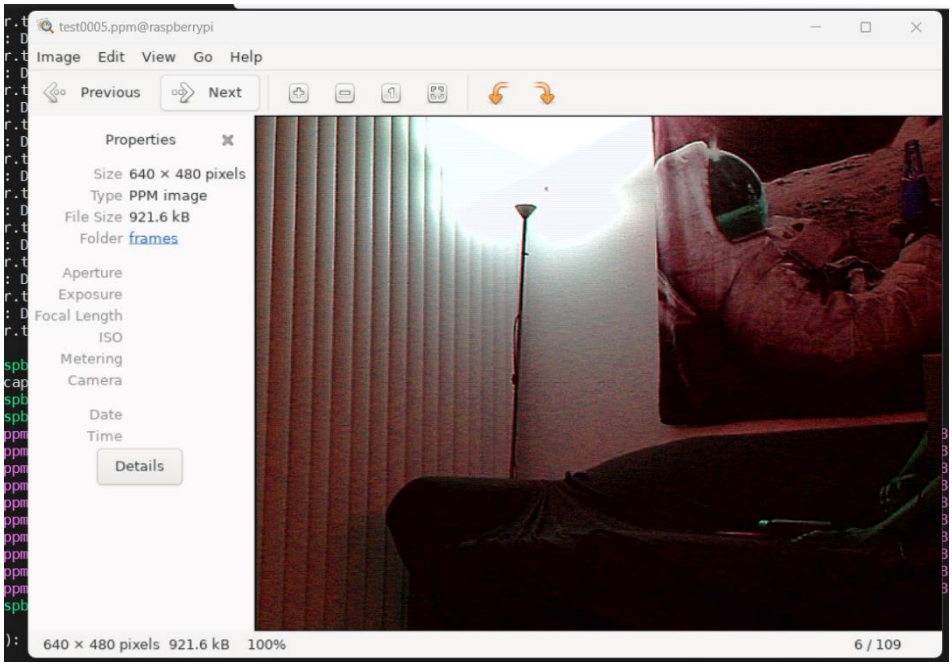


Figure 2: After transformation

b) Provide a detailed explanation of your code and research uses for the continuous transformations and describe how this might be used in a machine vision application.

1. open_device(): If no argument is passed while running the program, the default device file is '/dev/video0'. The function checks the status of the file with the system call- stat(). It checks if the device passed is a character device with the macro- S_ISCHR. Finally, the file is opened with both the read and write flags enabled.
2. init_device(): The function checks if the device opened is a V4L2 device and if it supports video capture. The method of IO used for capturing the video is "'IO_METHOD_MMAP' and for this, the function checks if the device supports streaming I/O. With xioctl() system calls, cropping capabilities are set and the video format is set to YUYV which works with the Logitech C270 camera.
3. init_mmap(): The requested video capture buffers' parameters are set and the buffers are requested from the device using a xioctl call. Memory is allocated for the buffers and the code enters a loop run for 6 times. Each requested buffer is mapped to the virtual address space of the calling process with a mmap() call.
4. start_capturing():After queueing the video capture buffers, the function calls xioctl() with the argument passed as VIDIOC_STREAMON to start the video streaming.
5. mainloop(): The frame rate is set and the loop runs until the specified number of frames are captured. The select() call is used to wait on the file descriptor for 'timeout' amount of time. For every frame, a call to read_frame() function is made.
6. read_frame(): The function dequeues the captured buffer in the kernel driver's outgoing queue to read the frame information from the camera. Then, a call to process_image() is made for each captured buffer ie. frame.
7. process_image(): The pixel format from the camera is checked to be set as YUYV and depending on the macro- COLOR_CONVERT_RBG, the YUYV pixels are transformed to RGB or YY ie. black and white. The function yuv2rgb() is called to do the former. The function reads 4 bytes of YUV and outputs 6 bytes of RGB data. For BnW image requirements, alternating bytes of data ie. Y are taken.
8. sharpen_image(): The function performs an image convolution algorithm which is set to sharpening based on the passed PSF coefficients. Based on these new pixel values, the passed buffers will now hold a sharpened image.
9. dump_ppm() or dump_pgm(): Based on the code requirements, the image is stored in a ppm or pgm format for rgb or bnw. A file to store the image is created using open() system call in create mode and non-blocking mode. The header consists information like image dimensions, timestamp. All of this along with the buffer passed is written into the file.
10. stop_capturing(): The video stream is turned off by passing the VIDIOC_STREAMOFF as an arguments to the xioctl call.
11. uninit_device(): The function calls mumap() to remove mapping for all the pages in the process allotted in the earlier mmap() call.

