Joey Carnicle

7/17/20

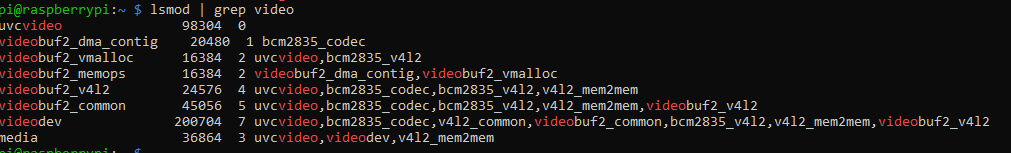
Github: <https://github.com/JoeyCarnicle/RTES_lab4>

Lab 4 Writeup

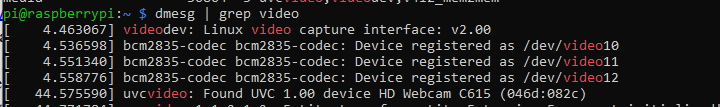
**Problem 1:**

1. 

LSUSB shows the connections made through the USB buses. Here we recognize the Logitech C615 on Bus 1.

1. 

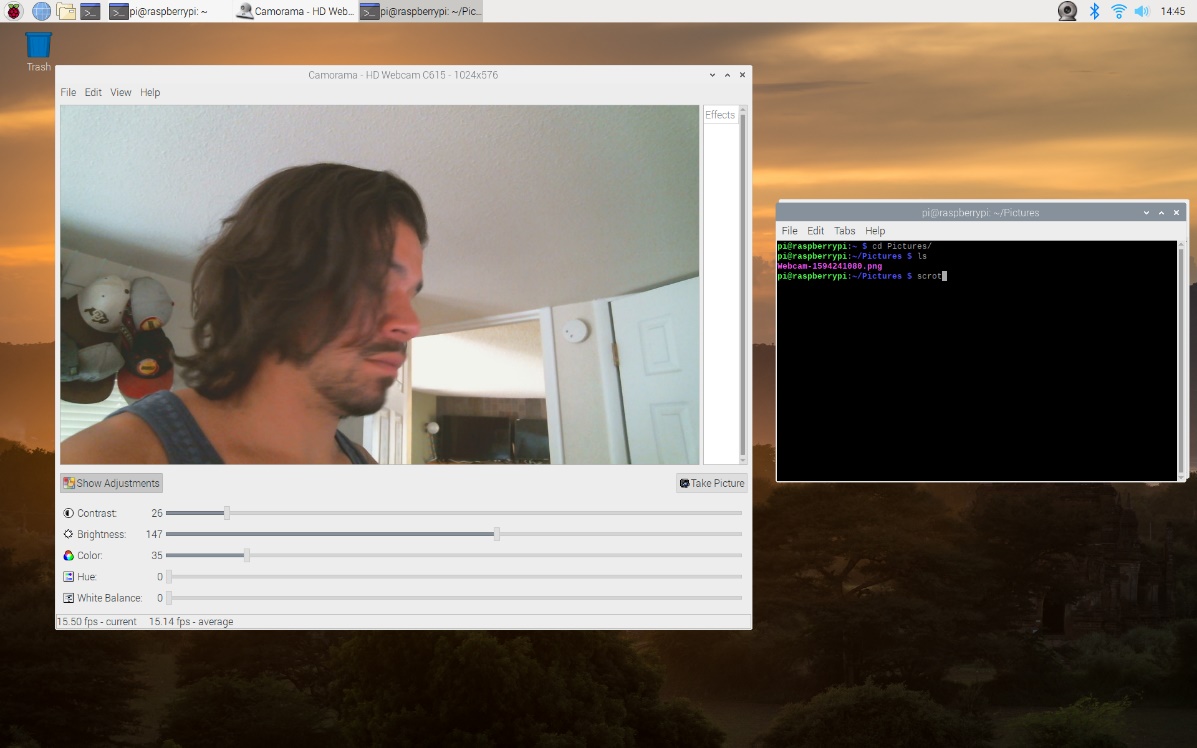
LSMOD shows the kernel modules that are loaded. Here the UVC video module has loaded appropriately for the webcam.

