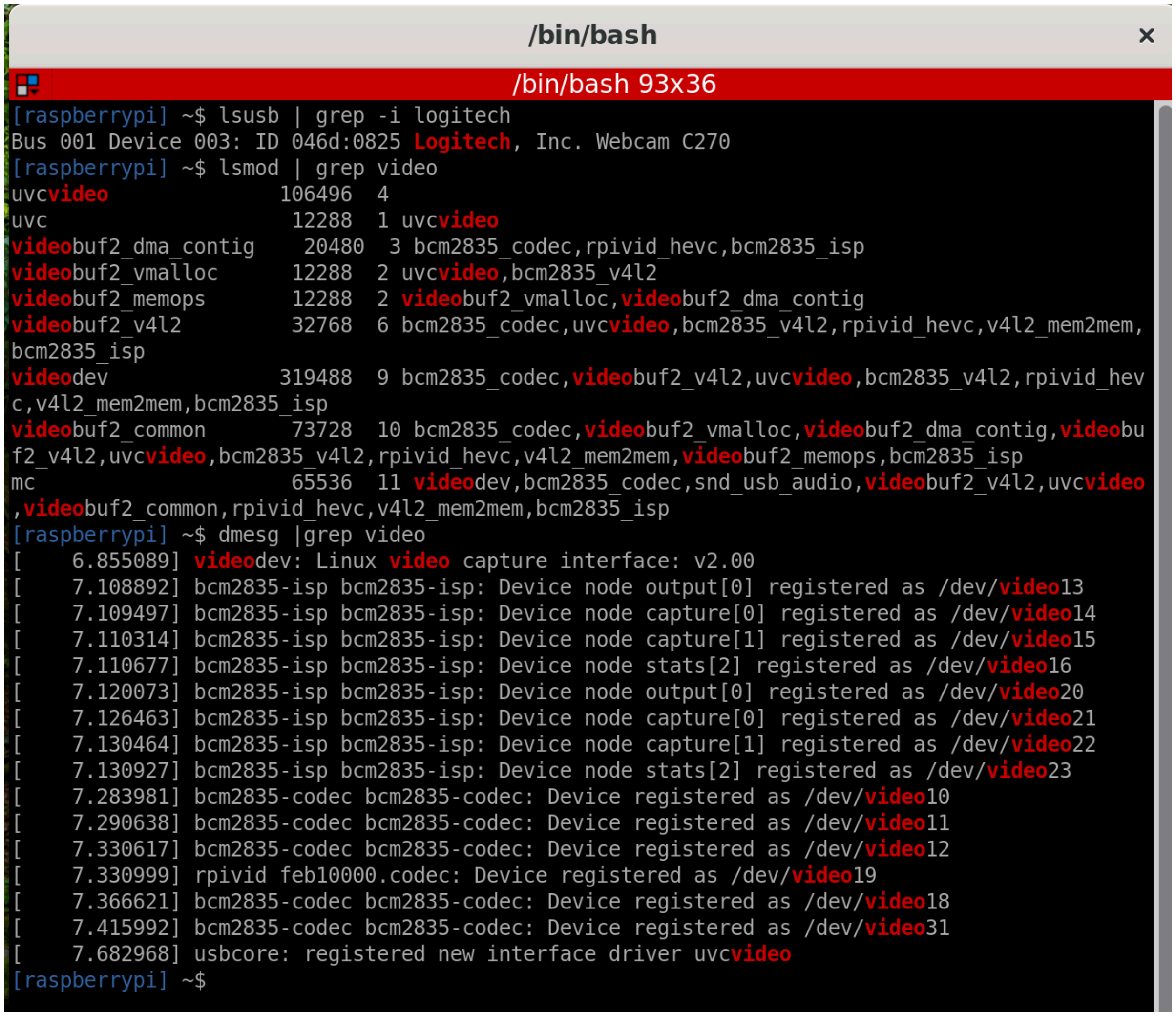
# ECEE 5623: Real-Time Systems

*Exercise 4: Real-Time Continuous Media*
*Due: July 12th, 2024*
*Jack Center*

## Part 1: Verify Camera

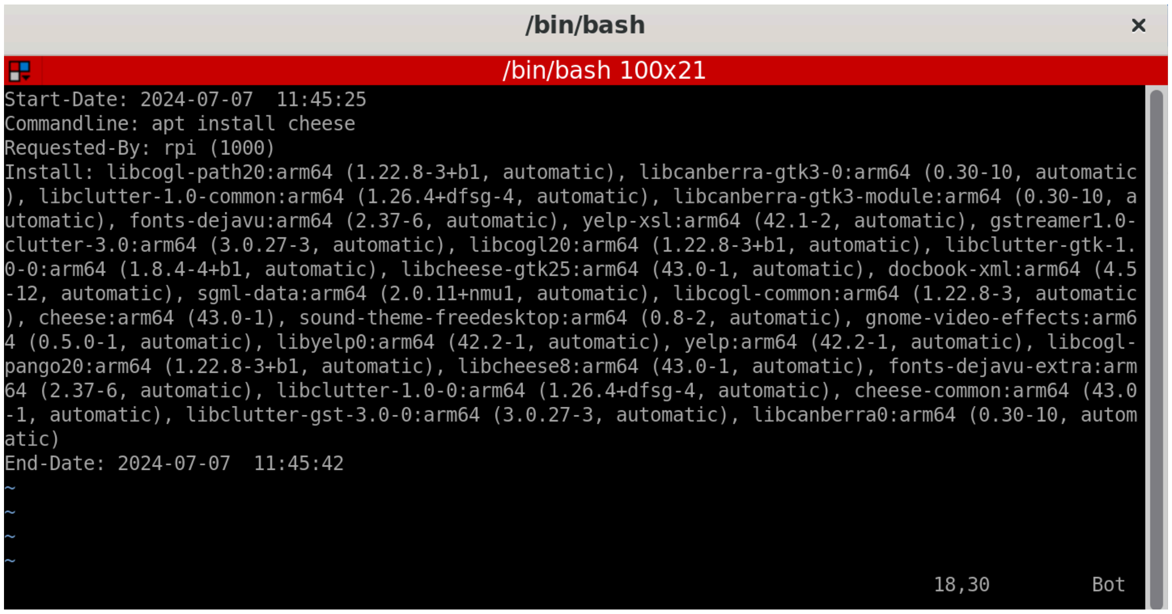


```
[raspberrypi] ~$ lsusb | grep -i logitech
Bus 001 Device 003: ID 046d:0825 Logitech, Inc. Webcam C270
[raspberrypi] ~$ lsmod | grep video
uvcvideo              106496  4
uvc                    12288  1 uvcvideo
videobuf2_dma_contig    20480  3 bcm2835_codec,rpivid_hevc,bcm2835_isp
videobuf2_vmalloc      12288  2 uvcvideo,bcm2835_v4l2
videobuf2_memops       12288  2 videobuf2_vmalloc,videobuf2_dma_contig
videobuf2_v4l2         32768  6 bcm2835_codec,uvcvideo,bcm2835_v4l2,rpivid_hevc,v4l2_mem2mem,
bcm2835_isp
videodev              319488  9 bcm2835_codec,videobuf2_v4l2,uvcvideo,bcm2835_v4l2,rpivid_hev
c,v4l2_mem2mem,bcm2835_isp
videobuf2_common       73728  10 bcm2835_codec,videobuf2_vmalloc,videobuf2_dma_contig,videobu
f2_v4l2,uvcvideo,bcm2835_v4l2,rpivid_hevc,v4l2_mem2mem,videobuf2_memops,bcm2835_isp
mc                     65536  11 videodev,bcm2835_codec,snd_usb_audio,videobuf2_v4l2,uvcvideo
,videobuf2_common,rpivid_hevc,v4l2_mem2mem,bcm2835_isp
[raspberrypi] ~$ dmesg |grep video
[    6.855089] videodev: Linux video capture interface: v2.00