12. close_device(): The file descriptor for the device is closed using the close() system call.

Applications: Sharpening enhances the edges and fine details in an image, increasing its visual clarity and emphasizing high-frequency components. It helps to enhance the image's crispness and improve its overall appearance. Sharpening can be used to improve the quality of the captured image. This has applications in medical image analysis to enhance medical scans like X-rays, CT scans. In machine vision and computer vision applications, object detection relies on identifying objects in a frame. Here, sharpening can be used in pre-processing step to enhance edges and make objects more distinct.

c) What was the best frame rate for continuous transform processing for the transform you chose to investigate? Note that your real bottleneck may be writing your frames to your flash file system device, so try disabling frame write-back to the frame sub-directory and transforming frames, but not saving them

The best frame rate for continuous transform processing for 'sharpening' is 7.75 FPS. This is shown in the image below. The image transformation effects the WCET time and thus the frame rate is considerably lower.

```
anuhya@raspberrypi:~/Desktop/frame_rate_transformation $ sudo ./capture
FORCING FORMAT
allocated buffer 0
allocated buffer 1
allocated buffer 2
allocated buffer 3
allocated buffer 4
allocated buffer 5
Running at 30 frames/sec
frame -7: frame -6: frame -5: frame -4: frame -3: frame -2: frame -1: frame 0: frame 1: fra
: frame 10: frame 11: frame 12: frame 13: frame 14: frame 15: frame 16: frame 17: frame 18:
25: frame 26: frame 27: frame 28: frame 29: frame 30: frame 31: frame 32: frame 33: frame 3
e 41: frame 42: frame 43: frame 44: frame 45: frame 46: frame 47: frame 48: frame 49: frame
ame 57: frame 58: frame 59: frame 60: frame 61: frame 62: frame 63: frame 64: frame 65: fra
frame 73: frame 74: frame 75: frame 76: frame 77: frame 78: frame 79: frame 80: frame 81: f
: frame 89: frame 90: frame 91: frame 92: frame 93: frame 94: frame 95: frame 96: frame 97:
ame 104: frame 105: frame 106: frame 107: frame 108: frame 109: frame 110: frame 111: frame
118: frame 119: frame 120: frame 121: frame 122: frame 123: frame 124: frame 125: frame 126
 frame 133: frame 134: frame 135: frame 136: frame 137: frame 138: frame 139: frame 140: fr
me 147: frame 148: frame 149: frame 150: frame 151: frame 152: frame 153: frame 154: frame
61: frame 162: frame 163: frame 164: frame 165: frame 166: frame 167: frame 168: frame 169:
frame 176: frame 177: frame 178: frame 179: frame 180: frame 181: frame 182:
 Total capture time=23.355448, for 182 frames, 7.792615 FPS
Frame acquisition avg dt : 0.000024 or 41636.897424 FPS
Frame processing avg dt : 0.012814 or 78.037171 FPS
Frame storage avg dt : 0.003255 or  307.175595 FPS
Avg WCET=0.133167, Max WCET=0.173099
```

5) [30 points] Using a Logitech C270 (or equivalent), choose ONE real-time frame transformation and measure the frame processing time (WCET). I have made some videos on the "L-N9.#-Final-Project" (found in Coursera Video) that demonstrate the analysis of a single thread design (simple-capture-1800) and use of syslog for tracing (CW-21-C4-simplecapture-1800-starter-code-for-project) and you may also find simple-capture-1800-syslog/ useful.

The WCET, average frame rate and average time taken for each step of acquisition, transformation and write-back is calculated for the capturing a frame without the transformation initially.

