Stettbacher Signal Processing
Margrit Rainer Strasse 12a
CH-8050 Zürich

Phone: +41 43 299 57 23
Fax: +41 43 299 57 25
E-Mail: cam@stettbacher.ch

STETTBACHER SIGNAL PROCESSING

O-3000 Camera Series

Documentation of Driver Package and Demo Application

Version 1.20
2013-05-24

Abstract: This document describes the driver package and the demo application provided with the O-3000 camera series. The first part of the text shows how to use the driver package and demo application. Then it explains how the user can modify the source code and build his own application.

Contents

1 Introduction

The O-3000 camera series is accompanied with a driver package. The driver package hides the complexity involved in interfacing the USB and offers a platform-independent application programming interface. The user-space driver is based on the open-source project libusbx, works with Linux, MacOS X, and MS Windows, and is available as open-source software too.

The driver package contains:

- The O-3000 camera driver library (binaries and sources).
- A Java native interface (JNI) wrapper library for the O-3000 camera driver library (binaries and sources).
- A Java Demo application (Java archive (JAR) and sources).

2 Driver Installation

The driver package can be obtained from the Stettbacher Signal Processing website [1], see in the download section. Make sure, the appropriate installer is chosen for your operating system. Table 1 lists the available installers. Please note, that *${VERSION}* stands for the current installer package's version number.

2.1 Installation under Linux / MacOS X

Download the installer suitable to your system. Open a shell or terminal in order to execute the commands below. The pseudo-prompt >> just indicates, that the text following the prompt must be typed/copied to the shell and executed there. If a command requires administrator privileges, either prepend the command *sudo* to the actual command you would like to execute, or open a root shell. The installer is designed to minimise the impact on your system and keeps all files contained in a single folder.

2.1.1 Create Destination Folder

Make sure, the folder */opt* exists on your system. If not, create it (might require administrator privileges):

Package name	Operating system
o3000_Linux_i686_installer_${VERSION}.run.bz2 o3000_Linux_i686_installer_${VERSION}.run.zip	Linux Intel/AMD 32-bit machines
o3000_Linux_x86_64_installer_${VERSION}.run.bz2 o3000_Linux_x86_64_installer_${VERSION}.run.zip	Linux Intel/AMD 64-bit machines
o3000_MacOSX_installer_${VERSION}.run.bz2 o3000_MacOSX_installer_${VERSION}.run.zip	MacOS X Intel 64-bit machines only
o3000_WinXP_i686_installer_${VERSION}.exe	Windows XP Intel/AMD 32-bit machines
o3000_Win7_i686_installer_${VERSION}.exe	Windows 7 Intel/AMD 32-bit machines
o3000_Win7_x86_64_installer_${VERSION}.exe	Windows 7 Intel/AMD 64-bit machines

Table 1: Available installer packages.

```
>> mkdir /opt
```

2.1.2 Install the driver package

In the shell, change to the folder where the installer was downloaded to. Extract the installer using *unzip* or *bunzip2*, depending on the installer version you have chosen. Then, make the installer executable and start it (might require administrator privileges). The example below assumes, the installer (in BZ2 format) was downloaded to */temp_dir* and the operating system is Linux on a 64-bit Intel/AMD machine. Change both according to your needs.

```
>> cd /temp_dir
>> bunzip2 ./o3000_Linux_x86_64_installer_V1.00.run.bz2
>> chmod +x ./o3000_Linux_x86_64_installer_V1.00.run
>> ./o3000_Linux_x86_64_installer_V1.00.run
```

The installer will create the folder */opt/o3000*, containing the driver package contents. This folder will be referenced to as *$O3000_DIR* through this document. Either keep in mind this location, or just create an environment variable as a shortcut, and literally use the commands as written down:

```
>> export O3000_DIR=/opt/o3000
>> cd $O3000_DIR
```

2.2 Installation under Windows

Download the installer suitable to your system. Then, execute the installer and follow the on-screen instructions. The installer will create an entry in the start menu, from where the driver can be removed if not required anymore.