[    7.108892] bcm2835-isp bcm2835-isp: Device node output[0] registered as /dev/video13
[    7.109497] bcm2835-isp bcm2835-isp: Device node capture[0] registered as /dev/video14
[    7.110314] bcm2835-isp bcm2835-isp: Device node capture[1] registered as /dev/video15
[    7.110677] bcm2835-isp bcm2835-isp: Device node stats[2] registered as /dev/video16
[    7.120073] bcm2835-isp bcm2835-isp: Device node output[0] registered as /dev/video20
[    7.126463] bcm2835-isp bcm2835-isp: Device node capture[0] registered as /dev/video21
[    7.130464] bcm2835-isp bcm2835-isp: Device node capture[1] registered as /dev/video22
[    7.130927] bcm2835-isp bcm2835-isp: Device node stats[2] registered as /dev/video23
[    7.283981] bcm2835-codec bcm2835-codec: Device registered as /dev/video10
[    7.290638] bcm2835-codec bcm2835-codec: Device registered as /dev/video11
[    7.330617] bcm2835-codec bcm2835-codec: Device registered as /dev/video12
[    7.330999] rpivid feb10000.codec: Device registered as /dev/video19
[    7.366621] bcm2835-codec bcm2835-codec: Device registered as /dev/video18
[    7.415992] bcm2835-codec bcm2835-codec: Device registered as /dev/video31
[    7.682968] usbcore: registered new interface driver uvcvideo
[raspberrypi] ~$
```

