BIOLOGICALLY INSPIRED
ROBOTICS GROUP (BIRG)

# Sony Aibo ERS-210: Quickstart Manual

Lukas Hohl, semester 7

BIRG, Logic Systems Laboratory (LSL)
School of Computer and Communication Sciences
Swiss Federal Institute of Technology Lausanne

February 8, 2004

## Contents

# 1 Introduction

This document summarizes the basic technical and practical knowledge that was gained during the first EPFL semester project using *Sony's robotic dog Aibo ERS-210*. It is intended for other students doing a project with Aibo and provides help to get started quickly and to avoid certain pitfalls.

The starting point of every Aibo project should be the official web page [OPEN-R]. The first goal of the project will certainly be to get Aibo up and running and to install the OPEN-R SDK on a PC. For the first EPFL Aibo project an *Aibo ERS-210A equipped with an ERA-201-D1 wireless LAN card* was used. The workstation was a *Linux PC with an IEEE 802.11b compliant wireless LAN card* installed. Working with the OPEN-R SDK is more straightforward in a Linux environment, because on Windows platforms Cygwin has to be used in plus. Development is also possible with the Mac OS X operating system, but there are some limitations, i.e. Remote Processing OPEN-R will not work. [Install] provides more information on supported platforms.

The OPEN-R SDK allows the cross-compilation of programs that will be executed on Aibo. The resulting binary files have to be transfered on a special Sony Memory Stick which is then plugged into Aibo. Despite the fact that Aibo is principally sold as an entertainment robot system, it has powerful capabilities such as wireless network communication, a wide range of input and output devices such as speaker and microphone, color camera, distance sensor, acceleration sensors, various touch sensors and of course controllable joints and LEDs.

The procedure applied to set up everything thats is necessary for OPEN-R development on a Linux system is described in the following paragraphs and links to useful documents are provided. Installation on Mac OS X is very similar. Detailed instructions for the installation on a Windows system can be found in [Install]. Nevertheless, this document still contains a lot of useful information valid for all development environments.

# 2 Where to start?

As already mentioned in the introduction, the first thing to do is a visit on the *[OPEN-R] web page*. You must provide some personal information to obtain a login, but no other conditions have to be met to become a member of the OPEN-R developers community.

Go to [Download] where everything necessary on the software side (including the documentation) can be found.

# 3 Files to download

Depending on your operating system, it is not necessary to download everything in *[Download]*. There's a tiny flag next to every downloadable file. It indicates the operating system(s) the file can be used with. Before clicking on everything marked "Linux", consult the following list of files you really need:

- OPEN-R SDK Documents English [Docs]
- OPEN-R SDK [SDK]

- Sample Programs [Samples]
- Shell script for building cross development tools [Script]
- gcc source files [gcc]
- binutils source files [binutils]
- newlib source files [newlib]
- FlashUpdater for ERS-210 [Updater]

You don't need to download all files at once. In any case, start with the first one on the list (the official documentation). When you start reading the enclosed PDF files, you will soon discover, what the different files on the above list are good for.

The last file is of no importance for you when you are working with an Aibo ERS-210A or an ERS-210 that has been updated via the "220 transform kit (ERS-220E1)". If you have an ERS-210, you must update the flash memory in your ERS-210 by the means of the tool contained in this file (see [Install] for detailed instructions).

# 4 Documents to read

The file [Docs] contains all the *official documentation* available for OPEN-R in PDF format. The most important and useful files are:

- Installation Guide [Install]
- Programmer's Guide [Prog]
- Model Information for ERS-210 [Model]
- Level2 Reference Guide [Ref]
- OPEN-R Internet Protocol Version4 [IP]

# 5 Setting everything up

Start by reading *[Install] and [Prog] in parallel.* This will allow you to install the development environment and to try out the first sample programs in [Samples]. You can also start by just reading [Install] and then try out the samples that are mentioned there, but you will probably not understand much of the programs' concept without reading [Prog]. This document also explains the directory structure on the Memory Stick. [Install] and [Prog] are easy to understand in general. It follows a list of a few additions that might be useful:

- Skip paragraphs mentioning installation files you haven't downloaded.
- The *Memory Stick reader/writer* will perfectly work without any additional drivers on Mac OS X. For Windows systems there is a driver CD coming with the reader/writer. The following passage was taken from [BBS] and explains the usage of the Memory Stick reader/writer on a Linux system:

```
MOUNTING THE MEMORYSTICK

It is recommended to plug the usb reader into the USB port
before turn on the power of the computer, although most of
the linux distributions support hotplug technology for USB
devices. Make sure the Sony AIBO Memorystick is correctly
plugged into the usb reader.

After log into the bash shell, make sure that the usb reader
is recognized by the system. You can check it by typing:

$ cat /proc/bus/usb/devices | grep S:

and you will probably see something like this:

S: Product=USB OHCI Root Hub
S: SerialNumber=ccacb000
S: Manufacturer=SCM Microsystems Inc.
S: Product=eUSB MemoryStick Reader
S: SerialNumber=0000000011CF

Now we can mount the memorystick into the system by the
following steps:

1. Create a empty directory to which be mounted,
   ex. /mnt/usb:
   $ mkdir /mnt/usb

2. And then mount the memorystick
   (You must have supervisor privilege first):
   $ mount /dev/sda1 /mnt/usb

USING THE MEMORYSTICK

After mount the memorystick to the specified directory
(/mnt/usb), you can use it just like other storage device,
such as harddrive. Most of the filesystem command, such as
ls, mkdir, cat, etc., can be applied.

When you "ls" the memorystick for the firsttime, you will
find a file names memstick.ind already existing in the
memorystick. DO NOT delete it, since this memorystick
is to be used by the Sony AIBO robotic dog.
```

In the following explanations it is assumed that the Memory Stick is accessible at `/mnt/usb`.

- If you do not plan to use Remote Processing OPEN-R, it is sufficient to copy the directory `/usr/local/OPEN_R_SDK/OPEN_R/MS/WCONSOLE/memprot/OPEN-R` to the

Memory Stick once and for all (`OPEN-R` has to be at the top level of the Memory Stick, e.g `cp -r OPEN-R /mnt/usb`). This configuration enables the *wireless console and memory protection* and most samples will perfectly work with it. Only use other configurations when they are needed by the samples you want to try out.

- Aibo's *wireless LAN configuration* file `WLANCONF.TXT` is located in `/mnt/usb/OPEN-R/SYSTEM/CONF/` (was copied with the configuration).
  The easiest way to get wireless communication between Aibo and your PC is to do only minor changes in the configuration file and to set up your PC according to the settings in `WLANCONF.TXT`. In that case you define a net consisting only of Aibo and your PC by default. You only have to modify Aibo's `IP_GATEWAY` entry and set it to Aibo's IP. You might have to experiment with different settings of `APMODE`. The default setting did perfectly work in our case.
  Note: `#` comments out a line.

- The running of (non Remote Processing OPEN-R) *sample programs* always works the same way: Change the working directory to the directory `SampleName`. Type `make install`. This will build the source code in the directory `SampleName/SampleName`. All kinds of intermediate files will be generated there. This process can also be started with a simple call to `make`. `make install` however causes the generation of the binary files that will actually be executed on Aibo. These files are built using the intermediate files and are then placed into `SampleName/MS/OPEN-R/MW/OBJS`. If a sample program builds and includes other samples (e.g. `PowerMonitor` is built with every other sample), the final binary files will only be placed in the directory of the sample where the command was given. The intermediate files are created in the directory of the included sample. It is the `SampleName/MS/OPEN-R` directory than then has to be copied to the Memory Stick. This one must not replace the already existing directory (copied with the configuration)! The contents of the two directories need to be merged. This is automatically ensured by giving the command `cp -r MS/OPEN-R /mnt/usb` when working in `SampleName`.

- It is possible to have the binary files of *multiple sample programs* on the Memory Stick. Only those of the last one copied to the Memory Stick will be executed because the `OBJECT.CFG` file is replaced each time. But be sure to remove the `CONNECT.CFG` file of any previous sample if the new one doesn't have it's own instance of this file. Both files can be found in `OPEN-R/MW/CONF`.