Process	Time taken (seconds)	FPS
Total capture time	9.193875	19.7957
Frame acquisition time (avg)	0.000024	41025.59
Frame processing time (avg)	0.012961	77.15
Frame storage time (avg)	0.003277	305.1131
WCET (avg)	0.052606	
WCET (max)	0.074565	

The code is in the folder-'frame_rate_wo_transformation'. The total capture time is 9.1939 seconds for 182 frames ie. the frame rate is 19.795 FPS.

```
anuhya@raspberrypi:~/Desktop/frame_rate_wo_transformation $ sudo ./capture
FORCING FORMAT
allocated buffer 0
allocated buffer 1
allocated buffer 2
allocated buffer 3
allocated buffer 4
allocated buffer 5
Running at 30 frames/sec
frame -7: frame -6: frame -5: frame -4: frame -3: frame -2: frame -1: frame 0:
: frame 10: frame 11: frame 12: frame 13: frame 14: frame 15: frame 16: frame
25: frame 26: frame 27: frame 28: frame 29: frame 30: frame 31: frame 32: frame
e 41: frame 42: frame 43: frame 44: frame 45: frame 46: frame 47: frame 48: fra
ame 57: frame 58: frame 59: frame 60: frame 61: frame 62: frame 63: frame 64:
frame 73: frame 74: frame 75: frame 76: frame 77: frame 78: frame 79: frame 80
: frame 89: frame 90: frame 91: frame 92: frame 93: frame 94: frame 95: frame
ame 104: frame 105: frame 106: frame 107: frame 108: frame 109: frame 110: fram
118: frame 119: frame 120: frame 121: frame 122: frame 123: frame 124: frame 12
 frame 133: frame 134: frame 135: frame 136: frame 137: frame 138: frame 139:
me 147: frame 148: frame 149: frame 150: frame 151: frame 152: frame 153: frame
61: frame 162: frame 163: frame 164: frame 165: frame 166: frame 167: frame 168
frame 176: frame 177: frame 178: frame 179: frame 180: frame 181: frame 182:
 Total capture time=9.193876, for 182 frames, 19.795785 FPS
Frame acquisition avg dt : 0.000024 or 41025.594787 FPS
Frame processing avg dt : 0.012961 or 77.151925 FPS
Frame storage avg dt : 0.003277 or  305.113155 FPS
Avg WCET=0.052606, Max WCET=0.074565
```

a) Please implement and compare transform performance in terms of average transform processing time and overall average frame rate for acquisition, transformation and writeback of only the transformed frame. Note the average for each step clearly (acquisition, processing, write-back).

The average frame processing rate and average time taken for each step of acquisition, transformation and write-back is calculated in the code. This is in the folder- 'frame_rate_transformation'. The time taken for every frame acquisition, transformation and write back is printed in the syslog file. The output is transferred to the excel sheet- 'transform processing times'. The code captures 182 frames. The average and max WCET is shown in the table below.