*Figure 1: Output from system commands for part 1.*

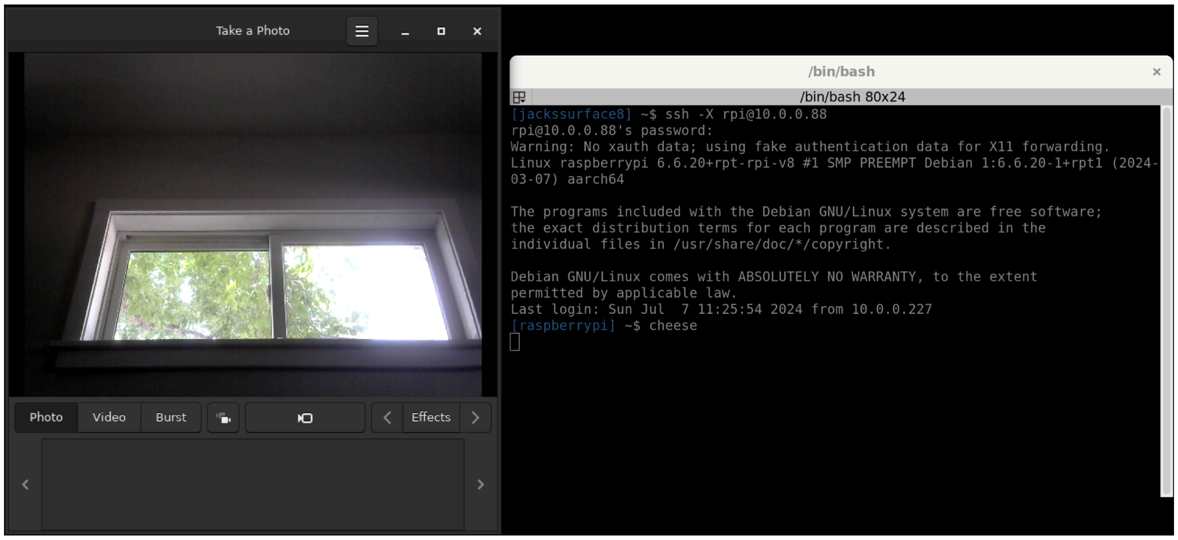
a. \`lsusb\` lists USB devices and shows the Logitech camera.
b. \`lsmod\` shows the status of modules and shows the UVC driver is loaded, the size of the module in bytes, and what is using the module.
c. \`dmesg\` shows the kernel ring buffer and shows messages stored in it. It shows the Linux video capture interface is in the buffer.

