FlightView Installing Manual
Last updated: 2024-03-15 EHL

Installing FlightView

Part 1: Prerequisites:

OS:

FlightView has been tested on recent versions of Ubuntu Linux and Linux Mint. Most development has taken place using Linux Mint 20.2 with kernel 5.4.0-74-generic.

GPU:

The computer must be using the proprietary Nvidia graphics drivers as well as a matching cuda driver. Additionally, a CUDA development kit must be installed which includes the nvcc nvidia cuda compiler and associated libraries. All these can be installed by using a single download.

Nvidia CUDA linux how-to:
https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html

Be sure to remove any prior version(s) of CUDA and/or the Nvidia drivers prior to installing.

Install your kernel headers and build-essential files. You may need to install the kernel headers again if you update your OS:

```
sudo apt-get install build-essential
sudo apt-get install libc-dev libc6-dev gcc g++
sudo apt-get install linux-headers-$(uname -r)
```

It is strongly recommended to install the ssh server so that if the graphics do not come up, an ssh connection can be made for diagnostics:

```
sudo apt-get install openssh-server
```

The Nvidia CUDA download will include drivers for the nvidia card, cuda support files, and the nvcc.

It is recommended to use the ".run" shell script for the installation, and to not use the RPM or DEB options.

To run the ".run" installer file, download it to a known location that is not in /tmp. Next, log out from the GUI desktop. Press CONTROL-ALT-F2 to get to a text console, or ssh in from a local machine. Issue the following command to stop the desktop process (this is needed in order to run the installer):

```
sudo systemctl stop lightdm # mint
sudo systemctl stop gdm3 # ubuntu
```

Now add execute permission to the runfile, like this (modify for your exact filename):
`chmod 755 cuda_11.6.0_510.39.01_linux.run`

Now run the file, as root:
`sudo ./cuda_11.6.0_510.39.01_linux.run`

If you run into errors, there are two log files to examine:
`/var/log/cuda-installer.log`
and
`/var/log/nvidia-installer.log`

Be sure to keep the installer logs, and follow the directions carefully to include the library location with your system. On this particular linux workstation, the file `/etc/ld.so.conf.d/cuda-11-6.conf` was created with the following contents:

```
/usr/local/cuda-11.6/targets/x86_64-linux/lib
```

Do not copy this text directly; adjust it for your system. Be sure to run `sudo ldconfig` after updating the file.

Make sure to blacklist the open source nvidia graphics driver prior to rebooting:

```
sudo bash -c "echo blacklist nouveau > /etc/modprobe.d/blacklist-nvidia-nouveau.conf"
sudo update-initramfs -u
sudo update-grub
```

You may want to take the opportunity to blacklist the wifi or other unneeded hardware while you are editing these files.

Camera Link drivers:

Even when not planning to use camera link, the driver header files are still required for building. Only versions prior to 6.0 are supported.

https://edt.com/file-category/pdv/
(download the .run file and follow the directions. The installer must be run as root, and the default location, /opt/EDTpdv is recommended)

Typically, the installer will attempt to build all the EDTpdv utilities. Some of these may fail, but generally the most important ones will build without too much trouble.

Recent EDT drivers need these additional steps:
```
sudo chmod -R o+r /opt/EDTpdv
sudo mv /opt/EDTpdv/version /opt/EDTpdv/version-orig
```

The drivers include a kernel module, which can be manually re-built with the following commands:

```
cd /opt/EDTpdv/module
sudo make
```

(make includes install)

```
sudo update-initramfs -u
sudo update-grub
```

Rebuilding this module after OS updates is generally required as the kernel will update without bringing in the EDTpdv drivers. Always make sure you have the correct linux-headers package installed (see previous steps for the nvidia driver).

To initialize the camera link port, the `initcam` program is used with a camera link config file. By default, a large set of camera link config files are provided in `/opt/EDTpdv/camera_config`. These plaintext files may be used as a starting point for building a custom config file.

