[DRAFT] How to Achieve 30 fps with BeagleBone Black, OpenCV, and Logitech C920 Webcam [DRAFT]

Michael Darling
FndrPlayer39@gmail.com

September 24, 2013

This "How-to" outlines some of the issues associated with video capture on the BeagleBone Black (BBB) for robotic vision applications using a USB webcam and OpenCV, and presents a possible solution for improved video capture performance along with a set of detailed instructions. In particular, this document addresses the challenge of achieving a suitably high framerate (30 fps) for robotic vision applications making use of the BBB. If you wish, you can skip the introductory material and jump directly to the How-to.

Contents

Problem Background

Many robotic vision applications require video capture and frame processing to occur at a high rate of speed. For example, my master's thesis seeks to implement autonomous close formation flight of two remote controlled aircraft using computer vision for localization. In short, the lead aircraft will be outfitted with very high intensity LEDs on each of its wingtips and tail surfaces while the following aircraft will be equipped with a vision system consisting of the BeagleBone Black and a USB webcam. Both vehicles will be controlled using the open source ArduPilot Mega autopilot. The OpenCV vision software running on the BBB will detect the LEDs in each video frame, estimate the 3-D relative position and orientation of the leader, and pass this information over to the autopilot which will handle the control. This type of application requires high bandwidth localization estimates for the follower to maintain its position behind the leader. The rate at which video frames can be processed and captured becomes even more important considering the fact that the LEDs may not be detected in every frame due to background noise in the image.

First Attempts with the PlayStation 3 Eye

For me, a reasonable place to start was by using the PlayStation 3 Eye USB webcam in combination with OpenCV. The PS3 Eye is used by many "do-it-yourself" robotics enthusiasts due to its availability, very low cost, high framerate (up to 120 fps at 320 × 240 and up to 60 fps at 640 × 480 resolution), and multi-platform support by 3rd party drivers. [1] Unfortunately for those who want to use the PS3 Eye with the BBB, this kind of performance can't be expected—at least not easily.

If you were to set up a simple OpenCV video capture program and attempted to operate the PS3 eye at 640 × 480 resolution at 60 fps, you would end up with repeated "select timeout" errors and no video frames. You would have identical results at 30 and 15 fps as well, however if you settle for 320 × 240 resolution, you would get a stream of video, but always at 30 fps regardless of the framerate you set. *Why?* It turns out that the OpenCV functions for setting the video framerate do not work (at least for Video4Linux devices) and the default 30 fps is used. In order to set the camera framerate, you have to write your own capture code using the Video4Linux2 API. [2] But even after using custom capture code, you would find that you can only acquire 640 × 480 frames with the camera set to 15 fps. And even then you would *actually* be capturing frames at about 7 fps at best.

After more days... weeks... months than I'm willing to admit, I finally came to find that issue lies with the way the PS3 Eye transfers the frames over USB. The PS3 Eye captures video in the uncompressed YUYV pixel format and transfers the frames in bulk mode, which guarantees data transmission but has no guarantee of latency. In the case of the BBB, the large amount of data being sent by the webcam saturates the bulk allotment on the BBB's USB bandwidth and select timeout errors result. [3]

Shift to the Logitech C920 and MJPEG Format

In order to reduce the USB bandwidth required by the webcam, a compressed pixel format such as MJPEG or H.264 can be used. The PS3 Eye does not support video compression, so I looked to the Logitech C920 USB webcam instead. H.264 compression comes at a cost however, and will set you back about $72 to purchase a C920 on Amazon. (Other cameras, such as the Logitech C270 also support the MJPEG pixel format, which I ended up using over H.264. However, using the Logitech C270 will require a little extra work. The C270 does not include the Huffman table as part of the MJPEG stream and may need to be added for the video frames to be correctly read by OpenCV and other programs. See [3] for more.)

Since the C920 transfers compressed images in *isochronos* mode, it can easily deliver 640 × 480 frames at 30 fps using very little CPU. [3] If you save the video frames to individual JPEG image files, you can easily transfer them to your desktop computer and view them in any image viewer. If we want to use the MJPEG stream in a vision application written with OpenCV though, these images will have to be decompressed in real-time and converted to a cv::Mat object so that OpenCV can work with the image.

JPEG Decompression with OpenCV

Luckily, OpenCV includes functions for decoding images from a buffer, specifically the `cvDecodeImage()` and `imdecode()` functions, depending on if you are working in C or C++. [4] [5] The primary reason for using MJPEG over the H.264 compression format is that MJPEG uses *intra*frame compression whereas H.264 uses *inter*frame compression. Put simply, each MJPEG frame gets compressed individually as a JPEG image and each compressed frame is independent of all others. H.264 on the other hand, uses interframe prediction to "take advantage from temporal redundancy between neighboring frames to achieve higher compression rates". [6] While this is good for compressing video streams meant to be viewed as a continuous stream, it is not well-suited for embedded vision applications since H.264 decompression is CPU intensive and can exhibit decompression artifacts and lag when there is a lot of motion in the video.

If you installed OpenCV on your BBB with a package manager such as Ubuntu's Advanced Packaging Tool (`apt-get`) or Ångstrom's `opkg`, odds are that you will still only see about 10-20 fps when you try and capture at 640 × 480 resolution at the 30 fps setting. And if you profile your code by including calls to `time()` from the `<ctime>` header file, you will see that most of the time spent by your program is dedicated to decoding the image and converting it to the cv::Mat object. Moreover, the decompression eats up nearly all of the CPU. Luckily, there are some steps that you can take to significantly reduce decompression time and CPU usage—leaving more time and resources for your vision program to process the frames.