## Part 2: Install Cheese



```
Start-Date: 2024-07-07  11:45:25
Commandline: apt install cheese
Requested-By: rpi (1000)
Install: libcogl-path20:arm64 (1.22.8-3+b1, automatic), libcanberra-gtk3-0:arm64 (0.30-10, automatic
), libclutter-1.0-common:arm64 (1.26.4+dfsg-4, automatic), libcanberra-gtk3-module:arm64 (0.30-10, a
utomatic), fonts-dejavu:arm64 (2.37-6, automatic), yelp-xsl:arm64 (42.1-2, automatic), gstreamer1.0-
clutter-3.0:arm64 (3.0.27-3, automatic), libcogl20:arm64 (1.22.8-3+b1, automatic), libclutter-gtk-1.
0-0:arm64 (1.8.4-4+b1, automatic), libcheese-gtk25:arm64 (43.0-1, automatic), docbook-xml:arm64 (4.5
-12, automatic), sgml-data:arm64 (2.0.11+nmu1, automatic), libcogl-common:arm64 (1.22.8-3, automatic
), cheese:arm64 (43.0-1), sound-theme-freedesktop:arm64 (0.8-2, automatic), gnome-video-effects:arm6
4 (0.5.0-1, automatic), libyelp0:arm64 (42.2-1, automatic), yelp:arm64 (42.2-1, automatic), libcogl-
pango20:arm64 (1.22.8-3+b1, automatic), libcheese8:arm64 (43.0-1, automatic), fonts-dejavu-extra:arm
64 (2.37-6, automatic), libclutter-1.0-0:arm64 (1.26.4+dfsg-4, automatic), cheese-common:arm64 (43.0
-1, automatic), libclutter-gst-3.0-0:arm64 (3.0.27-3, automatic), libcanberra0:arm64 (0.30-10, autom
atic)
End-Date: 2024-07-07  11:45:42
~
~
~
~
                                                                                   18,30         Bot
```

*Figure 2.1: Summary log for installation of Cheese.*



```
[jackssurface8] ~$ ssh -X rpi@10.0.0.88
rpi@10.0.0.88's password:
Warning: No xauth data; using fake authentication data for X11 forwarding.
Linux raspberrypi 6.6.20+rpt-rpi-v8 #1 SMP PREEMPT Debian 1:6.6.20-1+rpt1 (2024-
03-07) aarch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Jul  7 11:25:54 2024 from 10.0.0.227
[raspberrypi] ~$ cheese
```