Here is an example for a 640x481 14-bit camera:

```
camera_class:                   "Imaging Spectrometer"
camera_model:                   "Camera Link 14-bit"
camera_info:                    "640x481 HSI"
width:                          640
height:                         481
depth:                          14
extdepth:                       14
CL_CFG_NORM:                    02
continuous:1
MODE_CNTL_NORM: 00
```

Here is an example for a 1280x328 16-bit camera:

```
camera_class:                   "Imaging Spectrometer"
camera_model:                   "FPA"
camera_info:                    "1280x480 16-bit HSI"
width:                          1280
height:                         328
depth:                          16
extdepth:                       16
rbtfile: aiagcl.bit
continuous: 1
CL_CFG_NORM:                    00
htaps: 4
serial_baud: 19200
```

A Camera Link port can be initialized like this:

```
/opt/EDTpdv/initcam -f /opt/EDTpdv/camera_config/spectrometer.cfg
```

It is recommended to build a shell script for this task and then to call it from a desktop launcher, with the terminal opening to show any error messages.

Example desktop file: init_camera.desktop:

```
[Desktop Entry]
Name=initcam NGIS
Exec=/home/eliggett/bin/init_ngis.sh
Comment=640x481
Terminal=true
Icon=cinnamon-panel-launcher
Type=Application
```

Example shell script file: init_ngis.sh:

```
#!/bin/bash
CFGFILE=/opt/EDTpdv/camera_config/NGIS.cfg
echo "Running $0 for config file $CFGFILE ..."
/opt/EDTpdv/initcam -f $CFGFILE
echo "Done! Closing window in 2 seconds."
sleep 2;
```

To verify the port was initialized, run the `/opt/EDTpdv/take` command, which attempts to acquire one frame from the camera link port. The command will typically return one of the following:

- `EDT /dev/pdv0 open failed. pdv_open(pdv0_0): No such device or address`
  - This means that either the kernel module did not load, or the card has a problem or isn't physically installed. Check the output of `lsmod` and `dmesg` for details.
- `pdv0: Invalid image size. Make sure device has been initialized.`
  - (Run `initcam`)
- `1 images 1 timeouts 0 overruns 0 bytes`
  - The port was initialized, but the camera was not running.
- `1 images 0 timeouts 0 overruns`
  - The port was initialized, and the camera returned a frame without timeout. This means the camera link system is ready for use.

The return value is generally "1" upon error and "0" upon success.

Part 2: Building and Installing FlightView

Libraries needed to build FlightView:

It is required to install the following packages before building FlightView:

```
sudo apt-get install build-essential
sudo apt-get install libc-dev libc6-dev gcc g++
sudo apt-get install linux-headers-$(uname -r)
sudo apt-get install qt5-default qt5-qmake qt5-qmake-bin
sudo apt-get install git
```

```
sudo apt-get install libboost-all-dev
sudo apt-get install libqt5svg5 libqt5svg5-dev
sudo apt-get install qtcreator
sudo apt-get install libgsl-dev libgsl23 libgslcblas0 # adjust names
as needed.
```

FlightView also requires libgst and specific libgst plugins.

```
sudo apt-get install libgstreamer1.0-0   libgstreamer1.0-dev
sudo apt-get install gstreamer-1.0-plugins-base  gstreamer-1.0-
plugins-good gstreamer-1.0-plugins-good-dev
```

RTP support is provided via two libraries: gstreamer, and rtpnextgen.

gstreamer:

- Supports IPV4 and IPV6
- Requires patch for 16-bit grayscale (see below)

rtpnextgen:

- Supports IPV4
- Supports faster frame rates (>3 gigabit/sec)
- Less picky about timestamps and other data
- Native 16-bit grayscale

The following procedure is used to patch gstreamer. If you're not using gstreamer, you may install gstreamer using the apt repository, and skip the custom gstreamer patch as detailed below.

For work with raw 16-bit grayscale RTP sources, you will need to patch the gstreamer RTP plugin using the following procedure:

First prepare the apt system for source installs:
Enable apt-src repositories, typically by editing `/etc/apt/sources.list` and uncommenting the first few apt-src lines.

Update the apt cache:

```
sudo apt-get update
```

Install apt-src:

```
sudo apt-get install apt-src
```

Install the source for gstreamer1.0-plugins-good:

```
mkdir ~/Downloads/gst
cd ~/Downloads/gst
apt-src install gstreamer1.0-plugins-good
```

Patch the library:

```
cd gst-plugins-good1.0-16.3/gst/rtp
cp gstrtpvrawdepay.c  gstrtpvrawdepay.c-orig
```

For this next step, you will need a copy of the patch file which is within the LiveViewLegacy repository. Download the patch file separately or clone the repo ahead of time:

https://github.com/nasa-jpl/LiveViewLegacy/tree/flight/utils/gst

```
cp ~/Documents/FlightView/LiveViewLegacy/utils/gst/gstrtpvrawdepay.c.patch .
patch gstrtpvrawdepay.c gstrtpvrawdepay.c.patch
```

Examine the patched file, as some versions do not patch correctly.