How-To: Achieve 30 fps

Disclaimer

Please keep in mind that I am *not* an expert in embedded Linux, OpenCV, or C/C++ programming. I am a graduate student studying aerospace engineering. I have only been working with embedded hardware, Linux, OpenCV, and C/C++ for about a year. My thesis has taken me on a detour into investigating ways to improve the framerate when using a USB webcam with the BeagleBone Black. This "How-To" is essentially a compilation of other resources and an outline of the

steps that I used to solve this particular problem—your mileage may vary. As always, it is your responsibility to understand the commands you are invoking. I have spent a lot of time on this problem and have relied heavily on the help of the open source community. I have put this guide together as a way of giving back to the open source community, and hope that some can find it useful. If you have any comments or suggestions for improving this "How-To" please email me at the address provided at the top of this page.

Prerequisites

This process can be followed with some slight variations to your setup. For reference, here is a list of what hardware and software I used.

- BeagleBone Black, Rev. A5
- BBB running Ubuntu 13.04 eMMC "flasher" image provided by [7]
- Logitech C920 USB webcam
- x86 PC running Ubuntu 13.04
- LAN network
- USB thumb drive

Before attempting any of these steps, you should already be familiar with the basics of using `ssh` to connect to the BBB over USB or a LAN network. You should also be comfortable working from the Linux command line and have some experience with GNU compilers, cmake, and the "configure, make, make install" build processes.

Objective

The main objective of this How-to is to take advantage of NEON hardware acceleration available on the BBB. [8] The details of how NEON acceleration works are a bit over my head, but its usefulness is obvious: "NEON technology can accelerate multimedia and signal processing algorithms such as video encode/decode, 2D/3D graphics, gaming, audio and speech processing, image processing, telephony, and sound synthesis by at least 3x the performance of ARMv5 and at least 2x the performance of ARMv6 SIMD." You can also see the clear benefits here. [16]

In order to use NEON, we will (1) build and install a more optimized JPEG codec called "libjpeg-turbo", and (2) rebuild OpenCV with NEON enabled. The latter is a bit tricky due to the limited processing power and storage capacity of the BBB. To speed up the build, I will introduce an easy way to cross-compile large projects for the BBB.

1. Install libjpeg-turbo

libjpeg-turbo is a highly-optimized version of the libjpeg JPEG codec library that is designed to take advantage of NEON acceleration. According to the libjpeg-turbo project page, the library is capable of encoding/decoding JPEG images 2–4 times faster than the standard libjpeg library. [9]

On the BBB, download the libjpeg-turbo source tarball and then extract it.

```
wget http://sourceforge.net/projects/libjpeg-turbo/files/1.3.0/libjpeg-turbo-1.3.0.tar.gz
tar xzvf libjpeg-turbo-1.3.0.tar.gz
```

Enter the source directory, then create a build directory and enter it.

```
cd libjpeg-turbo-1.3.0
mkdir build
cd build
```

By default libjpeg-turbo will install into `/opt/libjpeg-turbo`. You may install to a different directory by passing the `--prefix` option to the configure script. However, the remainder of these instructions will assume that libjpeg-turbo was installed in its default location.

```
../configure CPPFLAGS='-O3 -pipe -fPIC -mfpu=neon -mfloat-abi=hard'
make
sudo make install
```

Note that the `-O3`, `-fPIC`, and `-mfpu=neon` are particularly important as they enable code optimization, position-independent code generation, and NEON hardware acceleration, respectively.

2. Setting up for Distributed Cross-Compilation (Optional, but Recommended)

Since OpenCV is such a large project and the BBB has limited processing power, it is much more convenient to set up cross-compilation. Typically, setting up a cross-compilation environment can be a tedious process and is especially cumbersome when building a large project that has many dependencies which, in turn, depend on other dependencies, etc. . . etc.

Fortunately with *distributed* cross compiling, you can take advantage of the libraries already installed on the BBB while still using your (probably x86) PC to cross-compile the object files much faster than if they were compiled locally. With distributed cross-compilation you can execute the build from the BBB just as if you were building on the BBB, itself. Here is how you can set up a distributed cross-compiler with `distcc`: [10]

On your PC, download the 32-bit version of Linaro GCC and the associated libraries. I found the appropriate cross-compiler from [11].

```
sudo apt-get install ia32-libs
wget https://launchpad.net/linaro-toolchain-binaries/trunk/2013.08/+download/gcc-linaro-arm-linux-gnueabihf-4.8-2013.08_linux.tar.xz
```

Extract the files and set the cross-compiler (`CC`) to the one you just installed

```
tar xzvf gcc-linaro-arm-linux-gnueabihf-4.8-2013.08_linux.tar.xz
export CC=‘pwd‘/gcc-linaro-arm-linux-gnueabihf-4.8-2013.08_linux/bin/arm-linux-gnueabihf-
```

Confirm that the correct cross-compiler is active.

```
${CC}gcc --version

>> arm-linux-gnueabihf-gcc (crosstool-NG linaro-1.13.1-4.8-2013.08 - Linaro GCC 2013.08) 4.8.2 20130805 (prerelease)
>> Copyright (C) 2013 Free Software Foundation, Inc.
>> This is free software; see the source for copying conditions.  There is NO
>> warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Next, we will test that the compiler works by cross-compiling a simple test program for the BBB. Create a file called `hello_world.c` and paste into it the following:

```
int main(void)
{
  printf("Hello, cross-compilation world !\n");
  return 0;
}
```

Compile the program with:

```
${CC}gcc hello_world.c -o hello_world
```

Try running the program on your PC and confirm that it does *not* run.

