Measuring Camera Display Latency

Paul Rensing
FRC Team 2877
LigerBots

1 Abstract

Camera are used heavily in FRC Robotics, both for autonomous tracking of objects and driver assist. The latency of the camera image can make a significant difference to control algorithms and human response time when driving based on a camera stream. However, measuring the latency of the image stream is not commonly done, and is in fact not an easy task. We will talk about how we did this and give values for various camera and computer setups we have available. In particular, we show that Google Chrome is a suboptimal choice for a “live” video stream display.

2 Camera Latency

Latency, or lag, is the delay between when a event happens in real life and when the image or result can be acted upon. With respect to robots, there are actually two latencies of interest:

- the latency of collecting and processing (but not displaying) the image to find some parameters of interest (e.g. the position of an object, which is then used for navigation),
- the latency of collecting the image, transporting it across the network, and displaying it on the driver’s screen.

This paper is only going to talk about the second quantity. The “control” latency is guaranteed to be less than this, but can’t be measured in the same way.

3 Measuring Latency

A number of sites on the Internet talk about measuring latency by displaying a fast, digital clock in front of the display screen. You then point the camera at the clock and screen so that you get both in the same image. If you take a video of that whole feed, you can look at single frames and get the lag. You can see such a setup here:
https://makehardware.com/2016/03/29/finding-a-low-latency-webcam/.

The problem with this setup is that the fastest time digits (i.e. 10 and 1 msec place values) are typically highly smeared (at least whenever we have done it), so it can be very difficult to figure out the numbers and get accurate times. It is also not easy to collect statistics on the distribution of latency (see later). Instead, we chose to follow the setup described in this reference: https://oscarliang.com/fpv-camera-latency/.

The idea is to turn an LED on and off, creating “events” in the video stream. Using 2 photoresistors and a couple of other electric elements, you can measure both the time when the LED turns on directly, and the time you see it turn on on the display screen (i.e. the driver’s station screen). By measuring the signals using a good oscilloscope or digitizing with a computer, you can straightforwardly measure the difference between the two values and accumulate statistics on the spread.

4 Measurement Setup

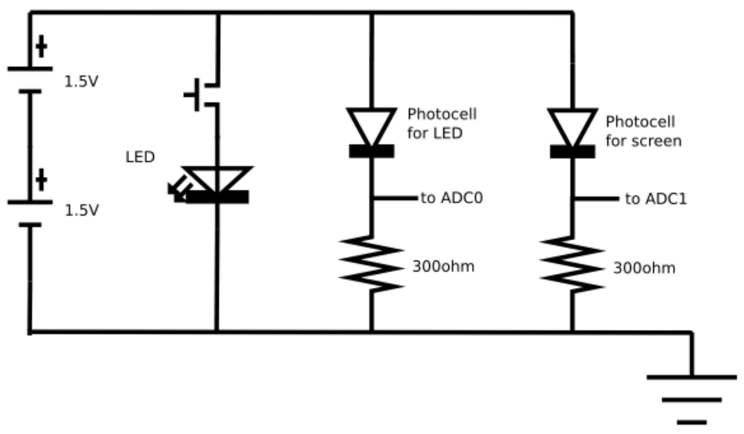

Figure 1: Electrical circuit for latency measurement

Figure 1 shows the electrical circuits we used. It consists of 3 basically independent parts. One drives the LED so we can turn it on and off, based on a physical push switch. (A signal generator or similar could be used if available, but it needs to be able to drive enough current for the LED, or drive a transistor or relay.) There are two identical circuits with a photoresistor (also referred to as a “photo cell”) and resistor to measure light output. The voltage across the resistors is fed (separately) to two ADC channels on a computer.