Now build the plugin:

```
cd ~/Downloads/gst
apt-src build gstreamer1.0-plugins-good
```

Install the modified plugin by installing each of the resulting deb files (adjust names to match):

```
sudo dpkg --install gstreamer1.0-plugins-good-doc_1.14.5-
0ubuntu1~18.04.3_all.deb gstreamer1.0-pulseaudio_1.14.5-
0ubuntu1~18.04.3_amd64.deb gstreamer1.0-qt5_1.14.5-
0ubuntu1~18.04.3_amd64.deb gstreamer1.0-gtk3_1.14.5-
0ubuntu1~18.04.3_amd64.deb gstreamer1.0-plugins-good_1.14.5-
0ubuntu1~18.04.3_amd64.deb gstreamer1.0-plugins-good-dbg_1.14.5-
0ubuntu1~18.04.3_amd64.deb libgstreamer-plugins-good1.0-0_1.14.5-
0ubuntu1~18.04.3_amd64.deb libgstreamer-plugins-good1.0-dev_1.14.5-
0ubuntu1~18.04.3_amd64.deb
```

Download, configure, and build FlightView:

To download, use the following steps:

```
mkdir ~/Documents/FlightView
cd ~/Documents/FlightView
git clone https://github.com/nasa-jpl/LiveViewLegacy.git -b
flightview
```

Now update the submodules (used for the Atlans A7 binary GPS library):

```
cd LiveViewLegacy
git submodule init
git submodule update
```

Compiling the back-end (cuda-take):

If your version of nvcc (`nvcc -V`) is prior to version 11.6:

- Edit the `Makefile (cuda_take/Makefile)`, and uncomment and edit an NVCCFLAGS line which matches your card. You can quyery the card type using the command "`nvidia-smi -L`"
- Visit this URL and find the correct specification for your card:
  - https://arnon.dk/matching-sm-architectures-arch-and-gencode-for-various-nvidia-cards/

- Look for the correct “compute” and “sm” codes. If the cuda version is greater than 11.6, these codes may be removed and “arch=native” may be substituted (this is the default NVCCFLAGS line).
  - `NVCCFLAGS  = -gencode arch=compute_86,code=sm_86 -lineinfo -Xcompiler -rdynamic --compiler-options '-Wall -Werror -Wno-unused-function'`

Verify that the version of `nvcc` on the `$PATH` is the correct version (manual nvidia driver installs typically reside in /usr/local, apt installs in /usr):

```
which nvcc
nvcc -V
nvidia-smi
```

If `nvcc` isn’t within the current path, be sure to correct this before proceeding.

To compile the nvidia CUDA back-end, issue the following command from within the `cuda_take` directory:

```
cd cuda_take
```

(current directory is `~/Documents/FlightView/LiveViewLegacy/cuda_take`)

```
make -j
```

If you receive errors about “unknown architecture” then the Makefile will need to be adjusted to specify the correct compute and sm codes. If the error persists, it may mean that your card is newer than the driver supports and you will need to find a beta version of the latest Nvidia cuda drivers.

If you receive this error:
`/bin/sh: 1: nvcc: not found`
Then you may need to add the nvidia cuda bin directory to your current path like this (verify this location on your system before proceeding):
`export PATH=$PATH:/usr/local/cuda/bin`
(On many systems, the specific cuda folder, such as “cuda-11.4” will be symlinked to “cuda” for convenience.)

Next, create a directory for building FlightView:

```
cd ../..
(location is now ~/Documents/FlightView)
mkdir build
cd build
(location is now ~/Documents/FlightView/build)
```

Run `qmake` to configure the Makefile:
For a release build (for deployment):
`qmake ../LiveViewLegacy/liveview.pro`

For a debug build (for testing):
```
qmake CONFIG+=debug ../LiveViewLegacy/liveview.pro
```

Build FlightView:
```
make -j
```

When FlightView builds, it will be located in a folder within the `build` directory called "`lv_release`", which also contains an icon and an example plaintext desktop launcher (which will likely need to be edited slightly).

It is recommended to install the files as follows:
```
liveview: /usr/local/bin/liveview
liveview_icon.png: /usr/share/pixmaps/liveview.png
liveview.desktop: /usr/share/applications/liveview.desktop
```