If you connect the camera for the first time, Windows will ask for a driver. Choose *manual search* and navigate to the folder *usb_driver* in the installation directory. Install the driver located there.

Through this document, the location where the driver package was installed to, will be referred to as *%O3000_DIR%*. Either replace this variable with the actual folder name at every occurence, or define an environment variable in a command shell and use the variable as is.

```
>> set O3000_DIR="C:\Program Files\O30xx Camera Demo V1.0"
>> cd %O3000_DIR%
```

3 Running the Demo Application

The demo application is written in Java. It demonstrates some basic functionality of the O-3000 camera series. It relies on the o3000 driver library, the corresponding JNI wrapper o3000_jni, and the OpenCV library for visualisation. The demo application comes packed in a Java archive file called *o3000.jar*. For execution, the Java Runtime Environment Version 6 (JRE6) or later is required. Make sure, that your system provides the runtime environment. Download it from [2] and install it if required.

3.1 Linux / MacOS X

Change to the binaries folder and start the demo application with the startup script provided:

```
>> cd $O3000_DIR/bin
>> ./run.sh
```

3.2 Windows

In order to start the demo app you have two possibilities, see sections 3.2.1 and 3.2.2.

3.2.1 Starting from the Command Line

Start the demo application by changing to the folder containing the binaries and executing the run.bat file:

```
>> cd %O3000_DIR%\bin
>> run.bat
```

In this run.bat file, all necessary path entries (directions to the library files) are already set. Therefore you can just double click it to run the app.

3.2.2 Starting from the GUI

The Java Virtual Machine is able to start the demo application directly by double-clicking the jar file. Please make sure, that all path environment variables are set correctly. The Java Library Path has to point to the JNI library location. For example update the path as follows:

```
>> PATH=%PATH%;C:\Program Files\o3000\lib;
```

3.2.3 Additional Information

Furthermore, you can compile the source code yourself and try to start the binary *.class files with an older JVM than 1.6. For a short description how to compile the source code (*.java), please see chapter 6.6.

4 The O-3000 Driver Explained

4.1 Purpose of the O-3000 Driver

The o3000 driver provides a simple and light-weight API to communicate with the O-3000 camera series. It abstracts user applications from the libusbx internas and deals with low-level interfacing which is tedious to implement and hard to debug if done the wrong way. The driver is written in plain C. For use in Java applications, the JNI wrapper may be used.

4.2 Driver API

The driver API is defined in the header file o3000.h under $O3000_DIR/o3000. Following functions are provided:

4.2.1 o3000_init

This function initialises the USB subsystem, allocates buffers and sets the callbacks. The XML and video receive callbacks are mandatory, i.e. they must not be NULL pointers. The logging callback is optional. Passing a NULL pointer disables logging. This function immediately returns. If no error has occured, the user can call o3000_connect now (see below) to start the driver.

4.2.2 o3000_connect

After the driver was initialised with o3000_init, o3000_connect processes the camera event loop. The function o3000_connect does not return until a disconnect events occurs, or the driver receives order to shut down (by o3000_stop; see below). Camera events will trigger the callbacks associated with the event.

Because of the blocking nature of this function, other functions which intend to interact with the driver, i.e. o3000_send_xml or o3000_stop, have to be called from another thread.

4.2.3 o3000_send_xml

This function serves to send XML messages to the camera. After the message has been submitted, the function immediately returns.

4.2.4 o3000_stop

Calling this function stops the driver by shutting down the event loop and causes the function o3000_connect to free the ressources allocated by o3000_init and to return.

4.2.5 Video Callback Function

This function, which has to be provided by the application developer to o3000_init, is called from the driver whenever an image frame is received. The video callback (or video handler) has to implement the desired functionality, e.g. displaying or storing the image received. As this function is triggered with the frame rate frequency, it must not block and should be programmed as efficient as possible. If the video handler is not able to keep pace with the camera, frames will be dropped!

