### Motivation:

Video/Imaging and robotics being my major field of interest, I often want/need the embedded platform to stream video from the edge device either continuously or on-demand. In other cases, you just want to keep a record of some events at the edge device, without using up Terabytes of storage. For real-time systems, the frame-rate and latency of the video feed is important too. Sending or storing uncompressed video over most commonly used media (Ethernet, Wireless LAN, etc) doesn't work out well; uncompressed video just requires enormous amounts of bandwidth.

Let's take some examples.

**720p60 (HD) video:**Let's take a simple case of 1280x720 coloured video at 60 frames per second. Bandwith required = 1280(Width) x 720(height) x 3 (colors per pixel) x 8 (bits per pixel) x 60 (frames per second)= 1.33 Gbps!!!

That's a huge amount of bandwidth reqiurement that cannot be satisfied by the common (Wired) LAN connection (1Gb Ethernet). Don't even think of trying to send this over Wifi. Also keep in mind that we have used a simplified case; actual bandwidth consumption will be higher due to overheads (framing signals, etc). Also real bandwidth of most communication channels is much lower than the maximum in most real-world scenarios.

And we haven't even reached Full HD video yet (Spoiler: Colored 1080p60 video will require more than 3Gbps bandwidth. That's 3Giga-bits per seconds!!!)

This is where video compression comes in. It depends on the compression settings you use, but to give a ball-park figures, you can expect at-least an order of magnitude reduction in bandwidth requirement.

For instance, a 1080p30 stream can be streamed at under 12Mbps at a decent quality (H.264-base, high quality). Compare that to 1.5Gbps for uncompressed video, and you get the idea! Similarly, the 720p60 stream (discussed earlier) can be streamed at under 12Mbps too when compressed to H.264. For H.265 compression, these decrease further.

### Gstreamer pipelines for Nvidia Jetson

The video encoding/decoding is a computationally heavy operation, and is best done using dedicated Encoder/Decoder hardware. The Nvidia Jetson devices come with Hardware Encoder & decoders built into the Silicon (known as NVENC and NVDEC respectively), and the Jetpack comes with gstreamer plugins to utilize this functionality in a super easy manner.

In this post, we I will share some basic gstreamer "pipelines" to get you started (and excited) after which you can hopefully explore further. We will not go into too much detail.

I have tried the following using Jetpack 4.4 and Nvidia Jetson NX device, but these should work without change on other Jetson devices too.

**Preparation:**

Let's start with some preparation.

1. Install the latest version of Jetpack on your device.

2. Install the Jetson Stats package, so we can see processor, GPU, memory and NVENC/NVDEC usage while we experiment. Easiest way to install is using the following command in a terminal

sudo -H pip install -U jetson-stats

To run jetson stats, you can type

sudo jtop

3. We will install v4l-utils (video-for-Linux utilities) since I am going to use my old usb-2 webcam as source. To install, simply type the following in a terminal

sudo apt-get install v4l-utils

To query v4l devices (after connecting them to your jetson's usb port, of-course), use the following

v4l2-ctl --list-devices

To query formats supported by your device (video0 in this case)

v4l2-ctl -d /dev/video0 --list-formats-ext

Or, you can simply query for all connected v4l devices

v4l2-ctl --list-formats-ext

### GSTREAMER PIPELINES

In gstreamer, you construct "pipelines" to accomplish your objective. Think of it as a series of blocks you connect starting with your sources (test source, camera etc), any processing blocks in between (format conversion, encoding/decoding etc), ending with your "sink" (screen display, udp streaming address, file etc).

Without going into too much detail, to keep it interesting, we will jump right into trying some simple pipelines, starting with the simplest.

**Play Video Test Source to screen:**

Always a good idea to start with a test source

gst-launch-1.0 videotestsrc ! 'video/x-raw, width=(int)1280, height=(int)720, format=(string)I420, framerate=(fraction)30/1' ! nveglglessink -e

This will display a test pattern on your screen of size 1280x720 at 30fps, as specified in the pipeline parameters.



gstreamer test source

**Display Live usb webcam feed on your screen:**

This is the simplest gstreamer pipeline possible

gst-launch-1.0 v4l2src ! xvimagesink

If you have several cameras attached to your Jetson device, you can specify one as follows

gst-launch-1.0 -ev v4l2src device=/dev/video0 ! xvimagesink

**Encode test pattern video to H264 and save to file on disk:**

gst-launch-1.0 videotestsrc ! 'video/x-raw, format=(string)I420, width=(int)1920, height=(int)1080' ! omxh264enc ! 'video/x-h264, stream-format=(string)byte-stream' ! h264parse ! qtmux ! filesink location=test.mp4 -e