```
./hello_world
```

You should be returned an error similar to the following:

```
bash:  ./hello_world:  cannot execute binary file
```

Now copy the executable to the BBB using `scp`. Your command should be similar to:

```
scp hello_world ubuntu@192.168.7.2:~/hello_world
```

Then execute the program on the BBB. You should see the “Hello, cross-compilation world !” message written to stdout.

```
cd ~
./hello_world
```

```
>> Hello, cross-compilation world !
```

If the above example of cross-compilation worked, you can now move on to setting up for *distributed* cross-compilation. We will begin by installing `distcc` on *both* the PC and the BBB. According to [11], we should build the most recent version of `distcc` from source to take advantage of a few features.

On **both** the PC and the BBB, begin by installing some prerequisite packages.

```
sudo apt-get install subversion autoconf automake python python-dev binutils-dev
libgtk2.0-dev
```

Again, on both machines, download the distcc source, and install it

```
svn checkout http://distcc.googlecode.com/svn/trunk/distcc-read-only
cd distcc-read-only
./autogen.sh
./configure --with-gtk --disable-Werror
make
sudo make install
```

Now that `distcc` is installed, we need to create some symbolic links on the BBB. Be careful to use the correct paths here or else you may overwrite one or more of your GNU compilers.

First, check which `gcc` and `distcc` are being called by default

```
which gcc
>> /usr/bin/gcc
which distcc
>> /usr/local/bin/distcc
```

Create the symlinks.

```
sudo ln -s /usr/local/bin/distcc /usr/local/bin/gcc
sudo ln -s /usr/local/bin/distcc /usr/local/bin/g++
sudo ln -s /usr/local/bin/distcc /usr/local/bin/c++
sudo ln -s /usr/local/bin/distcc /usr/local/bin/cpp
```

Now check your `PATH` variable to see if `/usr/local/bin` is included *before* `/usr/bin`. If it is not, then prepend `/usr/local/bin` to your `PATH`. For example:

```
echo $PATH
>> /usr/sbin:/usr/bin:/sbin:/bin
export PATH=/usr/local/bin:$PATH
```

So now you can check `distcc` is being called correctly (through the symlink you just created) by checking the following:

```
which gcc
/usr/local/bin/gcc
```

If this doesn't check out, then go back and make sure that you have all the symbolic links correct and prepended `/usr/local/bin` to your `PATH` variable. It is important that you add to the *beginning* of the path since gcc/cc/g++/c++ get called in the order they appear on the path.

Now we will create some environment variables for `distcc` in order to control some settings. I will just introduce the commands here, but if you want to read more, refer to [11] and [12]. You will need to have a working network (Either over LAN or USB through which the BBB and your PC can communicate.) The first environment variable sets the IP address of your PC that will be doing the compilation followed by a forward slash and the "number of jobs per machine". A good rule of thumb is to use twice your number of processor cores. So for me, I would use:

```
export DISTCC_HOSTS="192.168.2.3/4"
```

For the rest of the environment variables use the following:

```
export DISTCC_BACKOFF_PERIOD=0
export DISTCC_IO_TIMEOUT=3000
export DISTCC_SKIP_LOCAL_RETRY=1
```

In my case, I found that I had to execute the following command on the BBB before trying to build OpenCV so that the cmake build process would use `gcc` rather than `cc`, which I did not have installed:

```
export CC=/usr/local/bin/gcc
```

Now, we have to move back to the *PC* and create some similar symlinks.

```
cd gcc-linaro-arm-linux-gnueabihf-4.8-2013.08_linux/bin
ln -s arm-linux-gnueabihf-gcc gcc
ln -s arm-linux-gnueabihf-cc cc
ln -s arm-linux-gnueabihf-g++ g++
ln -s arm-linux-gnueabihf-c++ c++
ln -s arm-linux-gnueabihf-cpp cpp
```

You will have to prepend to your path as well to make the cross-compiler active.

```
export PATH=$HOME/gcc-linaro-arm-linux-gnueabihf-4.8-2013.08_linux/bin:$PATH
which gcc
>> /home/uname/gcc-linaro-arm-linux-gnueabihf-4.8-2013.08_linux/bin/gcc
```

When you are about ready to start compiling OpenCV, you can launch the `distcc` daemon with the following command (Your command may be different depending on the number of jobs, and your BBB's IP address):

```
distccd --daemon --jobs 4 --allow 192.168.2.9 --verbose --log-stderr --no-detach
```

At this point, you could begin building/compiling any program or project on the BBB and you should see some activity on your PC as it receives the jobs from the BBB and compiles the object files. In this case, you probably want to build OpenCV—for which you should go through the next section. But once you are done building, you will most likely want to be able to disable distcc so that you can compile programs natively on your BBB and PC again. Here is how you can disable `distcc`.

On the BBB, disable all of the symlinks you created.

```
sudo rm /usr/local/bin/{gcc, g++, cpp, c++}
```

The easiest way to restore all of your environment variables is simply to restart the BBB.

```
sudo reboot
```

Now you will want to confirm that gcc is called instead of distcc. You can use the following commands. You may also want to try compiling a small hello_world.c example.