Process	Time taken (seconds)	FPS
Total capture time	23.355	7.79
Frame acquisition time (avg)	0.000024	41636.89
Frame processing time (avg)	0.012814	78.03
Frame storage time (avg)	0.003255	307.17
WCET (avg)	0.133167	
WCET (max)	0.173099	

```
anuhya@raspberrypi:~/Desktop/frame_rate_transformation $ sudo ./capture
FORCING FORMAT
allocated buffer 0
allocated buffer 1
allocated buffer 2
allocated buffer 3
allocated buffer 4
allocated buffer 5
Running at 30 frames/sec
frame -7: frame -6: frame -5: frame -4: frame -3: frame -2: frame -1: frame 0: frame 1: fra
: frame 10: frame 11: frame 12: frame 13: frame 14: frame 15: frame 16: frame 17: frame 18
25: frame 26: frame 27: frame 28: frame 29: frame 30: frame 31: frame 32: frame 33: frame
e 41: frame 42: frame 43: frame 44: frame 45: frame 46: frame 47: frame 48: frame 49: frame
ame 57: frame 58: frame 59: frame 60: frame 61: frame 62: frame 63: frame 64: frame 65: fra
frame 73: frame 74: frame 75: frame 76: frame 77: frame 78: frame 79: frame 80: frame 81:
: frame 89: frame 90: frame 91: frame 92: frame 93: frame 94: frame 95: frame 96: frame 97
ame 104: frame 105: frame 106: frame 107: frame 108: frame 109: frame 110: frame 111: frame
118: frame 119: frame 120: frame 121: frame 122: frame 123: frame 124: frame 125: frame 126
 frame 133: frame 134: frame 135: frame 136: frame 137: frame 138: frame 139: frame 140: fr
me 147: frame 148: frame 149: frame 150: frame 151: frame 152: frame 153: frame 154: frame
61: frame 162: frame 163: frame 164: frame 165: frame 166: frame 167: frame 168: frame 169
frame 176: frame 177: frame 178: frame 179: frame 180: frame 181: frame 182:
 Total capture time=23.355448, for 182 frames, 7.792615 FPS
Frame acquisition avg dt : 0.000024 or 41636.897424 FPS
Frame processing avg dt : 0.012814 or 78.037171 FPS
Frame storage avg dt : 0.003255 or  307.175595 FPS
Avg WCET=0.133167, Max WCET=0.173099
```

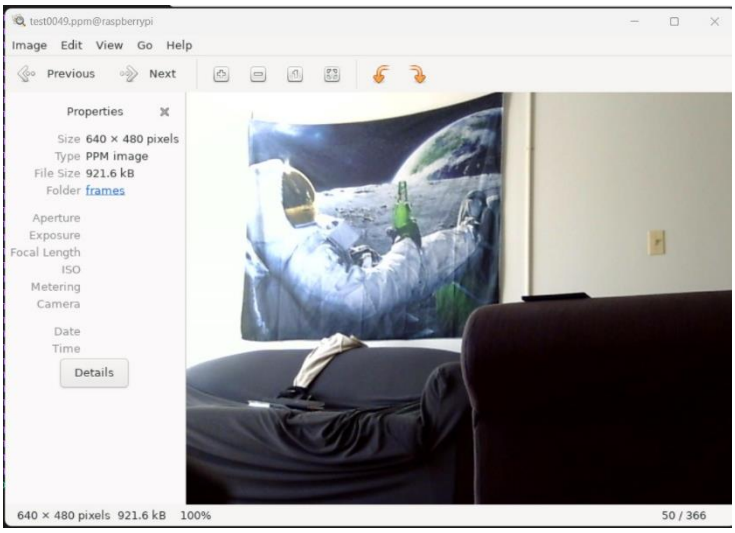
When the WCET for the transformed and captured frame are compared, it can be seen that the transformed frame takes longer amount of time for processing the frame and thus results in a worse frame rate.

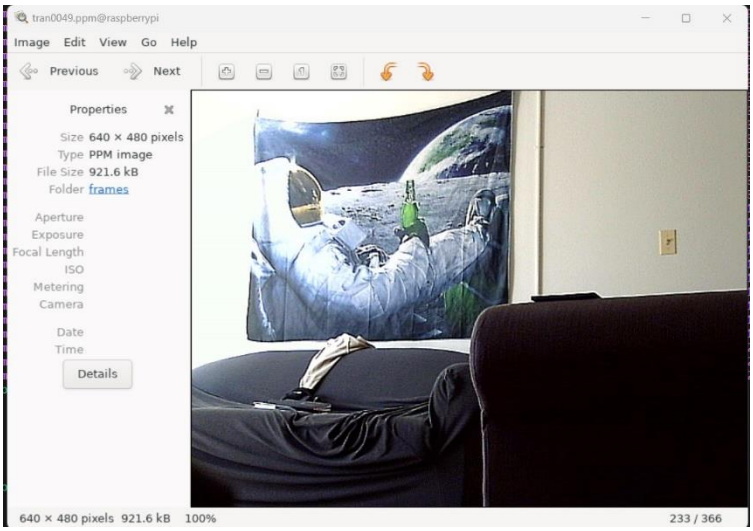
b) Please implement and compare transform performance in terms of average frame rate for transformation and write-back of both the transformed frame and captured.