*Figure 2.2: GUI output for Cheese.*

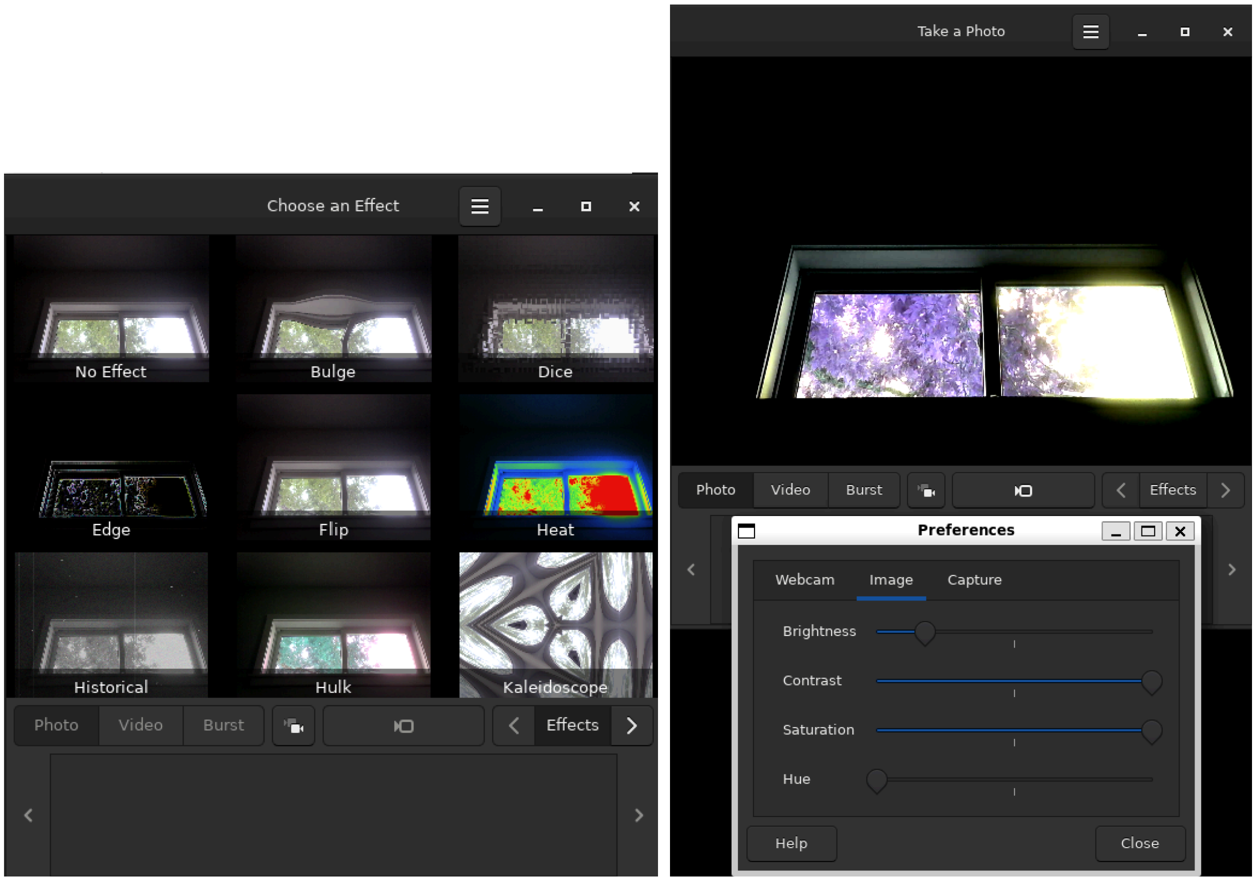
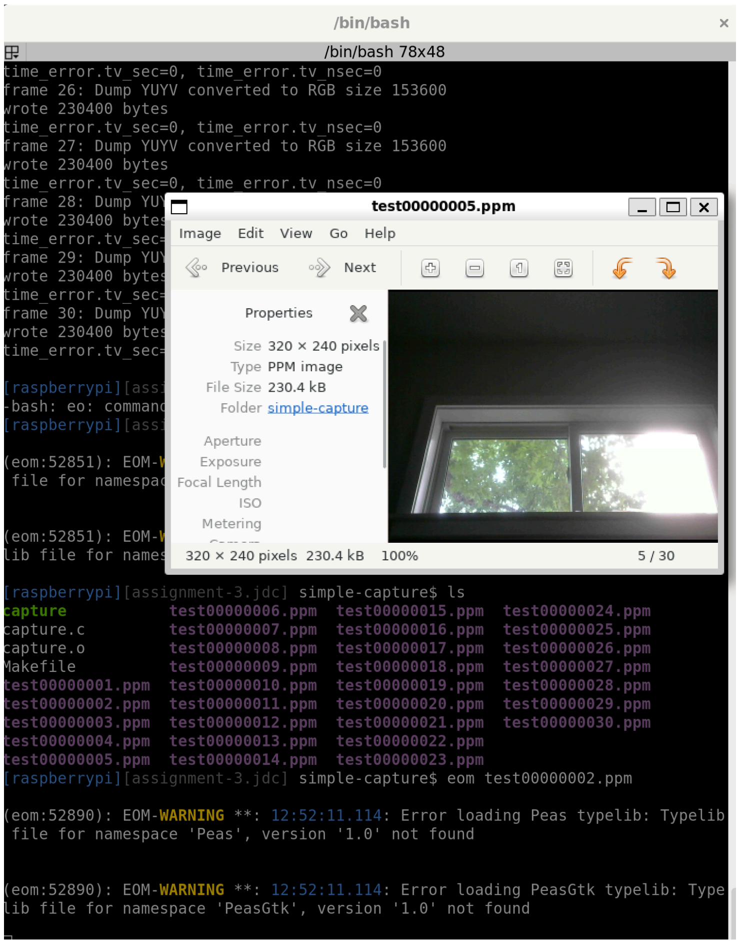