```
which gcc
>> /usr/bin/gcc
```

If you end up with errors that the gcc compiler doesn't exist, then you can remove then reinstall the compilers with:

```
sudo apt-get remove --purge build-essential
sudo apt-get install build-essential
```

Then we will do similarly on the PC. Make sure you are in the directory that you installed the cross-compiler: `gcc-linaro-arm-linux-gnueabihf-4.8-2013.08_linux/bin`

```
sudo rm gcc cc cpp g++ c++
```

Then reboot your computer to restore the `PATH`. Again, test to confirm that you can compile programs locally on your PC.

3. Building OpenCV with libjpeg-turbo and NEON

Now we will build OpenCV with libjpeg-turbo as the JPEG codec and with NEON hardware acceleration enabled. You do not have to use distributed cross-compiling, as described above, but it will dramatically reduce the build time. Due to the limited storage on the BBB, you will probably need some kind of external storage. I actually used a 16 GB μSD card and a USB card reader, but a regular USB thumb drive will probably work fine.

The first thing we have to do is reformat the USB drive with an ext2 partition. You might be able to use another kind of filesystem, but it needs to support symbolic links—which ext2 does. Depending on your OS this process will be different, but if your are running Ubuntu insert the USB drive and start up GParted. [13] (If GParted is not installed, you can install it with `sudo apt-get install gparted`.) In the top right corner select your USB drive. *Please be **certain** that you have chosen the correct device, or else you could end up erasing data on your PC's hard drive.* Go to Device > Create Partition Table and accept the prompt. Then to create the ext2 partition, right click on the "unallocated space" partition and in the File System drop-down menu, select ext2 and "Add" the partition. When complete, eject the USB drive and remove it.

From the BBB's terminal we need to mount the device. To know which drive is the USB drive, use the following command, insert the USB drive, and repeat the command. The new drive is the thumb drive. I will assume this `/dev/sda` with one partition, `/dev/sda1` as it was for me.

```
ls /dev/sd*
```

I found that it was easiest to do the following commands as root. (Be careful!)

```
sudo su
```

Create a mount point and mount the filesystem.

```
mkdir -p /mnt/ext2
mount -t ext2 /dev/sda1 /mnt/ext2
```

Navigate into the directory and download the OpenCV source code, then extract the files. [14]

```
cd /mnt/ext2
wget git clone https://github.com/Itseez/opencv.git
```

Enter the directory that is created—whatever it is called. Then create a build directory and enter that.

```
cd OpenCV
mkdir release
cd release
```

Now we will run cmake with a bunch of flags that should enable libjpeg-turbo and NEON. note that some of these flags, such as `USE_VFPV3=ON` and `USE_NEON=ON` may have no effect, as they only work when cross-compiling without `distcc` [15]. That is okay—the `-mfpu=neon` flag will enable NEON for us. I've just gone ahead and included all of the flags that I used anyways. If you are using `distcc` to cross-compile, make sure that you see some activity in your PC's terminal after you execute the cmake command. (The PC should compile a few programs while cmake tests the compilers it has available.)

```
cmake -D CMAKE_C_FLAGS='-O3 -mfpu=neon -mfloat-abi=hard' -D
CMAKE_CXX_FLAGS='-O3 -mfpu=neon -mfloat-abi=hard' -D CMAKE_BUILD_TYPE=RELEASE
-D CMAKE_INSTALL_PREFIX=/usr/local -D BUILD_PYTHON_SUPPORT=ON -DWITH_JPEG=ON
-DBUILD_JPEG=OFF -DJPEG_INCLUDE_DIR=/opt/libjpeg-turbo/include/
-DJPEG_LIBRARY=/opt/libjpeg-turbo/lib/libjpeg.a -DUSE_VFPV3=ON -DUSE_NEON=ON ..
```

Take careful note of the information displayed to stdout after cmake runs. You will want to make sure that `FFMPEG` support is enabled, `-mfpu=neon` appears in the release build flags, and that the JPEG codec is libjpeg-turbo. If everything looks okay, go ahead and begin the build... Even with `distcc`, the build will take awhile.

```
make
```

Now you can install the files to their default location, `/usr/local` and exit as the root user.

```
make install
exit
```

At this point, you will want to make sure that you can compile a simple OpenCV program. You can use the `framegrabberCV.c` program used in the testing section. I did this by navigating to the directory where I saved the program and executing: (See the Testing section)

```
gcc framegrabberCV.c -o framegrabberCV `pkg-config --cflags --opencv opencv
libv4l2` -lm
```

If the program compiles successfully, its likely that everything installed correctly. We can now unmount the USB drive. (NOTE: I had some issues with the USB drive being corrupted after removing it from the BBB. You may want to first compress the OpenCV source directory (including build files) to a `.tar.gz` and transfer it to your PC using `scp`.) When you are ready to remove the drive:

```
sudo umount /dev/sda1
sudo umount /mnt/ext2
sudo eject /dev/sda
sudo rm -rf /mnt/ext2
```

4. Testing

Testing the Framerate

Now it's time to test the framerate. First, download the program `framegrabberCV.c` to the BBB and compile it.

```
gcc framegrabberCV.c -o framegrabberCV `pkg-config --cflags --opencv opencv
libv4l2` -lm
```

I wanted to make sure the processor was running at its full 1 GHz, so I set it using:

```
sudo cpufreq-set -g performance
```

Note that you can set this back to default by executing:

```
sudo cpufreq-set -g ondemand
```

Make sure that the webcam is plugged in. Now you can test the framerate. [17]