The average frame rate for transformation and write-back of both the transformed frame and captured frame is calculated in the code. The capture frame is saved as 'test<frame_number>' while the transformed frame is saved as 'tran<frame_number>'. While the total capture time is 23.95, the average frames per second (FPS) is 15.114. The code is in the folder 'frame_rate_both_frames'. The captured and transformed image are shown in the next page. The program output is shown in the figure below.

```
anuhya@raspberrypi:~/Desktop/simple-capture-1800-syslog $ sudo ./capture
FORCING FORMAT
allocated buffer 0
allocated buffer 1
allocated buffer 2
allocated buffer 3
allocated buffer 4
allocated buffer 5
Running at 30 frames/sec
frame -7: frame -6: frame -5: frame -4: frame -3: frame -2: frame -1: frame 0: frame 1: frame 2: frame 3: frame 4: frame 5: frame 6: frame 7: frame 8: frame 9
: frame 10: frame 11: frame 12: frame 13: frame 14: frame 15: frame 16: frame 17: frame 18: frame 19: frame 20: frame 21: frame 22: frame 23: frame 24: frame
25: frame 26: frame 27: frame 28: frame 29: frame 30: frame 31: frame 32: frame 33: frame 34: frame 35: frame 36: frame 37: frame 38: frame 39: frame 40: fram
e 41: frame 42: frame 43: frame 44: frame 45: frame 46: frame 47: frame 48: frame 49: frame 50: frame 51: frame 52: frame 53: frame 54: frame 55: frame 56: fr
ame 57: frame 58: frame 59: frame 60: frame 61: frame 62: frame 63: frame 64: frame 65: frame 66: frame 67: frame 68: frame 69: frame 70: frame 71: frame 72:
frame 73: frame 74: frame 75: frame 76: frame 77: frame 78: frame 79: frame 80: frame 81: frame 82: frame 83: frame 84: frame 85: frame 86: frame 87: frame 88
: frame 89: frame 90: frame 91: frame 92: frame 93: frame 94: frame 95: frame 96: frame 97: frame 98: frame 99: frame 100: frame 101: frame 102: frame 103: fr
ame 104: frame 105: frame 106: frame 107: frame 108: frame 109: frame 110: frame 111: frame 112: frame 113: frame 114: frame 115: frame 116: frame 117: frame
118: frame 119: frame 120: frame 121: frame 122: frame 123: frame 124: frame 125: frame 126: frame 127: frame 128: frame 129: frame 130: frame 131: frame 132:
 frame 133: frame 134: frame 135: frame 136: frame 137: frame 138: frame 139: frame 140: frame 141: frame 142: frame 143: frame 144: frame 145: frame 146: fra
me 147: frame 148: frame 149: frame 150: frame 151: frame 152: frame 153: frame 154: frame 155: frame 156: frame 157: frame 158: frame 159: frame 160: frame 1
61: frame 162: frame 163: frame 164: frame 165: frame 166: frame 167: frame 168: frame 169: frame 170: frame 171: frame 172: frame 173: frame 174: frame 175:
frame 176: frame 177: frame 178: frame 179: frame 180: frame 181: frame 182:
 Total capture time=23.950957, for 364 frames, 15.114219 FPS

anuhya@raspberrypi:~/Desktop/simple-capture-1800-syslog $ cd frames/ ls
-bash: cd: too many arguments
anuhya@raspberrypi:~/Desktop/simple-capture-1800-syslog $ cd frames
anuhya@raspberrypi:~/Desktop/simple-capture-1800-syslog/frames $ ls
test0000.ppm  test0034.ppm  test0068.ppm  test0102.ppm  test0136.ppm  test0170.ppm  tran0021.ppm  tran0055.ppm  tran0089.ppm  tran0123.ppm  tran0157.ppm
test0001.ppm  test0035.ppm  test0069.ppm  test0103.ppm  test0137.ppm  test0171.ppm  tran0022.ppm  tran0056.ppm  tran0090.ppm  tran0124.ppm  tran0158.ppm
test0002.ppm  test0036.ppm  test0070.ppm  test0104.ppm  test0138.ppm  test0172.ppm  tran0023.ppm  tran0057.ppm  tran0091.ppm  tran0125.ppm  tran0159.ppm
```





c) Based on average analysis for transform frame only write-back, pick a reasonable soft real-time deadline (e.g., if average frame rate is 10 Hz, choose a deadline of 100 milliseconds) and convert the processing to SCHED_FIFO and determine if you can meet deadlines with predictability. Note, here are some examples of monotonic service analysis and jitter (here). The drift is shown as a red polynomial trend line and jitter is a plot of raw delta-t data compared to expected.

