ECEN5623, Real-Time Systems:

Exercise #4 – Real-Time Continuous Media

DUE: As Indicated on Canvas

Please thoroughly read Chapters 7, 8 & 16 in the text.

Please see example code provided - Linux/code/ Note that the computer_vision_cv3_tested versions of the code have specifically been updated and tested on the Jetson upgrade for OpenCV 3.1, which is recommended (OpenCV4 is not tuned to the example code and not recommended). See installation instructions for OpenCV on the DE1-SoC on Canvas if using the DE1-SoC, and similar for the Raspberry Pi and Jetson. On the Jetson it is recommended that you revert to Jetpack 4.3 to avoid compatibility issues with OpenCV and applications that appear on later versions of Jetpack.
This lab introduces the use of cameras for computer vision applications for emergent real-time systems such as intelligent transportation, but also for traditional real-time instrumentation applications including machine vision, digital imaging for science and defense, and avionics instrumentation.

Exercise #4 Requirements:

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

2) [10 points] Camera Tools – You only need to do either Option 1 or Option 2.
   Option 1: Camorama
   If you do not have ***camorama***, do ***apt-get install camorama*** on your DE1-SoC, Raspberry Pi, or Jetson board [you may need to first do ***sudo add-apt-repository universe; sudo apt-get update***].
   This should not only install nice camera capture GUI tools, but also the V4L2 API (described well in this series of Linux articles - http://lwn.net/Articles/203924/). Running camorama should provide an interactive camera control session for your Logitech C2xx camera – if you have issues connecting to

your camera do a "man camorama" and specify your camera device file entry point (e.g. /dev/video0). Run camorama and play with Hue, Color, Brightness, White Balance and Contrast, take an example image and take a screen shot of the tool and provide both in your report. If camorama doesn't appear to work, then consider the next option, Cheese.

Option 2: Cheese

If you do not have ***cheese***, do ***sudo apt-get install cheese*** on your Jetson, Raspberry Pi or DE1-SoC board or other native Linux system. 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 cheese should provide an interactive camera control session for your Logitech C2xx camera – if you have issues connecting to your camera do a "man cheese" and specify your camera device file entry point (e.g. /dev/video0). Show that you tested your camera with a cheese screen dump and test photo.

3) [10 points] Using your verified Logitech C2xx camera on a DE1-SoC, Raspberry Pi or Jetson, verify that it can stream continuously to a raw image buffer for transformation and processing using any one of the below methods, either a) or b):

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

   (OR)

   b) 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".

Note: If folders have identical names, the presence of a `.c` file indicates it's for V4L2, while a `.cpp` file suggests it's for OpenCV. If there's any confusion, feel free to contact TA's for clarification. ***Stick to either option a or option b for consistency across questions 3, 4 and 5.***

4) [20 points]
   a) Choose a continuous transformation OpenCV example from computer vision cv3 tested such as the canny-interactive, hough-interactive, hough-eliptical-interactive, or stereo-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 by looking up API functions in the OpenCV manual (http://docs.opencv.org) and for stereo-transform-improved either implement or explain how you could make this work continuously rather than snapshot only.
   Repeat with a second choice of these transformations.

   (OR)

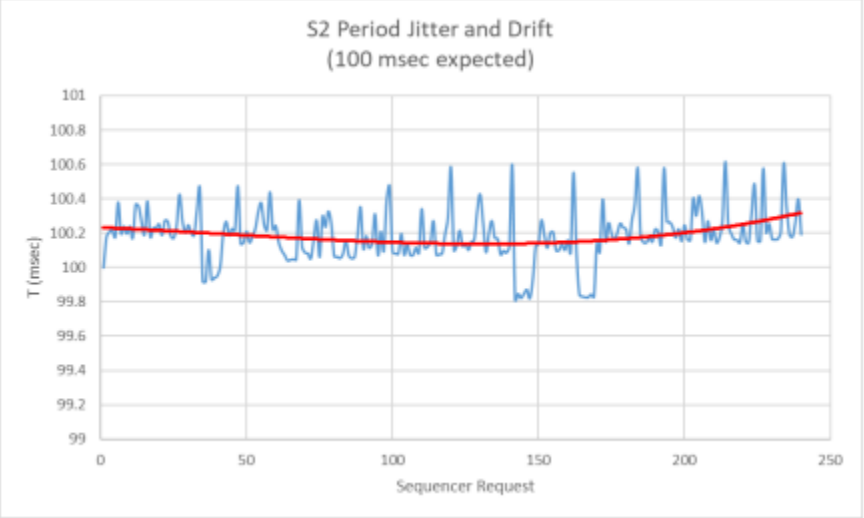
   b) Download starter code (simple-capture-1800) that captures 1800 frames from your camera using V4L2 and run it as is first to test the code (use "make clean", "make", and run – note that make clean will remove test frames). Modify the code so that it uses syslog rather than printf to the console for 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.