```
time ./framegrabberCV -f mjpeg -H 480 -W 640 -c 1000 -I 30 -o
```

Now take the the number of frames you captured and converted to OpenCV image objects (in this case, 1000) and divide it by the "real" time provided by the `time` function to get the framerate.

Incorporating Custom Capture Code as an OpenCV object

If you like, you can make some changes to the custom capture code in `frmegrabberCV.c` and compile it as a C++ class instead of a standalone command line program. That way you can create a capture object within your OpenCV code, grab and decode images, and process it using OpenCV functions and methods.

< To be continued...>

For now, take a look at [18]

Acknowledgments

I would like to give thanks to those in the open source community who have been enormously helpful. Thanks to Matthew Witherwax for all of his help in arriving at a solution [3]; Martin Fox for providing custom Video4Linux capture code [18]; Alexander Corcoran for answering my newbie Linux questions [19]; Robert C. Nelson for maintaining Ubuntu for BBB [7]; Max Thrun for answering some questions about the PS3 Eye drivers [20]; and everyone else who has had the patience to help me along my way.

References

[1] http://mechomaniac.com/node/32

[2] http://linuxtv.org/downloads/v4l-dvb-apis/

[3] http://blog.lemoneerlabs.com/post/BBB-webcams

[4] http://docs.opencv.org/modules/highgui/doc/reading_and_writing_images_and_video.html

[5] http://blog.lemoneerlabs.com/post/opencv-mjpeg

[6] http://en.wikipedia.org/wiki/Inter_frame

[7] http://elinux.org/BeagleBoardUbuntu

[8] http://www.arm.com/products/processors/technologies/neon.php

[9] http://libjpeg-turbo.virtualgl.org

[10] http://jeremy-nicola.info/portfolio-item/cross-compilation-distributed-compilation-for-the-raspberry-pi/

[11] http://eewiki.net/display/linuxonarm/BeagleBone+Black#BeagleBoneBlack-ARMCrossCompiler:GCC

[12] https://code.google.com/p/distcc/

[13] http://www.sitepoint.com/ubuntu-12-04-lts-precise-pangolin-using-gparted-to-partition-a-hard-disk/

[14] http://docs.opencv.org/doc/tutorials/introduction/linux_install/linux_install.html#linux-installation

[15] http://docs.opencv.org/doc/tutorials/introduction/crosscompilation/arm_crosscompile_with_cmake.html

[16] http://elinux.org/images/1/1c/Optimizing_the_Embedded_Platform_Using_OpenCV.pdf

[17] https://groups.google.com/forum/#!msg/beagleboard/G5Xs2JuwD_4/gCl48Nj61BwJ

[18] https://bitbucket.org/beldenfox/cvcapture/src

[19] http://digitalcommons.calpoly.edu/cpesp/50/

[20] http://bear24rw.blogspot.com/2009/11/ps3-eye-driver-patch.html

Appendix

framegrabberCV.c (Matthew Witherwax)

Download

```
/*******************************************************************************
*   framegrabber Version 0.1                                                   *
*   Copyright (C) 2013 by Matthew Witherwax (lemoneer)                         *
*   lemoneer@outlook.com                                                       *
*   blog.lemoneerlabs.com                                                      *
*                                                                              *
*   based on V4L2 Specification, Appendix B: Video Capture Example             *
*   (http://linuxtv.org/downloads/v4l-dvb-apis/capture-example.html)           *
*   and work by Matthew Witherwax on v4l2grab                                  *
*   (https://github.com/twam/v4l2grab)                                         *
*******************************************************************************
*******************************************************************************
                            BSD LICENSE

Copyright (c) 2013, Matthew Witherwax
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

 * Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.

 * Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in
   the documentation and/or other materials provided with the
   distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*******************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>
#include <assert.h>
#include <getopt.h>             /* getopt_long() */
#include <fcntl.h>              /* low-level i/o */
#include <unistd.h>
#include <errno.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/mman.h>
```

```
#include <sys/ioctl.h>
#include <time.h>

#include <linux/videodev2.h>

#include "opencv2/core/core_c.h"
#include "opencv2/highgui/highgui_c.h"

#define CLEAR(x) memset(&(x), 0, sizeof(x))

enum io_method {
IO_METHOD_READ,
IO_METHOD_MMAP,
IO_METHOD_USERPTR,
};

struct buffer {
void *start;
size_t length;
};

static char         *dev_name = "/dev/video0";
static enum         io_method io = IO_METHOD_MMAP;
static int          fd = -1;
struct buffer       *buffers;
static unsigned int n_buffers;
static int          out_buf;
static int          frame_count = 1;
static int          set_format;
static unsigned int width = 640;
static unsigned int height = 480;
static unsigned int fps = 30;
static unsigned int timeout = 1;
static unsigned int timeouts_max = 1;
static char         *out_name = "capture.jpg";

/* Allowed formats: V4L2_PIX_FMT_YUYV, V4L2_PIX_FMT_MJPEG, V4L2_PIX_FMT_H264
*  The default will not be used unless the width and/or height is specified
*  but the user does not specify a pixel format */
static unsigned int pixel_format = V4L2_PIX_FMT_MJPEG;

/* Signal Handling
 * Clean up on Ctrl-C as opposed to leaving
 * the device in an inconsistent state*/
static int s_interrupted = 0;
static void s_signal_handler (int signal_value)
{
s_interrupted = 1;
}

static void s_catch_signals (void)
{
struct sigaction action;
action.sa_handler = s_signal_handler;
action.sa_flags = 0;
sigemptyset (&action.sa_mask);
sigaction (SIGINT, &action, NULL);
sigaction (SIGTERM, &action, NULL);
}
```

```
static void errno_exit(const char *s) {
fprintf(stderr, "%s error %d, %s\n", s, errno, strerror(errno));
exit(EXIT_FAILURE);
}

static int xioctl(int fh, int request, void *arg) {
int r;

do {
r = ioctl(fh, request, arg);
} while (-1 == r && EINTR == errno);

return r;
}

static int countP = 0;
static int cc = 0;
static void process_image(const void *p, int size) {
        if (out_buf) {
// Mike, Take this out to convert
// every frame captured
if (countP % 3 != 0)
{
countP += 1;
return;
}
countP += 1;
cc += 1;
CvMat mat;
IplImage * img;

mat = cvMat(480, 640, CV_8UC3, (void*)p);

// decode the image
img = cvDecodeImage(&mat, 1);