Here we have used a filesink element at the end to specify saving to file, along with file name. Note that this will save to your present working directory. You can see your current working directory by executing the following in the same terminal window

pwd

**Read from file, decode and display on screen:**

Now we can read the file we saved earlier and display the contents on screen

gst-launch-1.0 filesrc location=test.mp4 ! qtdemux name=demux ! h264parse ! omxh264dec ! nveglglessink -e

**Encode live video from webcam, decode and display:**

It might look useless, but this can be used to see how much delay is being added to the video just by the encoding & decoding process, without any bottlenecks due to network, etc.

gst-launch-1.0 v4l2src ! decodebin ! videoconvert ! omxh264enc ! h264parse ! omxh264dec ! nveglglessink -e

**Encode webcam video to H.264 and stream using rtp (real time protocol):**

Here's a simple pipeline to accomplish this

gst-launch-1.0 v4l2src ! decodebin ! videoconvert ! omxh264enc ! video/x-h264, stream-format=byte-stream ! rtph264pay ! udpsink host=127.0.0.1 port=5000

And to receive, decode and display this stream, we can use the following (in a separate terminal)

gst-launch-1.0 udpsrc port=5000 ! application/x-rtp, encoding-name=H264, payload=96 ! rtph264depay ! h264parse ! omxh264dec ! nveglglessink

We are streaming to "localhost" (own own machine) by using host=127.0.0.1. To stream to a different host, you can replace with IP address of that PC.

**Encode multiple streams from webcam:**

Did you know that the Jetson Video Encoder and Decoder can encode and decode up to 4k video streams simultaneously without consuming all your CPU power. To demonstrate and make use of this, we will construct a gstreamer pipeline to do the following:

* Use our usb camera as source
* Make 3 copies of our camera video stream using "tee" element
* Display the first stream as is (i.e, before any compression)
* Encode the 2nd copy of stream using H.264 and stream to port 5000
* Encode the 3rd copy of stream using H.265 and stream to port 5001

gst-launch-1.0 v4l2src ! decodebin ! videoconvert ! tee name=t \  
t. ! queue ! xvimagesink \  
t. ! queue ! omxh264enc ! video/x-h264, stream-format=byte-stream ! rtph264pay ! udpsink host=127.0.0.1 port=5000 \  
t. ! queue ! omxh265enc ! video/x-h265, stream-format=byte-stream ! rtph265pay ! udpsink host=127.0.0.1 port=5001

Note that while we're casting to the same device ([localhost](http://localhost/)/ 127.0.0.1) in this case, you can replace it with the IP of a different computer/jetson device on the network too.

Now, in another terminal (or on another device, if you used an IP different than 127.0.0.1), we can receive, decode and display the H.264 stream using the following

gst-launch-1.0 udpsrc port=5000 ! application/x-rtp, encoding-name=H264, payload=96 ! rtph264depay ! h264parse ! omxh264dec ! nveglglessink

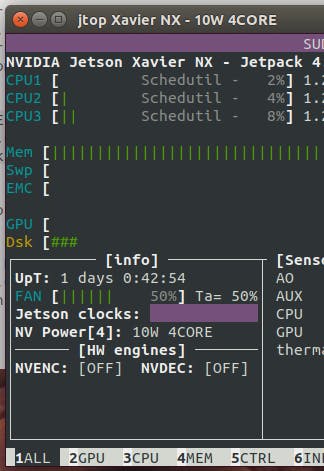
Similarly, we can do the same for the H.265 stream using the following pipeline

gst-launch-1.0 udpsrc port=5001 ! application/x-rtp, encoding-name=H265, payload=96 ! rtph265depay ! h265parse ! omxh265dec ! nveglglessink

To see a full matrix of Encoder and Decoder capabilities on the Jetson devices, see this blog post

<https://www.stereolabs.com/blog/h-264-h-265-video-encoding-support-matrix-for-nvidia-jetson/>

To ensure/check if the hardware encoder/decoder is getting used, look at the "ALL" page in the jtop window (mentioned in the "Preparation" section above), as in the screenshots below



1 / 2 • NVENC and NVDEC are OFF. Encoder/Decoder hardware not in use

### ADDITIONAL INFO/ HINTS..

1. You can get additional information on your gstreamer pipeline execution using the verbose mode. Simply add -v to your pipeline to display additional info. For example, notice the -v argument in the following pipeline.

gst-launch-1.0 v4l2src ! xvimagesink -v

This is also useful to get the "caps" information on your executing pipeline, if needed by the receiving pipeline, or for debug purposes.

2. To dig further into gstreamer pipelines, read up more here

<https://docs.nvidia.com/jetson/l4t/index.html#page/Tegra%20Linux%20Driver%20Package%20Development%20Guide/accelerated_gstreamer.html>