4.2.6 XML Receive Callback Function

Similar to the video callback function, this handler has to be implemented by the developer and passed to o3000_init. It will be called, whenever a XML message is received from the camera.

4.2.7 Logging Callback Function

This optional handler allows to fetch logging messages from the o3000 driver. It might be useful for debugging.

5 O-3000 Driver in Own Applications

A minimal user application might look like the code below:

```
#include "o3000/o3000.h"

#define VID             0        // USB device vendor ID (0 for default).
#define PID             0        // USB device product ID (0 for default).
#define VIDEO_CACHE_LEN 7000000  // desired video cache size (in bytes).

void video_handler(unsigned char *current_buffer,
        struct o3000_image_props_t* props) {
        // insert your code here.
}

void xml_in_handler(unsigned char* buf, int len) {
        // insert your code here.
}

int main() {

        int retval;

        retval = o3000_init(VID,PID,VIDEO_CACHE_LEN,xml_in_handler,
                video_handler,NULL);
        if(!retval) {
                o3000_connect(); // blocking call.
        } else {
                // insert your error handling here.
        }
}
```

What this code does:

- First of all, the callback functions are defined.
- Code inside the video_handler processes the image data received by the driver.
- Code inside the xml_in_handler processes XML messages received by the driver.
- The main function calls o3000_init, to initialise libusb and to reserve memory for the video buffers. It also registers the callback functions.
- If the initialisation succeeded, the program connects to the camera driver. From now on, the video_handler will be called when an image frame arrives, and the xml_in_handler when a XML messages arrives, respectively.

- The application continues until the camera is disconnected and o3000_connect returns.

In fact, this example is pretty useless, because there is no way provided, to send commands to the camera. A real-world application would have to spawn another thread before calling o3000_connect, from where functions like e.g. o3000_send_xml could be called from.

6 Modifiying the Driver Package

The open-source nature of the O-3000 camera series driver encourages you to study and modify the sources. After installation of the driver package, all required parts (libraries, executables etc.) for running the demo application are already in place. At the same time, the folder structure is designed to enable you to recompile any parts at will.

6.1 A Note on Cross-Compiling

Although it might be feasible, cross-compiling is not encouraged. All instructions in this manual assume, that you compile natively, i.e. for the same operating system and flavour (64-bits/32-bits) as you are using for the toolchain. It might be possible to build 32-bit binaries on a 64-bit system by using the *-m32* CFLAG for GCC, but it is beyond the scope of this document to elaborate further.

6.2 A Note on Library Paths

As the O-3000 driver package tries to minimise interference with the package management of your operating system, the search paths for header files and libraries in the Makefiles are setup to point to locations below the installation directory:

- header files location: $O3000_DIR/include
- libraries location: $O3000_DIR/lib

For production use, you might like to change that in such a way, that default directories are used, which are in the compiler's or system's search path. E.g.:

- header files location: /usr/local/include
- libraries location: /usr/local/lib

For latter, the include and library paths in the Makefiles have to be changed accordingly. Also note, that the library search path in the startup script for the Java demo is manipulated for above mentioned reasons.

6.3 Prerequisites

6.3.1 GNU Toolchain

On every platform, the GNU toolchain is required to recompile the o3000 library or the JNI wrapper. Most Linux distributions already provide these tools, otherwise they can be found in the distributions repository. Under MaxOS X, download and install either XCode from the Apple App Store, or the MacPorts Collection ([4]). For MS Windows, download and install Minimalist GNU for Windows ([3]).

6.3.2 libusbx

The o3000 driver relies on libusbx (see [5]). It was developed and tested under version 1.0.14 of libusbx. Support for other versions of libusbx or the original libusb is unknown. YMMV.

The libusbx headers and libraries are already included in the driver package for your convenience. If you prefer to compile libusbx yourself, refer to the project pages and the README contained in the libusbx sources for instructions. Make sure, the resulting libraries are copied to $O3000_DIR/lib.