            // release the image
cvReleaseImage(&img);
}
}

static int read_frame(void) {
struct v4l2_buffer buf;
unsigned int i;

switch (io) {
case IO_METHOD_READ:
if (-1 == read(fd, buffers[0].start, buffers[0].length)) {
switch (errno) {
case EAGAIN:
return 0;

case EIO:
/* Could ignore EIO, see spec. */

/* fall through */

default:
errno_exit("read");
}
```

```
}

process_image(buffers[0].start, buffers[0].length);
break;

case IO_METHOD_MMAP:
CLEAR(buf);

buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
buf.memory = V4L2_MEMORY_MMAP;

if (-1 == xioctl(fd, VIDIOC_DQBUF, &buf)) {
switch (errno) {
case EAGAIN:
return 0;

case EIO:
/* Could ignore EIO, see spec. */

/* fall through */

default:
errno_exit("VIDIOC_DQBUF");
}
}

assert(buf.index < n_buffers);

process_image(buffers[buf.index].start, buf.bytesused);

if (-1 == xioctl(fd, VIDIOC_QBUF, &buf))
errno_exit("VIDIOC_QBUF");
break;

case IO_METHOD_USERPTR:
CLEAR(buf);

buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
buf.memory = V4L2_MEMORY_USERPTR;

if (-1 == xioctl(fd, VIDIOC_DQBUF, &buf)) {
switch (errno) {
case EAGAIN:
return 0;

case EIO:
/* Could ignore EIO, see spec. */

/* fall through */

default:
errno_exit("VIDIOC_DQBUF");
}
}

for (i = 0; i < n_buffers; ++i)
if (buf.m.userptr == (unsigned long) buffers[i].start
&& buf.length == buffers[i].length)
break;
```

```
assert(i < n_buffers);

process_image((void *) buf.m.userptr, buf.bytesused);

if (-1 == xioctl(fd, VIDIOC_QBUF, &buf))
errno_exit("VIDIOC_QBUF");
break;
}

return 1;
}

static void grab_frames(void) {
        clock_t begin, end;
        double time_spent;

unsigned int count;
unsigned int timeout_count;

count = frame_count;
timeout_count = timeouts_max;

        begin = clock();

while (count-- > 0) {
for (;;) {

if (s_interrupted) {
fprintf(stderr, "\nInterrupt received - aborting capture\n");
return;
}

fd_set fds;
struct timeval tv;
int r;

FD_ZERO(&fds);
FD_SET(fd, &fds);

/* Timeout. */
tv.tv_sec = timeout;
tv.tv_usec = 0;

r = select(fd + 1, &fds, NULL, NULL, &tv);

if (-1 == r) {
if (EINTR == errno)
continue;
errno_exit("select");
}

if (0 == r) {
if (timeout_count > 0) {
timeout_count--;
} else {
fprintf(stderr, "select timeout\n");
exit(EXIT_FAILURE);
}
}
```

```
if (read_frame())
break;
/* EAGAIN - continue select loop. */
}
}
        end = clock();
        time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
        fprintf(stderr, "Captured %i frames and Processed %i in %f seconds\n", frame_count, cc, time_spent);
}

static void mainloop(void) {
grab_frames();
}

static void stop_capturing(void) {
enum v4l2_buf_type type;

switch (io) {
case IO_METHOD_READ:
/* Nothing to do. */
break;

case IO_METHOD_MMAP:
case IO_METHOD_USERPTR:
type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
if (-1 == xioctl(fd, VIDIOC_STREAMOFF, &type))
errno_exit("VIDIOC_STREAMOFF");
break;
}
}

static void start_capturing(void) {
unsigned int i;
enum v4l2_buf_type type;

switch (io) {
case IO_METHOD_READ:
/* Nothing to do. */
break;

case IO_METHOD_MMAP:
for (i = 0; i < n_buffers; ++i) {
struct v4l2_buffer buf;

CLEAR(buf);
buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
buf.memory = V4L2_MEMORY_MMAP;
buf.index = i;

if (-1 == xioctl(fd, VIDIOC_QBUF, &buf))
errno_exit("VIDIOC_QBUF");
}
type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
if (-1 == xioctl(fd, VIDIOC_STREAMON, &type))
errno_exit("VIDIOC_STREAMON");
break;

case IO_METHOD_USERPTR:
for (i = 0; i < n_buffers; ++i) {
struct v4l2_buffer buf;
```

```
CLEAR(buf);
buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
buf.memory = V4L2_MEMORY_USERPTR;
buf.index = i;
buf.m.userptr = (unsigned long) buffers[i].start;
buf.length = buffers[i].length;

if (-1 == xioctl(fd, VIDIOC_QBUF, &buf))
errno_exit("VIDIOC_QBUF");
}
type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
if (-1 == xioctl(fd, VIDIOC_STREAMON, &type))
errno_exit("VIDIOC_STREAMON");
break;
}
}

static void uninit_device(void) {
unsigned int i;

switch (io) {
case IO_METHOD_READ:
free(buffers[0].start);
break;

case IO_METHOD_MMAP:
for (i = 0; i < n_buffers; ++i)
if (-1 == munmap(buffers[i].start, buffers[i].length))
errno_exit("munmap");
break;

case IO_METHOD_USERPTR:
for (i = 0; i < n_buffers; ++i)
free(buffers[i].start);
break;
}

free(buffers);
}

static void init_read(unsigned int buffer_size) {
buffers = calloc(1, sizeof (*buffers));

if (!buffers) {
fprintf(stderr, "Out of memory\n");
exit(EXIT_FAILURE);
}

buffers[0].length = buffer_size;
buffers[0].start = malloc(buffer_size);