# 6 Understanding OPEN-R

Despite the fact that [Ref] is mainly a reference guide, it is good for the overall understanding of OPEN-R to have a first look at this document very early. Read mainly the *descriptions of classes and methods*. This will greatly improve your comprehension of the sample programs. The same is true for [IP]. Read the general chapters and those treating the *network services* you find useful.

A main source of information are the [Samples]. [Info] offers links to *training courses and tutorials* which often contain detailed explanations of certain samples. Here is a list of *samples* that have proven their usefulness for the indicated topics (the samples are presented in order of increasing difficulty):

**ObjectComm** inter-object communication

**PowerMonitor** robot status, boot conditions, shutdown

**BlinkingLED** LED and ear control

**MovingHead** joint control

**MovingLegs** joint control

**SensorObserver** sensor reading

**LMasterRSlave** sensor reading, joint control

**EchoServer** TCP/IP communication with PC
(see `echo_client/echo_client.c` for the PC client side)

**TinyFTPD** file transfer between Aibo and PC using FTP protocol (TCP/IP)
`util/mstreeput` did not work properly (following instructions in [FAQ]).

**MoNet** playback of MTN files

# 7 Writing your own programs

The best way to obtain a first result obtain relatively fast is to *adapt one of the sample programs* to your needs. Several [Samples] can be combined and exchange information using inter-object communication. The basic software designing rule is: *One object per concurrent task on Aibo.* All the samples use a finite state automaton to describe the working cycle of an object. When an object is commanding others, the Ready method contains an important part of the automaton. This seems to be the best approach to deal with OPEN-R's subject-observer architecture.

To *combine multiple adapted samples* to a new product, copy the directories of the samples you want to use to a new directory (the main directory of your product). Include the `PowerMonitor` sample in any case. Choose a main object for your product. Its `Makefile` in the directory `MainSampleName` must be adapted to compile the other objects. Just add the other samples' directory names as it has already be done by Sony for `PowerMonitor`. The modified file will then build all the samples in the specified directories and put the final binary files into `MainSampleName/MS/OPEN-R/MW/OBJS`. `OBJECT.CFG` and `CONNECT.CFG` in `MainSampleName/MS/OPEN-R/MW/CONF` must be modified accordingly (add the content of the corresponding files of the added samples).

Soon you will probably want to *change the name of a sample program* (let's say to `SampleName`). First of all you must change all the filenames and directory names (do `make clean` to remove generated files first, respect initial upper and lower case when doing the renaming). The two `Makefile` in `MainSampleName` and `SampleName` must be adapted (change the name of the component). Replace the old name everywhere in the `Makefile` in `SampleName/SampleName`

(again respect initial upper and lower case). The name of the final binary can have at most 8 characters (plus `.BIN`) and must be upper case. `OBJECT.CFG` and `CONNECT.CFG` in `MainSampleName/MS/OPEN-R/MW/CONF` must be modified according to the name changes. To keep consistency, also modify `OBJECT.CFG` and `CONNECT.CFG` in `SampleName/MS/OPEN-R/MW/CONF`. The last two files to modify are `stub.cfg` and `sampleName.ocf` in `SampleName/SampleName`. This last file could remain unchanged, the name change just guarantees that the correct object name will be displayed in the wireless console when Aibo boots.

*Additional source code files* are best places into `SampleName/SampleName`. By default, all source code files with the correct name ending are compiled, so the `Makefile` in the same directory only needs minor changes: For every added `filename.cc` you just have to add `filename.o` in the line `sampleName.bin:`. This guarantees correct linking.