For reading the voltages, we used our ODROID-XU4, which has 2 ADC channels, plus an accurate clock. A RaspberryPi or similar computer would work just as well. If you want to use an Arduino, be aware that you need an accurate time signal, since that is the quantity

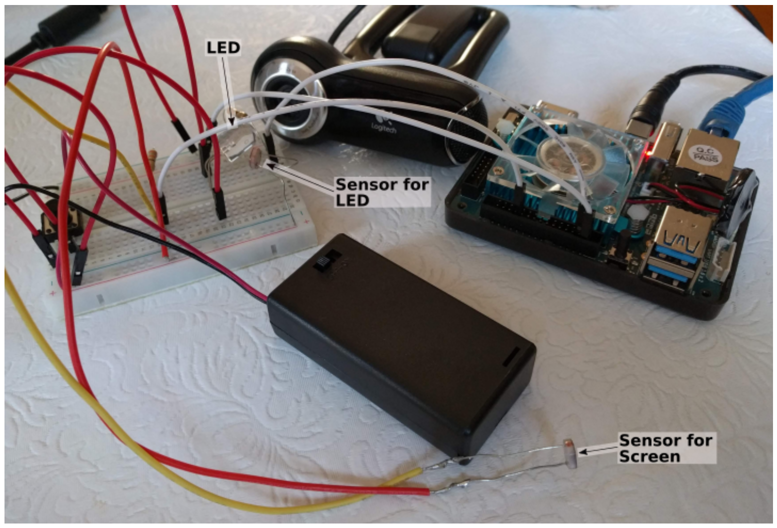

Figure 2: Physical setup of latency measurement

of interest. (We are not familiar enough with Arduinos to know how that should be done.)

In terms of the physical arrangement, you can see the circuit layout on Figure 2. One photoresistor is pointed directly at the LED to measure the immediate turn-on time. The other is connected using longer wires (about 1 foot) and is held directly at the display screen on top of the image of the photoresistor; this will give the time when the display screen "sees" the LED turn on.

For the data collection, we wrote a small Python program using the "wiringpi" module:

```
#!/usr/bin/python3

import sys
import wiringpi as wpi
from time import time, sleep
import csv

sleep_time = 0.005

# ADC0 is pin 3
# ADC3 (read(1)) is pin 23

wpi.wiringPiSetup()

csv_out = csv.writer(sys.stdout)
csv_out.writerow(('Time', 'ADC0', 'ADC3'))

startt = time()
while True:
    result = [time() - startt, wpi.analogRead(0), wpi.analogRead(1)]
    csv_out.writerow(result)

    sleep(sleep_time)
```

While the data are being collected and streamed to an output file, the operator pushes the switch to turn the LED on and off. The switch needs to be held both on and off for roughly 1/2 second or more during each cycle so that the signals get enough time to settle and cycles do not smear into each other. You want to collect many 10s of samples (switch presses) to get a good idea of the distribution.

There are some tricks to making the measurements. It is easier to take the measurements in a dimly lit or dark room; this allows the baseline signal on the photoresistors to be low, making for a better signal to noise ratio. However, cameras will tend to auto adjust their exposure to compensate, and that can mean slowing down the frame rate to get a longer exposure, resulting in higher latency (which would not be seen under normal light). To compensate, we allowed the camera to adjust the exposure under normal room light (checking the frame rate). We then set the camera to fixed exposure and turned down the lights, resulting in measurements which hopefully correctly represent normal conditions.

5 Camera Latency Measurements

The data were analyzed using Python in Jupyterlab. The turn-on time was measured by simply detecting the signal going above a threshold (we did not analyze the turn-off times). We created overlay plots to visualize the actual curves and spread.