*Figure 2.3: Screenshot of effects tools and an example using it.*

## Part 3: Continuous Streaming

Figure 3 shows continuous capture is working. The image files can be seen in the terminal and eom window. Table 3 compares the results of processing 1,800 graymap and color frames. The median time per frame shows that graymap processing is about three times faster than color frame processing. This makes sense since there are three times the data in a color frame. The worst-case execution times varied significantly, which also brought up the mean execution time for the color frame processing. Considering the final project needs a 10Hz processing time, neither of these frame rates is sufficient to meet the Nyquist requirement of a 5Hz signal. This means a different method of capturing and processing the images will need to be implemented to separate the image capture from writing to flash so the correct rate can be achieved.



```
time_error.tv_sec=0, time_error.tv_nsec=0
frame 26: Dump YUYV converted to RGB size 153600
wrote 230400 bytes
time_error.tv_sec=0, time_error.tv_nsec=0
frame 27: Dump YUYV converted to RGB size 153600
wrote 230400 bytes
time_error.tv_sec=0, time_error.tv_nsec=0
frame 28: Dump YUY
wrote 230400 bytes
time_error.tv_sec=
frame 29: Dump YUY
wrote 230400 bytes
time_error.tv_sec=
frame 30: Dump YUY
wrote 230400 bytes
time_error.tv_sec=

[raspberrypi][assi
-bash: eo: command
[raspberrypi][assi

(eom:52851): EOM-W
 file for namespac


(eom:52851): EOM-W
lib file for names

[raspberrypi][assignment-3.jdc] simple-capture$ ls
capture           test00000006.ppm  test00000015.ppm  test00000024.ppm
capture.c         test00000007.ppm  test00000016.ppm  test00000025.ppm
capture.o         test00000008.ppm  test00000017.ppm  test00000026.ppm
Makefile          test00000009.ppm  test00000018.ppm  test00000027.ppm
test00000001.ppm  test00000010.ppm  test00000019.ppm  test00000028.ppm
test00000002.ppm  test00000011.ppm  test00000020.ppm  test00000029.ppm
test00000003.ppm  test00000012.ppm  test00000021.ppm  test00000030.ppm
test00000004.ppm  test00000013.ppm  test00000022.ppm
test00000005.ppm  test00000014.ppm  test00000023.ppm
[raspberrypi][assignment-3.jdc] simple-capture$ eom test00000002.ppm

(eom:52890): EOM-WARNING **: 12:52:11.114: Error loading Peas typelib: Typelib
 file for namespace 'Peas', version '1.0' not found


(eom:52890): EOM-WARNING **: 12:52:11.114: Error loading PeasGtk typelib: Type
lib file for namespace 'PeasGtk', version '1.0' not found
```

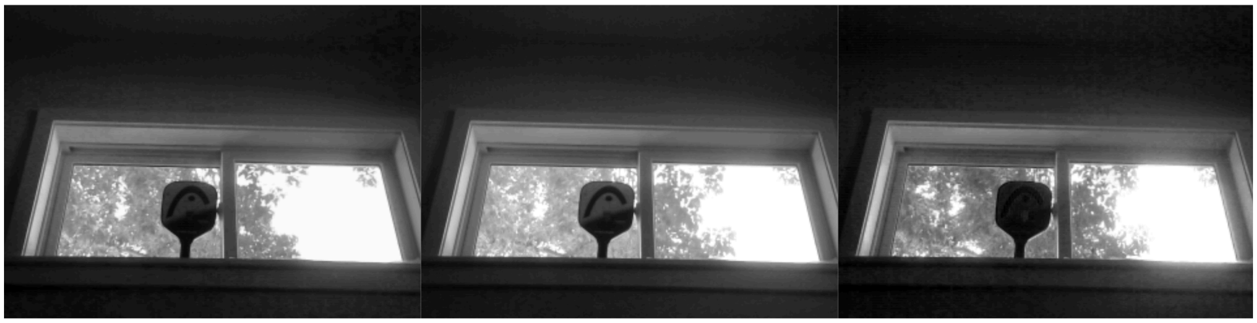
*Figure 3: Screenshot of using continuous capture.*

| | PGM | PPM |
|---|---:|---:|
| **Mean Execution Time (ms)** | 5.437 | 24.391 |
| **Median Execution Time (ms)** | 5.288 | 14.052 |
| **Worst Case Execution Time (ms)** | 133.798 | 3,558.454 |

*Table 3: Results from simple-capture-1800 on graymap (PGM) and color (PPM) frames.*

## Part 4: Continuous Transformation

I implemented conversion to grayscale using each of the single color bands and the balanced equation the GIMP uses. Overall, the balanced image is the highest quality, but the green band also does a good job since a lot of the image was green. The blue image is dark, but, as expected, the blue tape is more visible with this conversion.