6.3.3 OpenCV

The o3000 JNI wrapper depends on the OpenCV library. Development and testing was performed with version 2.4.3 of that library.

The OpenCV headers and libraries are already included in the driver package for your convenience. If you prefer to compile yourself, refer to the project Wiki for instructions. Building OpenCV requires Cmake. Make sure, the resulting libraries are copied to $O3000_DIR/lib.

6.3.4 Java Development Kit

Bulding the Java demo application requires JDK6 or later, available from Oracle (see [2]).

6.4 Building the o3000 Driver

6.4.1 Dependencies

All o3000 sources depends on the o3000 and libusbx headers, located in /opt/o3000/include and the libusbx library located at /opt/o3000/lib.

6.4.2 Linux / MacOS X

Building the driver is as simple as executing:

```
>> cd $O3000_DIR
>> make -C src/o3000
```

In order to use the freshly compiled driver in the o3000 JNI wrapper, you will have to copy it to the appropriate location:

```
>> make install -C src/o3000
```

Note: this will replace the pre-compiled o3000 library!

6.4.3 Windows

Follow the instructions provided for Linux / MacOS X under 12, but execute all commands from within the MinGW shell.

6.4.4 Other build targets

Execute:

```
>> make help -C src/o3000
```

To get a list of valid make targets. For example, on Linux, you might build the o3000 documentation with:

```
>> make docu -C src/o3000
```

Note: building the documentation requires Doxygen ([7]).

6.5 Building the o3000 JNI Wrapper

6.5.1 Dependencies

o3000 JNI wrapper depends on the o3000 and OpenCV headers, located in $O3000_DIR/include and the corresponding libraries located at $O3000_DIR/lib.

6.5.2 Linux / MacOS X

Simply execute:

```
>> cd $O3000_DIR
>> make -C src/o3000jni
```

and install the resulting library with:

```
>> make install -C src/o3000jni
```

Note: this will replace the pre-compiled o3000 wrapper library!

6.5.3 Windows

Follow the instructions provided for Linux / MacOS X under 12, but execute all commands from within the MinGW shell.

6.6 Building the Demo Application

Building the demo application is quite simple. Make sure you have the complete source code (all classes), organised in a single folder, e.g. C:\Program Files\o3000\src\o3000_demo_app for Windows.

In the Windows command prompt, change to the sources directory:

```
>> cd %O3000_DIR%\src\o3000_demo_app
```

Now, you should see the folders *mvc*, *o3000*, *ctrl_structures* and *resources* in your current directory. If that is not the case, the command *javac* might not work! In addition, it is highly recommended to create a separate folder for the compiling output (java class files = binaries), for example like that:

```
>> mkdir compiled_app
```

Then, start compilation:

```
>> javac *\*.java -d .\compiled_app
```

Now copy the resources (pictures = app icon):

```
>> xcopy /T /E resources compiled_app
```

If compilation takes place in the MinGW shell, instead of inside the Windows command prompt, the commands are:

```
>> cd $O3000_DIR/src/o3000_demo_app
>> mkdir compiled_app
>> javac */*.java -d ./compiled_app
>> cp -r resources compiled_app
```

7 Driver De-Installation

7.1 Linux / MacOS X

Just remove the folder created by the installer:

```
>> rm -rf $O3000_DIR
```

7.2 Windows

Just use the option *Uninstall O30xx Camera Demo V1.xx* in the Windows start menu *Program Files / O30xx Camera Demo V1.xx.*

8 Literature and References

[1] Stettbacher Signal Processing Website, http://www.stettbacher.ch/cameras

[2] Oracle Java Website, http://java.com/en/download/index.jsp

[3] MinGW Website, http://www.mingw.org

[4] MacPorts Website, http://www.macports.org

[5] libusbx Project, http://libusbx.org

[6] OpenCV Project, http://opencv.org

[7] Doyxgen Website, http://www.doxygen.org