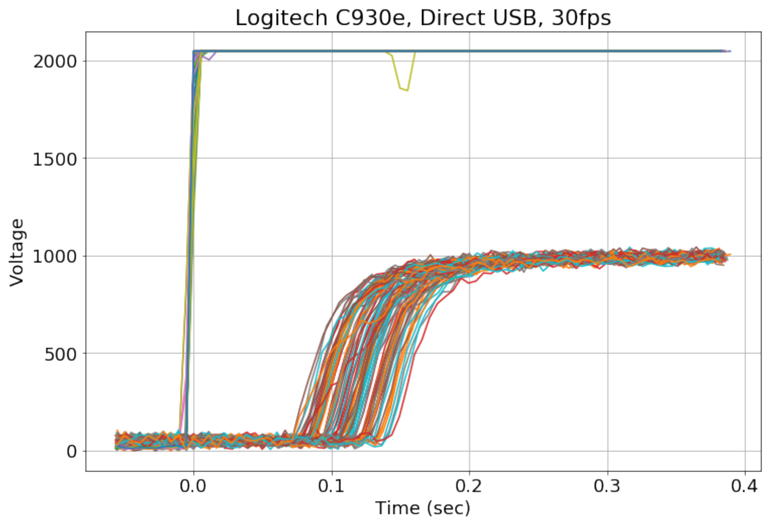

Figure 3: Signal traces for a Logitech C930e at 30FPS

Figure 3 shows the overlayed signals from a Logitech C930e camera at 30 frames/sec (FPS). It was connected directly to a (Linux) laptop via USB. The image stream was generated by a simple OpenCV program reading and displaying the images as fast as possible. You can clearly see the latency of the video stream, with the display turn-on appearing roughly 80-140 msec after the direct trigger. The spread of the traces is generally expected; the 30 FPS camera frequency represents a spread of 33 msec and that will be convolved with other times in the whole system (i.e. USB readout, laptop screen refresh, etc.).

To get values to compare, we need to characterize the distribution, both central value and spread. For the central value, we used the median of the values; this is more robust than the mean against a few outliers. Both the minimum and maximum values of the distribution are of interest. The minimum indicates the "fundamental" latency of the system, that is, the latency when all the elements are at their best. The maximum value indicates the worst-case value. However, using the strictly measured minimum and maximum is very sensitive to the exact dataset. To provide a bit more robust measurement, we used the average of the 3 smallest and 3 largest readings for the lower and upper limits of the range. (Note there is no real justification for this choice; suggestions for a better method are welcome.)

The data from the measurements are shown in Table 1. We measured two cameras, the Logitech C930e and the PlayStation Eye ("PS Eye"). The C930e is our standard camera and has been used for a few years; it has a wide field of view and good quality image. The PS Eye is inexpensive, yet is reputed to have low latency. The cameras were measured at 320x180 pixels (C930e) or 320x240 pixels (PS Eye), since this is a typical image size of FRC.

Camera	Viewer/Server	FPS	Median	Min Value	Max Value
Logitech C930e	OpenCV/Direct USB	30	112	97	129
PS Eye	OpenCV/Direct USB	30	78	50	116
Logitech C930e	Shuffleboard/CSCore	30	125	104	139
Logitech C930e	Shuffleboard/CSCore	15	142	107	189
Logitech C930e	Shuffleboard/CSCore	10	159	112	231
Logitech C930e	Chrome/CSCore	30	163	140	183
Logitech C930e	Chrome/CSCore	15	216	181	316
Logitech C930e	Chrome/CSCore	10	278	194	325
PS Eye	Shuffleboard/CSCore	30	119	93	149
PS Eye	Shuffleboard/CSCore	15	138	97	174
PS Eye	Shuffleboard/CSCore	10	164	107	226

Table 1: Measured data for latency. All latencies are in msec.

Three different "Viewer/Server" setups where used:

OpenCV/Direct USB a simple program using OpenCV to fetch an image directly from the camera (USB) and display it on the screen

Shuffleboard/CSCore WPILib Shuffleboard program version 2019.2.1 against our CSCore-based server

Chrome/CSCore Google Chrome web browser version 71 against our CSCore-based server

The CSCore server is the LigerBots' Python-based implementation (https://github.com/ligerbots/VisionServer) running on a ODROID-XU4. The server was doing no processing on the image, just sending it to be displayed, but it does pass through the OpenCV module. The "Driver's Station" was a Linux laptop connected over WiFi running the viewer (Shuffleboard or Chrome).

6 Comparisons

Looking at Table 1, we see the PS Eye is roughly 30 msec faster than the Logitech C930e when directly connected over USB. This would imply that it is the better camera for driving a robot. However, Figure 4 shows the comparison of the two cameras when used in a setup close to those at FRC competitions ("Shuffleboard/CSCore" in the table). The figure indicates that the PS Eye is only slightly faster than the C930e, both in median and minimum values. It is very unclear why the cameras show similar latencies with CSCore/Shuffleboard when the PS Eye is noticeably faster over USB. It may be that all the extra steps in the chain wash out the speed difference, but that does not fit all the data. It deserves more investigation, but that is left for future work.