if (!buffers[0].start) {
fprintf(stderr, "Out of memory\n");
exit(EXIT_FAILURE);
}
}

static void init_mmap(void) {
struct v4l2_requestbuffers req;
```

```
CLEAR(req);

req.count = 4;
req.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
req.memory = V4L2_MEMORY_MMAP;

if (-1 == xioctl(fd, VIDIOC_REQBUFS, &req)) {
if (EINVAL == errno) {
fprintf(stderr, "%s does not support "
"memory mapping\n", dev_name);
exit(EXIT_FAILURE);
} else {
errno_exit("VIDIOC_REQBUFS");
}
}

if (req.count < 2) {
fprintf(stderr, "Insufficient buffer memory on %s\n",
dev_name);
exit(EXIT_FAILURE);
}

buffers = calloc(req.count, sizeof (*buffers));

if (!buffers) {
fprintf(stderr, "Out of memory\n");
exit(EXIT_FAILURE);
}

for (n_buffers = 0; n_buffers < req.count; ++n_buffers) {
struct v4l2_buffer buf;

CLEAR(buf);

buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
buf.memory = V4L2_MEMORY_MMAP;
buf.index = n_buffers;

if (-1 == xioctl(fd, VIDIOC_QUERYBUF, &buf))
errno_exit("VIDIOC_QUERYBUF");

buffers[n_buffers].length = buf.length;
buffers[n_buffers].start =
mmap(NULL /* start anywhere */,
buf.length,
PROT_READ | PROT_WRITE /* required */,
MAP_SHARED /* recommended */,
fd, buf.m.offset);

if (MAP_FAILED == buffers[n_buffers].start)
errno_exit("mmap");
}
}

static void init_userp(unsigned int buffer_size) {
struct v4l2_requestbuffers req;

CLEAR(req);
```

```
req.count = 4;
req.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
req.memory = V4L2_MEMORY_USERPTR;

if (-1 == xioctl(fd, VIDIOC_REQBUFS, &req)) {
if (EINVAL == errno) {
fprintf(stderr, "%s does not support "
"user pointer i/o\n", dev_name);
exit(EXIT_FAILURE);
} else {
errno_exit("VIDIOC_REQBUFS");
}
}

buffers = calloc(4, sizeof (*buffers));

if (!buffers) {
fprintf(stderr, "Out of memory\n");
exit(EXIT_FAILURE);
}

for (n_buffers = 0; n_buffers < 4; ++n_buffers) {
buffers[n_buffers].length = buffer_size;
buffers[n_buffers].start = malloc(buffer_size);

if (!buffers[n_buffers].start) {
fprintf(stderr, "Out of memory\n");
exit(EXIT_FAILURE);
}
}
}

static void init_device(void) {
struct v4l2_capability cap;
struct v4l2_cropcap cropcap;
struct v4l2_crop crop;
struct v4l2_format fmt;
struct v4l2_streamparm frameint;
unsigned int min;

if (-1 == xioctl(fd, VIDIOC_QUERYCAP, &cap)) {
if (EINVAL == errno) {
fprintf(stderr, "%s is no V4L2 device\n",
dev_name);
exit(EXIT_FAILURE);
} else {
errno_exit("VIDIOC_QUERYCAP");
}
}

if (!(cap.capabilities & V4L2_CAP_VIDEO_CAPTURE)) {
fprintf(stderr, "%s is no video capture device\n",
dev_name);
exit(EXIT_FAILURE);
}

switch (io) {
case IO_METHOD_READ:
if (!(cap.capabilities & V4L2_CAP_READWRITE)) {
fprintf(stderr, "%s does not support read i/o\n",
```

```
dev_name);
exit(EXIT_FAILURE);
}
break;

case IO_METHOD_MMAP:
case IO_METHOD_USERPTR:
if (!(cap.capabilities & V4L2_CAP_STREAMING)) {
fprintf(stderr, "%s does not support streaming i/o\n",
dev_name);
exit(EXIT_FAILURE);
}
break;
}


/* Select video input, video standard and tune here. */


CLEAR(cropcap);

cropcap.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

if (0 == xioctl(fd, VIDIOC_CROPCAP, &cropcap)) {
crop.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
crop.c = cropcap.defrect; /* reset to default */

if (-1 == xioctl(fd, VIDIOC_S_CROP, &crop)) {
switch (errno) {
case EINVAL:
/* Cropping not supported. */
break;
default:
/* Errors ignored. */
break;
}
}
} else {
/* Errors ignored. */
}


CLEAR(fmt);

fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
if (set_format) {
fmt.fmt.pix.width = width;
fmt.fmt.pix.height = height;
fmt.fmt.pix.pixelformat = pixel_format;
fmt.fmt.pix.field = V4L2_FIELD_INTERLACED;

if (-1 == xioctl(fd, VIDIOC_S_FMT, &fmt))
errno_exit("VIDIOC_S_FMT");

                if (fmt.fmt.pix.pixelformat != pixel_format) {
                        fprintf(stderr,"Libv4l didn't accept pixel format. Can't proceed.\n");
                        exit(EXIT_FAILURE);
                }

/* Note VIDIOC_S_FMT may change width and height. */
```

```
} else {
/* Preserve original settings as set by v4l2-ctl for example */
if (-1 == xioctl(fd, VIDIOC_G_FMT, &fmt))
errno_exit("VIDIOC_G_FMT");
}

CLEAR(frameint);

/* Attempt to set the frame interval. */
frameint.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
frameint.parm.capture.timeperframe.numerator = 1;
frameint.parm.capture.timeperframe.denominator = fps;
if (-1 == xioctl(fd, VIDIOC_S_PARM, &frameint))
fprintf(stderr, "Unable to set frame interval.\n");