5) [40 points]
   a) Using a Logitech C2xx, choose 3 real-time interactive transformations to compare in terms of average frame rate at a given resolution for a range of at least 3 resolutions in a common aspect ratio (e.g. 4:3 for 1280x960, 640x480, 320x240, 160x120, 80x60) by adding time-stamps (there should be a logging thread) to analyze the potential throughput. You should then get at least 9 separate datasets running 1 transformation at a time. Based on average analysis, pick a reasonable soft real-time deadline (e.g. if average frame rate is 12 Hz, choose a deadline of 100 milliseconds to provide some margin) and convert the processing to SCHED_FIFO and determine if you can meet deadlines with predictability and measure statistical jitter in the frame rate relative to your deadline for both the default scheduler and SCHED_FIFO in each case.

   (OR)

   b) 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).
   Using a Logitech C270 (or equivalent), choose ONE real-time frame transformation and measure the frame processing time (WCET) and use of syslog for tracing. i) 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). ii) Please implement and compare transform performance in terms of average frame rate for transformation and write-back of both the transformed frame and captured. iii) 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 (RM-Ex-0-request-period-TRACE). The drift is shown as a red polynomial trend line and jitter is a plot of raw delta-t data compared to expected.



6) [10 points] Demonstrate the results and answer questions from the TA demonstrating that your camera is recognized, that you have camorama or cheese running, and that you can stream to an image buffer and do transformations as described in #1-#5 above.

Overall, provide a well-documented professional report of your findings, output, and tests so that it is easy for a colleague (or instructor) to understand what you've done. Include any C/C++ source code you write (or modify) and Makefiles needed to build your code. Any real-time control code must be in C/C++. I will look at your report first, so it must be well written and clearly address each problem providing clear and concise responses to receive credit. Note that it is up to you to pick an OOP for implementation and tools of your choice to develop and test your application – this is a large part of the challenge of this assignment.

Grading Rubric

[10 points] Camera and device driver verification:

Section Evaluation	Points Possible	Score	Comments
USB hot-plug	5		
UVC driver verify	5		
Total	10		

Provide screenshots of the three commands, with a description of each command and the output you see.

[10 points] Camera streaming checkout:

Section Evaluation	Points Possible	Score	Comments
Show the installation of either camorama or cheese	5		
Verification of camera control features including Screenshot of captured image	5		
Total	10		

Provide screenshots of the normal image and image with contrast/hue/brightness.

[10 points] Camera continuous streaming tests and applications:

Section Evaluation	Points Possible	Score	Comments
Install OpenCV and build code, modify it and document program execution -or- Use V4L2 APIs to test capture showing raw image frame	5		
Streaming verification by capture screenshot	5		
Total	10		

Explain the API's used with a brief description of the code.

[20 points] Continuous transformation tests and applications:

Section Evaluation	Points Possible	Score	Comments
Use OpenCV to demonstrate 2 transformations chosen from Canny, Hough Lines, Hough Elliptical, or Stereo Transform, and provide CPU loading by core and code description -or-	20		

-or- Use V4L2 and modify starter code to capture 1800 frames and compare PPM and PGM frame rates by capture screenshot with average and worst-case latency analysis			
Total	20		

Explain the API's used with a brief description of the code. For each vision algorithm, explain the code and show execution with screenshots.

[40 points] Soft real-time analysis and conversion to SCHED_FIFO with predictability analysis:

Section Evaluation	Points Possible	Score	Comments
Design concepts (Flow Chart or Diagram of your solution)	10		
Algorithm analysis (Description of your solution and each transform used)	10		
Prototype analysis including output screenshots	10		
Final predictable response jitter analysis	10		
-or-			
Apply transformation on real-time stream of image	10		
Average transform processing rate and time taken each step of acquisition, transformation and write-back	10		
Average frame rate for transformation and write-back of both frames	10		
Final predictable response jitter analysis (with write-back)	10		
Total	40		

The final predictable response jitter should be analyzed through a graph (Jitter vs No of Frames & No. of frames & Execution Time) and table for the various transformation techniques. Provide at least two graphs per transformation technique.

[10 points] TA Demonstration

Section Evaluation	Points Possible	Score	Comments
Individually answer questions about the Exercise sections 1-5	5		

Demonstrate to the TAs that your code for section 2 through 5 is complete and working.	5		
TOTAL	10		