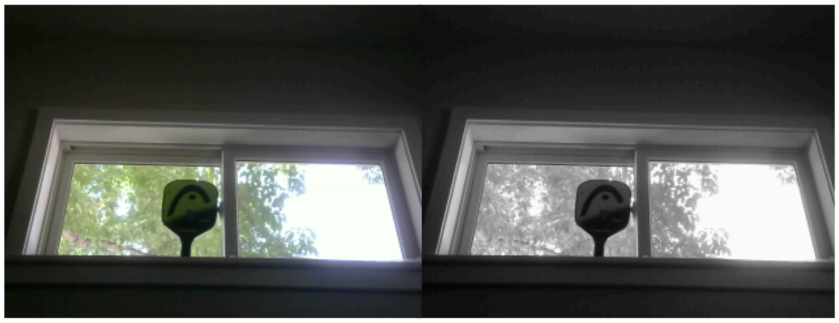


*Figure 4: Output from grayscale transformations using red, blue, and green color bands along with a balanced combination of the three based on the conversion used by GIMP.*

The code to complete this transformation was based on the starter code in \`simple-capture\`. There are three major phases outside of setup and teardown: image acquisition, transformation, and write-back to flash. The code updated for this project was in the \`process_image\` function under the color conversion option and mostly dealt with the image transformation. The updated algorithm takes the output from the YUV to RGB conversion and loops through each group of RGB values provided. Each of these groups is combined according to the balancing equation used by GIMP:

$$Y = 0.3R + 0.59G + 0.11B$$

These values, which are one-third the size of the original RGB data, are then written to a PGM file in flash memory.

Transformation such as this can be used in machine vision applications to focus in on relevant data. For example, if a camera is pointed at a stop light, it could be used to mask the incoming colors and only focus on red to identify if the light is red. This can be taken even further by focusing on a specific shade of red to isolate only the specific wavelength coming off of the stoplight. Or, this technique could be used to track an identifying marker on a vehicle which could then be used to calculate how many pixels it has moved from one frame to the next; this type of information could be used to predict or measure the object's motion. This narrowing of scope for what the image is seeing simplifies follow-on tasks that need to interpret what the image means and reduce the likelihood of noise interfering with the analysis.

The best frame rate for the transformation was measured to be 8.213 milliseconds (Table 5). This value is independent of writing to the flash file system since the time measurements for the three stages were separated for Part 5. The best frame rate time without this conversion was 3.018 milliseconds. This indicates that the grayscale transformation more than doubles the time the transformation takes without the conversion.

## Part 5: Deadline Analysis

The results for the transform performance are summarized in Table 5.a. and 5.b.:

| | Mean (ms) | Median (ms) | Worst (ms) | Best (ms) |
|---|---|---|---|---|
| **Acquisition** | 0.016 | 0.016 | 0.034 | 0.007 |
| **Transformation** | 12.298 | 12.419 | 24.349 | 8.213 |
| **Write-back** | 0.416 | 0.315 | 72.985 | 0.217 |
| **Total** | 12.788 | 12.815 | 85.466 | 8.515 |

*Table 5.a.: Timing metrics from capturing 1,800 color frames and converting them to grayscale using the balanced implementation described in Part 4.*

| | Mean (ms) | Median (ms) | Worst (ms) | Best (ms) |
|---|---|---|---|---|
| **Acquisition** | 0.018 | 0.018 | 0.047 | 0.011 |
| **Transformation** | 5.677 | 5.741 | 9.581 | 3.018 |
| **Write-back** | 0.653 | 0.484 | 127.801 | 0.314 |
| **Total** | 6.412 | 6.319 | 133.733 | 3.390 |

*Table 5.b.: Timing metrics for capturing 1,800 color frames. This is unique from Table 5.a. because this data does not execute the grayscale transform.*

Reviewing the performance shows that the transformation is typically the longest part of the procedure; however, the write-back can get backed up and cause substantial delays. These delays are substantial enough to cause missed deadlines. The grayscale transformation also has the effect of approximately doubling the total transformation time. This is likely due to how the current implementation first converts to RGB, then converts to grayscale. This could likely be improved by converting directly to grayscale. Overall, this analysis points to two major performance problems to be aware of in the final project: write-back can cause a significant delay and the transformation implementation details are important for service duration.