1. 

Dmesg displays the kernel ring buffer data for external interfacing. Here we search for video and see that the uvcvideo has found the C615 webcam.

1. Included in A, B, & C above.

**Problem 2:**

1. 

Screenshot of Camorama GUI.

1. 

Camera shot via Camorama using the C615 webcam.

**Problem 3:**

1. Screenshot of running the capture code: A screenshot of a computer

   Description automatically generated
2. I really enjoy learning the low level capture so I didn’t not explore OpenCV.

**Problem 4:**

1. Screenshot of the sharpen code:

A screenshot of a computer

Description automatically generated

1. The sharpen code is shifting each R, G, & B stored in the input PPM file and storing it in the output PPM file that the user inputs. The code opens the PPM file and steps through each pixel R, G, & B and uses the point spread function to sharpen each pixel, which it stores in convR, convG, & convB arrays and then writes these variables to the output PPM file.

Continuous transformation can be useful for various functions which include compression for storage of the files, recognition of changes in the file and altering system function, & continuous streaming with alteration to the initial camera picture (or any streaming technology).

1. Below is the screenshot of running the code captureupdate.c located in the github folder. The frame rates are logged using the syslog function. I added the sharpen transformation in the process\_image function after the YUYV is converted to RGB. The sharpen code is largely copied from the provided code, but updated to run through the bigbuffer[] array of RGB values and sharpen each with the PSF.

The average frame rate after the sharpen code is added is very slow, around 1 second (each log subtracts that frame time from the start time, so they are cumulative). I was unable to get this transformation to run much faster than this, I think this is best run as a post processing thread rather than continuous transformation.

A screenshot of a computer screen

Description automatically generated

**Problem 5:**

1. For this problem I chose to analyze YUYV to RGB, YUYV to RGB + sharpen, and YUYV to YY (Greyscale) at 640x480, 320x240, and 80x60 resolutions. The three cases are run in order for 30 frames each for a total of 90 frames at each resolution. This is the captureupdate2.c code in the github folder. The screenshots from the 3 resolutions are below:

320X240:

A screenshot of a computer

Description automatically generated

80x60:

A screenshot of a computer

Description automatically generated

680x460:

A screenshot of a computer

Description automatically generated

The average frame rate for each:

320x240:

RGB –39025 usec

RGB + Sharpen = 1195000 usec

Greyscale = 28526 usec

80x60:

RGB – 40465 usec

RGB + Sharpen = 1170475 usec

Greyscale = 27991 usec

640x480:

RGB – 39496 usec

RGB + Sharpen = 1205698 usec

Greyscale = 28988 usec

So the total averages are:

RGB – 39.67 msec

RGB + Sharpen = 1190.40 msec

Greyscale = 28.51 msec

Since the RGB + Sharpen is clearly too slow compared to the others, I will leave it out for the second part of Problem 4. The RGB and Greyscale transformations are averaging under 40msec or about 25Hz frame rate.

1. To give a margin, I attempted to keep the frame rate at 20Hz, or a deadline of 50msec, for this exercise. Captureupdatefinal.c changes the captureupdate2.c code to thread both the YUYV to RGB and Greyscale transformations on each frame. In the process\_image() function, the two threads are spawned with SCHED\_FIFO as the schedule. The frame\_time records the CLOCK\_REALTIME at the reception of the frame and the frame\_exit records the clock at the end of the RGB transformation and again at the end of the Greyscale transformation. The difference for each transformation is logged using syslog() and if the time is greater than 50msec, the deadmiss variable is incremented. At the end of the program the deadmiss value is printed and in the scenario of 90 frames it did not increment once. I have confidence that this program can meet the 20Hz deadline and jitter was measured to be around 5msec, as the longest recorded was 19msec and the shortest was 12msec to complete the thread. See the picture below for results :

