Handling IEEE1394 stereo cameras on a Linux machine

Başar Uğur
Boğaziçi University, Computer Engineering

Contents

Introduction

Operating on a Linux machine, one comes across to many problems related to developing software using special hardware devices. Because the drivers referred by the operating system may be outdated, and this leads to inconsistencies and program crashes. Furthermore, the drivers written by individuals, though professional, may contain tweaks. This document is about the experience earned on the way to use a IEEE1394 (*firewire*) stereo camera on an Ubuntu Linux machine, in the most efficient way possible.

Setup

The system setup consists of:

- PointGrey Bumblebee2 Stereo Camera
- IEEE1394 cable
- Ubuntu 8.04 Linux machine with IEEE1394 port

The natural way to use the camera is to look for the relevant drivers and libraries in the ubuntu repository. Initially, that was what we did. We used the drivers and libraries found in the repository, mainly under the name libdc1394 1.x[1], written by Damien Douxchamps. For testing, we searched for a sample program. We found Coriander[2], a program also written by the library developer. Similarly, the *1.x* version of Coriander exists in the ubuntu repositories.
These drivers and programs worked well for the simple video setup, with 640x480 RGB colored frames at 30 FPS. Using the libdc1394 1.x library, it became possible to embed the camera functionality in our software.

However, what we needed was not real-time *mono* video. Our system required *stereo* video for processing. Our first attempt was to use the *panning* property of the camera manually. At each frame grab, we tried to obtain two images from the left and right lenses by using the `dc1394_set_pan` functionality in the camera library. This attempt caused unavoidable glitches and frame jumps while processing.

These problems were due to the fact that we were not aware of PointGrey's technical considerations. As written on the relevant web page titled “Transmitting multiple images from stereo cameras”[3]: *To*

1 http://damien.douxchamps.net/ieee1394/libdc1394/v1.x/faq/
2 http://damien.douxchamps.net/ieee1394/coriander/
3 http://www.ptgrey.com/support/kb/index.asp?a=4&q=312

capture two images simultaneously, one should set the Format_7 Mode 3 pixel format to Raw16 for color cameras. Then we noticed that "Format_7" was available as an option in the Coriander GUI. However, with the 1.x versions of the library and the program, it was not possible to obtain two images simultaneously, due to some error in the library.

Then we searched for more recent versions of libdc1394 and Coriander, and found out that 2.x versions of them had been released. We downloaded and compiled these. Having everything in the right place, we managed to obtain stereo images simultaneously. Nevertheless, there have been certain issues that need to be considered. We will examine these in the next section.

Issues and Solutions

In this section, we share the technical issues related to obtaining two images (left and right) simultaneously from a stereo camera. All the details given are related to Ubuntu 8.04 with Gnome Desktop ver. 2.x specifically.

User access to video hardware

In Linux operating system, access to many of the devices is limited to the root (or *super*) user. Although this limitation seems editable under the window system in *System* > *Administration* > *Authorizations*, by experience we can say that giving access through this interface does not apply actually. Instead, one has two options:

1. Changing the permission of *device files* so that all the users can access the device: This is achieved by browsing to folder /dev/ and running the commands as root user:
   - `chmod 777 raw1394` for raw firewire data access, and
   - `chmod 777 video1394` for firewire video device access.
2. Running the programs (that need to access the video device) as root user: This option becomes cumbersome when you would like to run the program frequently under a programming environment. Then you would like to open the programming environment as root user too. But you rarely close it, and therefore it becomes unsafe due to the wide access rights of the root user, as many people may use the environment.

We preferred the first option because it is a task-specific solution, and you need to do it only once for a user session.

ISO channel and bandwidth allocation problems

When a program uses the firewire video device and the data handles are not closed properly, some data may remain allocated on the firewire cable. These are, the opened ISO channel and the allocated bandwidth. Then if a new program tries to use the device, and therefore needs to allocate new channel and data, the library gives an I/O error.

This issue is solved by the methods supplied by the libdc1394 2.x library. The first method is `dc1394_iso_release_bandwidth`, which enables releasing the currently allocated bandwidth on the camera. The second method is `dc1394_iso_release_channel`, which enables releasing the currently allocated ISO channel on the camera. We apply these methods during the initialization process to suppress I/O errors.

However, if the unusually terminating program is the same program that tries to access the device again, these methods do not work on some occasions. In order to suppress these cases, a simple "outer" program which only applies these releasing methods is called during initalization, by using `system` command.

Invisible processes running on the background and holding the device

Another occasion is due to unusual terminations of the programs that use the video device. Because of the nature of the termination, no GUI window related to the program is visible. If you try to run another program that accesses the video device, you get I/O error.

Hopefully, you know the names of the few processes that possibly hold the video device. To kill these processes, you open a terminal and run the command:

```
ps aux | grep <possible-process-name>
```

A list of processes matching your search with `grep` will come out. Then you kill the processes by using their id such as `kill 18803`, and new programs can be opened.