Based on the average analysis for the transform frame-only write-back, 15 milliseconds was selected for the deadline; however, initial analysis showed that the FIFO implementation was taking closer to 35 milliseconds on average, so that value was used instead for the 1,800 frames. The results in Figure 5, show significant variability in the time for a single iteration of the service to capture, transform, and write back a frame. The first service reliably takes much longer than the other processes, which indicates a need to make sure everything is ready to go before expecting the deadlines to be hit. After the first 250 frames, the process time returns from approximately 35 milliseconds back to the expected 12 millisecond range. However, there are still significant steps in average processing time in the figure that are currently unexplained.

Solving these steps will be crucial for getting predictable results. There are also some long outliers, but these can be explained by the write-back delays discussed previously.

Overall, this implementation does not predictably meet deadlines primarily due to these significant steps in the service duration.

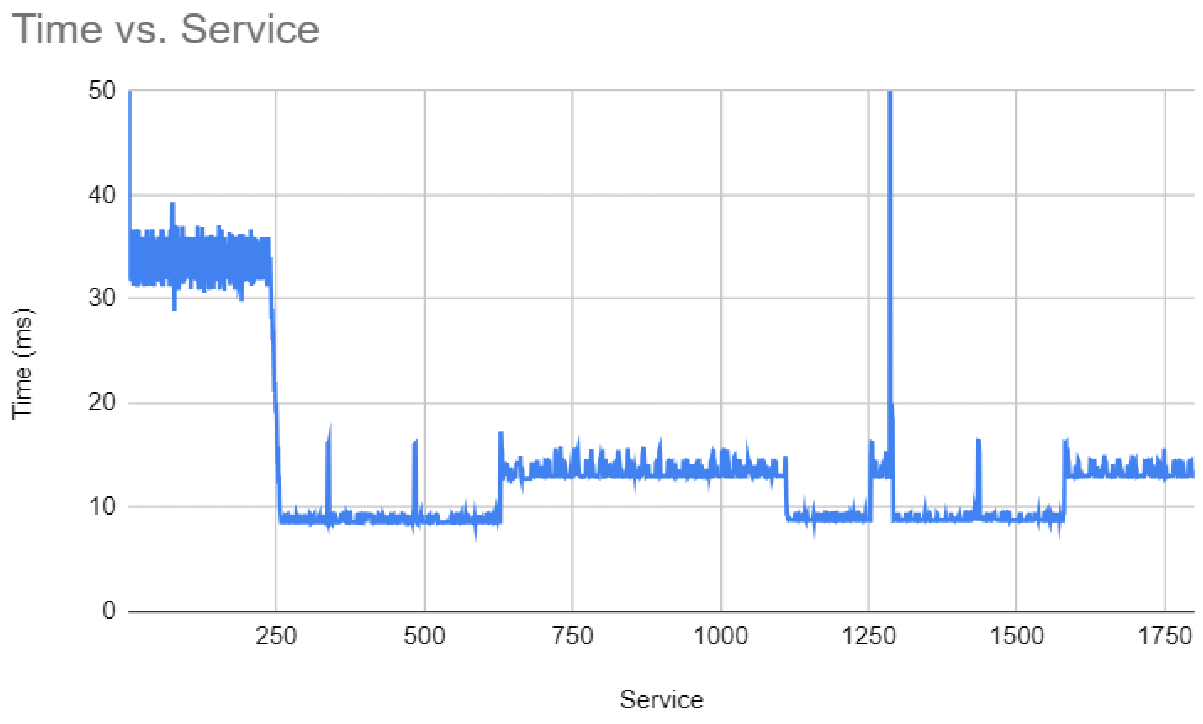


*Figure 5: The service duration in milliseconds versus the service number for transform frame-only write-back with a 35-millisecond deadline.*

## Appendix A: Code

Note: code was not included here for this assignment due to the large size of the files.