Figure 5 show an interesting comparison between using Google Chrome and Shuffleboard as the viewing application. Chrome is consistently slower than Shuffleboard, and gets worse at lower frame rates. If you look closely at the differences, it works out that Chrome is fairly

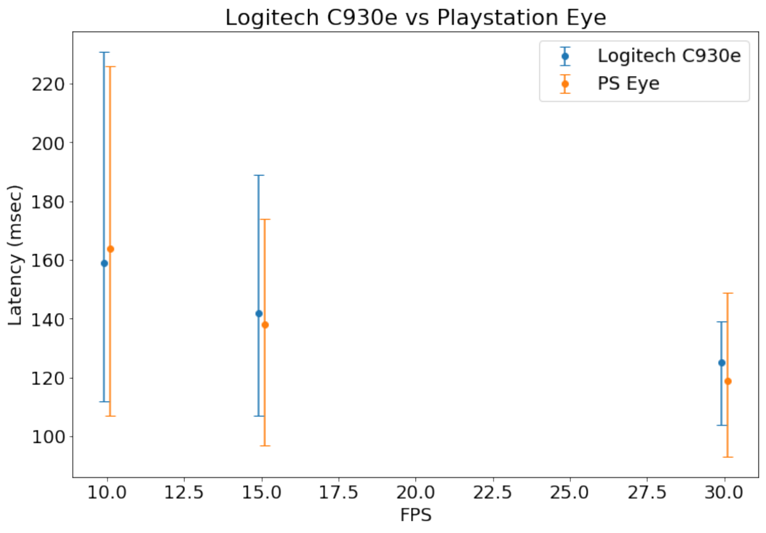

Figure 4: Comparison of Logitech C930e to PlayStation Eye camera at different frame rates.

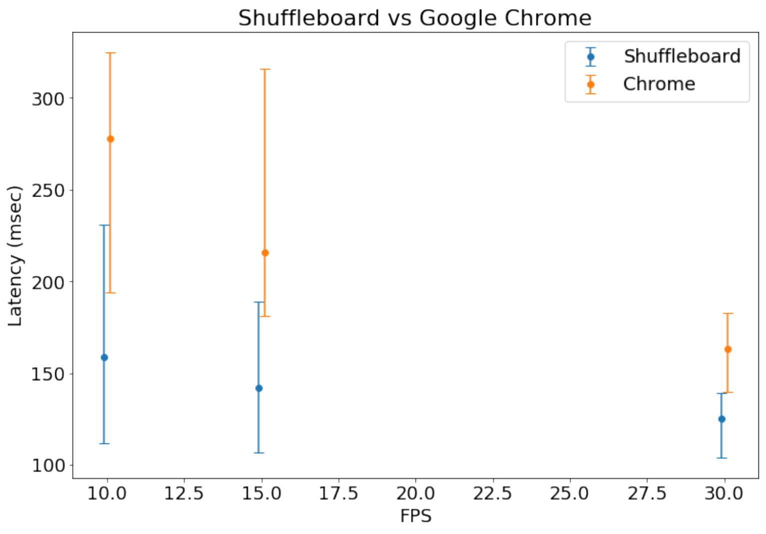

Figure 5: Comparison of viewing camera stream with Shuffleboard versus Google Chrome.

close to 1 frame period slower at each of the frame rates. This suggests that Chrome always waits 1 frame before displaying the video, whereas Shuffleboard is explicitly design to have no buffering. This makes Chrome a poor choice for driving based on the camera feed.

7 Conclusion

We have presented a straightforward method of measuring the latency of a camera and display system. It allows you to measure the distribution of the latency, and accumulate good statistics.

We measured two different camera and 3 different processing/display chains. The PlayStation Eye camera can be faster than the Logitech C930e, but that difference is lost when used in a full FRC "competition" setup. More investigation of this would be useful. However, we did discover that Google Chrome (and presumably other browsers) are unsuitable for live displays during competition, as they add extra latency.