/* Buggy driver paranoia. */
min = fmt.fmt.pix.width * 2;
if (fmt.fmt.pix.bytesperline < min)
fmt.fmt.pix.bytesperline = min;
min = fmt.fmt.pix.bytesperline * fmt.fmt.pix.height;
if (fmt.fmt.pix.sizeimage < min)
fmt.fmt.pix.sizeimage = min;

switch (io) {
case IO_METHOD_READ:
init_read(fmt.fmt.pix.sizeimage);
break;

case IO_METHOD_MMAP:
init_mmap();
break;

case IO_METHOD_USERPTR:
init_userp(fmt.fmt.pix.sizeimage);
break;
}
}

static void close_device(void) {
if (-1 == close(fd))
errno_exit("close");

fd = -1;
}

static void open_device(void) {
struct stat st;

if (-1 == stat(dev_name, &st)) {
fprintf(stderr, "Cannot identify '%s': %d, %s\n",
dev_name, errno, strerror(errno));
exit(EXIT_FAILURE);
}

if (!S_ISCHR(st.st_mode)) {
fprintf(stderr, "%s is no device\n", dev_name);
exit(EXIT_FAILURE);
}

fd = open(dev_name, O_RDWR /* required */ | O_NONBLOCK, 0);
```

```
if (-1 == fd) {
fprintf(stderr, "Cannot open '%s': %d, %s\n",
dev_name, errno, strerror(errno));
exit(EXIT_FAILURE);
}
}

static void usage(FILE *fp, int argc, char **argv) {
fprintf(fp,
"Usage: %s [options]\n\n"
"Version 1.0\n"
"Options:\n"
"-d | --device name   Video device name [%s]\n"
"-h | --help          Print this message\n"
"-m | --mmap          Use memory mapped buffers [default]\n"
"-r | --read          Use read() calls\n"
"-u | --userp         Use application allocated buffers\n"
"-W | --width         Set image width\n"
"-H | --height        Set image height\n"
"-I | --interval      Set frame interval (fps) [%i]\n"
"-f | --format        Set pixel format [YUYV | MJPG | H264]\n"
"-t | --timeout       Set capture timeout in seconds [%i]\n"
"-T | --timeouts-max  Set the maximum number of timeouts [%i]\n"
"-o | --output        Outputs stream to stdout\n"
"-c | --count         Number of frames to grab [%i]\n"
"",
argv[0], dev_name, fps, timeout, timeouts_max, frame_count);
}

static const char short_options[] = "d:hmruW:H:I:f:t:T:oc:";

static const struct option
long_options[] = {
{ "device",        required_argument, NULL, 'd'},
{ "help",          no_argument,       NULL, 'h'},
{ "mmap",          no_argument,       NULL, 'm'},
{ "read",          no_argument,       NULL, 'r'},
{ "userp",         no_argument,       NULL, 'u'},
{ "width",         required_argument, NULL, 'W'},
{ "height",        required_argument, NULL, 'H'},
{ "interval",      required_argument, NULL, 'I'},
{ "format",        required_argument, NULL, 'f'},
{ "timeout",       required_argument, NULL, 't'},
{ "timeouts-max",  required_argument, NULL, 'T'},
{ "output",        no_argument,       NULL, 'o'},
{ "count",         required_argument, NULL, 'c'},
{ 0, 0, 0, 0}
};

int main(int argc, char **argv) {
s_catch_signals ();
for (;;) {
int idx;
int c;

c = getopt_long(argc, argv,
short_options, long_options, &idx);

if (-1 == c)
```

```
break;

switch (c) {
case 0: /* getopt_long() flag */
break;

case 'd':
dev_name = optarg;
break;

case 'h':
usage(stdout, argc, argv);
exit(EXIT_SUCCESS);

case 'm':
io = IO_METHOD_MMAP;
break;

case 'r':
io = IO_METHOD_READ;
break;

case 'u':
io = IO_METHOD_USERPTR;
break;

case 'W':
// set width
width = atoi(optarg);
set_format++;
break;

case 'H':
// set height
height = atoi(optarg);
set_format++;
break;

case 'I':
// set fps
fps = atoi(optarg);
break;

case 'f':
// set pixel format
if (strcmp(optarg, "YUYV") == 0 || strcmp(optarg, "yuyv") == 0) {
pixel_format = V4L2_PIX_FMT_YUYV;
set_format++;
} else if (strcmp(optarg, "MJPG") == 0 || strcmp(optarg, "mjpg") == 0) {
pixel_format = V4L2_PIX_FMT_MJPEG;
set_format++;
} else if (strcmp(optarg, "H264") == 0 || strcmp(optarg, "h264") == 0) {
pixel_format = V4L2_PIX_FMT_H264;
set_format++;
}
break;

case 't':
// set timeout
timeout = atoi(optarg);
```

```
break;

case 'T':
// set max timeout
timeouts_max = atoi(optarg);
break;

case 'o':
out_buf++;
break;

case 'c':
errno = 0;
frame_count = strtol(optarg, NULL, 0);
if (errno)
errno_exit(optarg);
break;

default:
usage(stderr, argc, argv);
exit(EXIT_FAILURE);
}
}
        clock_t begin, end;
        double time_spent;

        begin = clock();
open_device();
init_device();
start_capturing();
        end = clock();
        time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
        fprintf(stderr, "Startup took %f seconds\n", time_spent);

mainloop();

        begin = clock();
stop_capturing();
uninit_device();
close_device();
        end = clock();
        time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
        fprintf(stderr, "Shutdown took %f seconds\n", time_spent);
fprintf(stderr, "\n");
return 0;
}
```