Appendix: Example Interface Class for Stereo Camera

A typical program (or class, in this example) using the stereo video device contains standard initializations, terminations and data structures supplied by the libdc1394 2.x library. We present the interface class for firewire stereo camera here, with added explanations.

```
#include "dc1394/dc1394.h"

class IFiWiCamera
{
public:

    dc1394_t *dc_handle;
    dc1394camera_list_t *cam_list;
    dc1394camera_t *camera;
    dc1394video_frame_t *vf;

    dc1394error_t err;

    uint8_t *imageBuf, *imageBufRGB; // direct image buffers

    IFiWiCamera()
    {
        system("../CameraCleanup");
    }

    ~IFiWiCamera()
    {
        err = dc1394_video_set_transmission(camera, DC1394_OFF);

        release_iso_and_bw();

        dc1394_camera_free(camera);

        dc1394_free(dc_handle);
    }

    bool init()
    {
        // First we try to find a camera
        dc_handle = dc1394_new();
        err = dc1394_camera_enumerate (dc_handle, &cam_list);
        if (err != DC1394_SUCCESS)
        {
            printf("Could not get Camera List: %d\n", err);
            return false;
        }

        if (cam_list->num == 0)
        {
            printf("No cameras found");
            return false;
        }

        // we just use the first one returned
        camera = dc1394_camera_new (dc_handle, cam_list->ids[0].guid);

        if (!camera)
        {
           printf("Failed to initialize camera with guid %l", cam_list->ids[0].guid);
           return false;
        }
```

We deallocate ISO channel and bandwidth by an outer program.

The ISO channel and bandwidth that the class itself has used is being deallocated.

```
    dc1394_camera_free_list(cam_list);

    release_iso_and_bw();

    if ((err = dc1394_video_set_mode(camera, DC1394_VIDEO_MODE_FORMAT7_3)) != DC1394_SUCCESS)
        cout << "Could not set Format7 mode: " << err << endl;

    dc1394_capture_setup(camera, 4, DC1394_CAPTURE_FLAGS_DEFAULT);

    dc1394video_mode_t vm;
    if ((err = dc1394_video_get_mode(camera, &vm)) == DC1394_SUCCESS)
        cout << "Video mode: " << vm << endl;

    if ((err = dc1394_feature_set_value(camera, DC1394_FEATURE_GAIN, gain)) != DC1394_SUCCESS)
        cout << "Could not set gain: " << err << endl;

    uint32_t val;
    dc1394speed_t sp;
    if ((err = dc1394_video_get_iso_channel(camera, &val)) == DC1394_SUCCESS)
        cout << "ISO Channel: " << val << endl;

    if ((err = dc1394_video_get_iso_speed(camera, &sp)) == DC1394_SUCCESS)
        cout << "ISO Speed: " << sp << endl;

    dc1394_video_set_transmission(camera, DC1394_ON);                  /* Start transmission */

    // get a sample video frame for grabbing the resolution:

    if ((err = dc1394_capture_dequeue(camera, DC1394_CAPTURE_POLICY_WAIT, &vf)) != DC1394_SUCCESS ||
        vf == NULL)
    {
        cout << "Could not capture first frame: " << err << endl;
        return false;
    }

    imageBuf = new uint8_t[vf->size[0] * vf->size[1] * 2];
    imageBufRGB = new uint8_t[vf->size[0] * vf->size[1] * 6];

    err = dc1394_capture_enqueue(camera, vf);

    return true;
}

void release_iso_and_bw()
{
    uint32_t val;

    if ( dc1394_video_get_bandwidth_usage(camera, &val) == DC1394_SUCCESS &&
         dc1394_iso_release_bandwidth(camera, val) == DC1394_SUCCESS )
        cout << "Succesfully released " << val << " bytes of Bandwidth." << endl;

    if ( dc1394_video_get_iso_channel(camera, &val) == DC1394_SUCCESS &&
         dc1394_iso_release_channel(camera, val) == DC1394_SUCCESS )
        cout << "Succesfully released ISO channel #" << val << "." <<  endl;
}
```

We try to release previous ISO channel and bandwidth usage.

We set video mode to Format7.

In this function, first we get the bandwidth usage value and release the resource accordingly, and then do the same for the ISO channel.

There are three main steps for obtaining stereo images by Format7 data structure:

- Capture video frame into `dc1394video_frame_t` structure.
- Deinterlace the "interlaced" stereo images into two separate images (image buffers).
- De-Bayer the separate images and obtain the RGB versions for processing.

These steps are followed in the `capture_image` method.

```
bool capture_image()
{
    dc1394video_frame_t* vf_temp;

    dc1394_feature_set_value(camera, DC1394_FEATURE_GAIN, gain);

    bool res = false;

    if ((err = dc1394_capture_dequeue(camera, DC1394_CAPTURE_POLICY_WAIT, &vf_temp)) != DC1394_SUCCESS)
        cout << "Could not capture frame: " << err << endl;
    else if (vf_temp == NULL)
        cout << "Video frame is NULL: " << (vf_temp) << endl;
    else
    {
        vf = vf_temp;
```

```
            if ((err = dc1394_deinterlace_stereo(vf->image, imageBuf, vf->size[0], vf->size[1]*2)) != DC1394_SUCCESS)
                cout << "Could not deinterlace stereo images: " << err << endl;

            if ((err = dc1394_bayer_decoding_8bit(imageBuf, imageBufRGB,
                                                  vf->size[0], vf->size[1]*2,
                                                  DC1394_COLOR_FILTER_RGGB,
                                                  DC1394_BAYER_METHOD_NEAREST)) != DC1394_SUCCESS)
                cout << "Could not apply Bayer conversion: " << err << endl;

            res = true;
        }

        if ( vf_temp && (err = dc1394_capture_enqueue(camera, vf_temp)) != DC1394_SUCCESS )
            cout << "Could not return frame: " << err << endl;

        return res;
    }
};
```