# 8 Hints

- To have a good overview of Aibo's capabilities (physical limits, return values of sensors, output devices) consult [Model].
- Doing a `telnet` on Aibo will significantly slow down the overall network performance when a lot of characters are printed to the wireless console. Even when the printouts are not displayed on the PC by `telnet` and do not affect the network performance, execution on Aibo will be slower when large amounts of characters are printed to the console.
- The most common error on Aibo is "segmentation fault" (not displayed on the wireless console). This causes Aibo to shut down immediately. Use `OSYSPRINT` for debugging, but often not all messages up to the point of failure are displayed on the wireless console. Test your programs often to track down errors early. No experience with other low level debugging techniques was made.
- It is good practice to always execute `make clean` after any modification in any of the `*.h` files. Often an untraceable segmentation fault will disappear afterwards.
- When Aibo takes a long time to shut down after pushing the pause button, plug in the battery charger. This will cause Aibo to crash (when using the default `PowerMonitor`).
- No indication on speed and acceleration limits is given in the documentation. In [Notes] there is a direct link to a [FAQ] entry where at least angular speed limits for Aibo's joints are listed:

```
The limit of angle speed for ERS-210

Primitive Locator         angle speed limit
                  (degree/frame  1frame=16msec)
-------------------------------------------------
/r1/c1-Joint2:j1          1.93
/r1/c1/c2-Joint2:j2       2.76
```

```
/r1/c1/c2/c3-Joint2:j3    2.76
/r1/c1/c2/c3/c4-Joint2:j4 4.01
/r2/c1-Joint2:j1          2.58
/r2/c1/c2-Joint2:j2       2.29
/r2/c1/c2/c3-Joint2:j3    2.60
/r3/c1-Joint2:j1          2.58
/r3/c1/c2-Joint2:j2       2.29
/r3/c1/c2/c3-Joint2:j3    2.60
/r4/c1-Joint2:j1          2.58
/r4/c1/c2-Joint2:j2       2.29
/r4/c1/c2/c3-Joint2:j3    2.60
/r5/c1-Joint2:j1          2.58
/r5/c1/c2-Joint2:j2       2.29
/r5/c1/c2/c3-Joint2:j3    2.60
/r6/c1-Joint2:j1          4.10
/r6/c2-Joint2:j2          4.10
-----------------------------------------------------

NOTE: The motion file (*.mtn) which is created by AIBO
Master Studio has 1 frame = 16msec. But minimum interval of
motion of AIBO(ERS-210/220) is 8msec.
(Time of one frame that AIBO recognizes actually is 8msec)
```

- Before setting joint gains using OPENR::SetJointGain, a valid CommandVector must have been passed to OVirtualRobotComm. In the sample programs this is done using AdjustDiffJointValue which fills a CommandVector with the current position. If no CommandVector is provided, Aibo will jump to a standing position as soon as OPENR::SetJointGain is called.
- Aibo's MIPS processor is little-endian. This has to be considered when transferring data to systems working in big-endian mode.

# References


[OPEN-R] Official Sony OPEN-R web page:
`http://openr.aibo.com/`
Direct link to the English web page:
`http://openr.aibo.com/openr/eng/index.php4`

[BBS] Section "Bulletin Board" on [OPEN-R]

[Download] Section "Download" on [OPEN-R]

[FAQ] Section "Frequently Asked Questions (FAQ)" on [OPEN-R]

[Info] Section "Information of OPEN-R SDK" on [OPEN-R]

[Notes] Section "Important notes when using the OPEN-R SDK" on [OPEN-R]

[Docs] File "OPEN-R SDK Documents English":
`OPEN_R_SDK-docE-XXX.tar.gz`[1] in [Download]

[Install] Document "Installation Guide":
`InstallationGuide_E.pdf` in [Docs]

[Prog] Document "Programmer's Guide":
`ProgrammersGuide_E.pdf` in [Docs]

[Model] Document "Model Information for ERS-210":
`ModelInformation_210_E.pdf` in [Docs]

[Ref] Document "Level2 Reference Guide":
`Level2ReferenceGuide_E.pdf` in [Docs]

[IP] Document "OPEN-R Internet Protocol Version4":
`InternetProtocolVersion4_E.pdf` in [Docs]

[SDK] File "OPEN-R SDK":
`OPEN_R_SDK-XXX.tar.gz` in [Download]

[Samples] File "Sample Programs":
`OPEN_R_SDK-sample-XXX.tar.gz` in [Download]

[Script] File "Shell script for building cross development tools":
`build-devtools-XXX.sh` in [Download]

[gcc] File "gcc source files":
`gcc-XXX.tar.gz` in [Download]

[binutils] File "binutils source files":
`binutils-XXX.tar.gz` in [Download]

[newlib] File "newlib source files":
`newlib-XXX.tar.gz` in [Download]

[Updater] File "FlashUpdater for ERS-210":
`upgrade-OPEN_R-XXX.tar.gz` in [Download]


[1] "XXX" in a filename stands for the current version number of the file (not the same for all files)