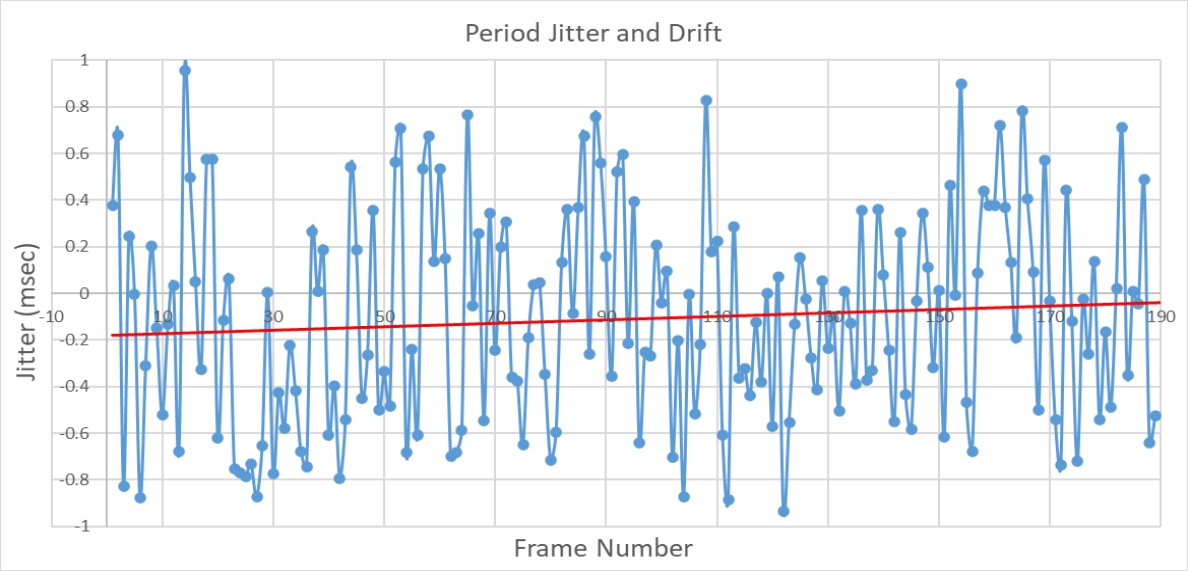
The full processing is converted to a service thread- this includes image acquisition, image processing, transformation and write-back to save the frame. The mainloop function is essentially run twice- once to calculate the WCET to meet the deadlines with predictability and the second time to calculate the jitter from the set deadline in the previous step.

As shown in the graph 1 (shown in the next page), the system meets the deadlines accurately. The jitter seen is positive and negative but is always within the range of 1 millisecond. The drift can be seen to be 0.1 millisecond from the graph. The deadline is set as the avg. WCET to show a variation in the jitter.

```
anuhya@raspberrypi:~/Desktop/wcet calculation $ sudo ./capture
System has 4 processors configured and 4 available.
rt_max_prio=99
rt_min_prio=1
Pthread Policy is SCHED_FIFO
Service threads will run on 1 CPU cores
FORCING FORMAT
allocated buffer 0
allocated buffer 1
allocated buffer 2
allocated buffer 3
allocated buffer 4
allocated buffer 5
Service thread entry
 Deadline is calculated to be 0.128107

 Total jitter = -0.020723

 Total capture time=47.513919, for 378 frames, 7.955564 FPS
Avg WCET=0.128107, Max WCET=0.186314
```



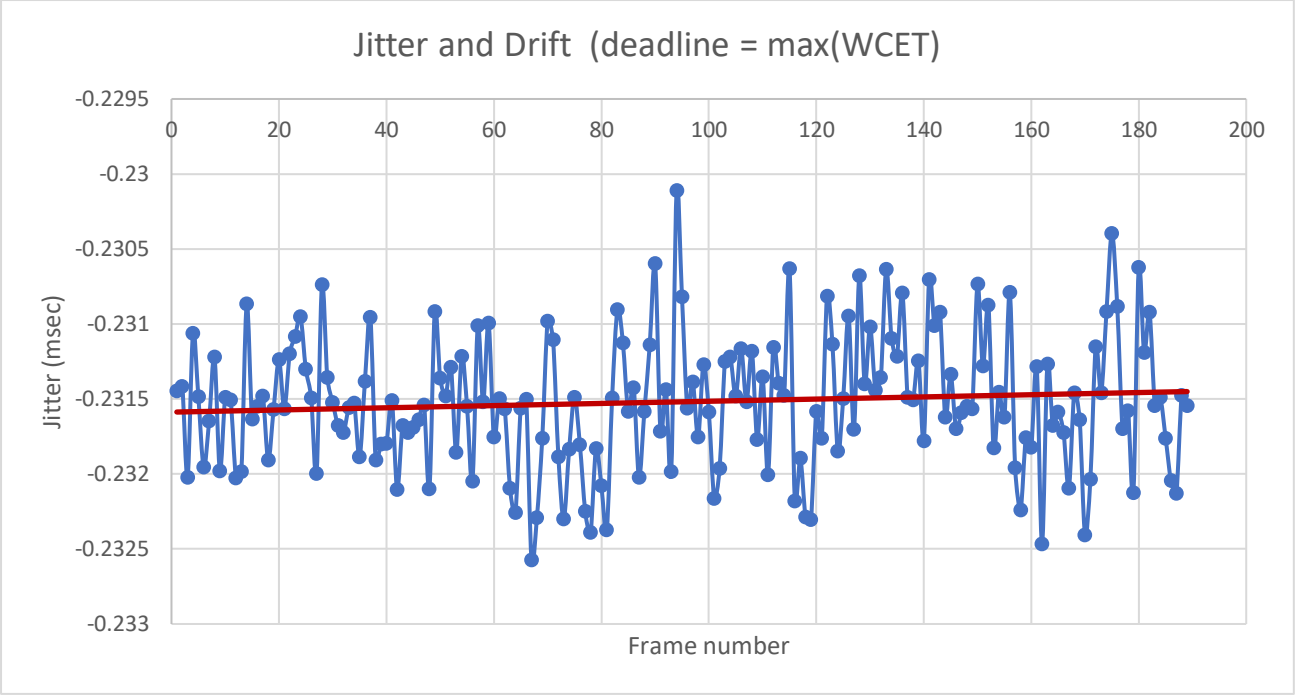
Graph 1: Jitter and drift in the predictable analysis when deadline is set to average of WCETs.

In the case of where the deadline is set as the maximum of all WCETs, all the services will meet the deadlines and show only negative jitter (ie. service time would be completed before the deadline). However, ideally to ensure that frame processing meets the deadline, the deadline would be set as the maximum of all WCETs. This is shown in the figure and graph 2 below.

```
anuhya@raspberrypi:~/Desktop/code/question_5/jitter_analysis $ sudo ./capture
System has 4 processors configured and 4 available.
rt_max_prio=99
rt_min_prio=1
Pthread Policy is SCHED_FIFO
Service threads will run on 1 CPU cores
FORCING FORMAT
allocated buffer 0
allocated buffer 1
allocated buffer 2
allocated buffer 3
allocated buffer 4
allocated buffer 5
Service thread entry
 Deadline is calculated to be 0.360643

 Total jitter = -43.757044

 Total capture time=48.170246, for 378 frames, 7.847168 FPS
Avg WCET=0.130503, Max WCET=0.360643
```



Graph 2: Jitter and drift in the predictable analysis when deadline is set of the maximum of all WCETs.