BCI

Brain Computer Interface

for alternative input

Gianluca Moro

Revision 20130512

Contents

Chapter 1

Introduction

BCI, or Brain Computer Interface is a research field which try to verify and hopefully realize an interface to a computer controlled just by the user mind. This is technically feasible, and it has already been done, but mainly with professional EEG. Now the tecnology is giving us low cost devices whose quality is sufficient to use this tecnique with an affordable investment.

The possible application are both for users which need to use hands for other purposes, or to integrate the standar interface to the PC, or persons unable to use their hands, or even other parts of their body. In this case the device could notably increase the quality of life of the person.

The technology is mainly based on EEG, and the idea is practically feasible in the recent years with the availability of portable low cost devices. The possible hardware is:

- http://www.neurosky.com/ a chip used by various producer, such as PlxWave or some "mind games" available from Amazon;
- http://www.plxwave.com/ a low cost device, which offer interface APIs for IOS and Android;
- http://www.emotiv.com/ A low cost - not as low as PlxWave, but with more sensors.

Let's see the "Big Picture" (from the technical point of view):

- an acquisition device, an headset such as Emotiv Epoc
- EEG data acquisition
- data is sent via Bluetooth (preferred) or USB

- data elaboration
- data extrapolation of some meaningful signals to be interpreted by the computer/mobile as inputs, such as mouse simulation

And the "Big Picture" (from the user point of view):

- syndromes such as Locked in syndrome (one the most critical situations) http://en.wikipedia.org/wiki/Locked-in_syndrome let the patient conscious but unable to move almost any muscle. This can be an opportunity to give a new life to such a patient.
- a user with the ability to control 1 signal, can communicate with the external world ... the more the device can understand from patient's EEG, the quicker will the communication run.

Chapter 2

Ongoing research

"The Berlin brain–computer interface: non-medical uses of BCI technology", by Benjamin Blankertz, MichaelTangermann, CarmenVidaurre, Siamac Fazli, Claudia Sannelli, Stefan Haufe, Cecilia Maeder, Lenny Ramsey, Irene Sturm, Gabriel Curio and Klaus-Robert Müller (available on http://www.frontiersin.org/neuroprosthetics/10.3389/fnins.2010.00198/full is an overview with many references about the various uses of BCI: the overview includes:

- the importance of dry electrodes to make EEG easily usable, with references to studies about their measure quality
- theproblem of training and the possibilities to avoid it, for example with large pattern databases
- the use of BCI to understand the mental state, for example the attention level, the incoming sleep
- BCI in entertainment, with examples of use in "fast reaction" games, such as pinball machines

The article **"NeuroPhone: Brain-Mobile Phone Interface using a Wireless EEG Headset"**, by Andrew T. Campbell, Tanzeem Choudhury, Shaohan Hu, Hong Lu, Matthew K. Mukerjee, Mashfiqui Rabbi, and Rajeev D. S. Raizada (http://sensorlab.cs.dartmouth.edu/pubs/neurophone.pdf)is an overview of an implementation of P300 on iPhone using the Emotiv Epoc as headset: the implementation uses a Windows PC as data bridge from the headset to the iPhone.

A description of the OpenVibe framework in present in **"OpenViBE: An Open-Source Software Platform to Design, Test and Use Brain-**

Computer Interfaces in Real and Virtual Environments" availabe at http://www.hal.inserm.fr/docs/00/47/71/53/PDF/Renard_et_al_2010_draft.pdf

The thesis **"Adaptive Computer Interfaces"** by D. Ward is a 2001 work (http://www.inference.phy.cam.ac.uk/djw30/papers/thesis.pdf), submitted in candidature for the degree of Doctor of Philosophy, at the University of Cambridge, and exploring predictive tecniques to mare data input faster, in particular, it is exposed the research about the Dasher input system (http://www.inference.phy.cam.ac.uk/dasher/).

A tactile P300 brain-computer interface http://www.frontiersin.org/neuroprosthetics/10 show the possibility to record P300 potentials evoked by tactile stimulation.

A simple format for exchange of digitized polygraphic recordings: the paper proposing EDF format.

A "Turing Test" and BCI for locked-in children and adults a proposed test to use evoked potential with childen and locked in persons. A stimulation based on YES/NO (as a variant of different sounds' types)

Open vocabulary language modeling for binary response typing interfaces by Brian Roark: predictive text input.

Chapter 3

Emotiv Epoc

The Emotiv is the first device we will exploit: is has an affordable price (from 300$ - just the device - to 750$ for a Research Development Kit) and makes 14 data channels available. Here an interesting demo on the device: http://www.ted.com/talks/lang/eng/tan_le_a_headset_that_reads_your_brainwaves.html

There are other devices, such as the PlxWave: the idea is to design the software in such a way as it will be easy to change acquiring device, both to be able to use future headset, when they will be available, and to compare the various devices, or even to just give a more economically affordable solution, if available, and if the reached control capacity will be sufficient.

3.1 Technical infos

The Emotiv Epoc (with the Research SDK) gives the following features (from http://emotiv.com):

- 14 channel (plus CMS/DRL references, P3/P4 locations) high resolution, neuro-signal acquisition and processing wireless neuroheadset. Channel names based on the International 10-20 locations are: AF3, F7, F3, FC5, T7, P7, O1, O2, P8, T8, FC6, F4, F8, AF4 (see http://www.bem.fi/book/13/13.htm)

This is what the hardware gives, but, depending on the official SDK used, not everything can be available. The research SDK gives:

- real-time display of the Emotiv headset data stream, including EEG, contact quality, FFT, gyro (if fitted, custom option), wireless packet acquisition/loss display, marker events, headset battery level

- record and replay files in binary EEGLAB format1. Command line file converter included to produce .csv format.

The Emotiv communicates with PC via a ultra low power Bluetooth interface (to use less battery): there can be some communication problems if the headset and the receiver are not in the optimal position (the nearest, the best :-) http://www.emotiv.com/forum/messages/forum4/topic484/message2659/?phrase_id=158643#messa

3.1.1 From Emotiv forum

```
A little about the electronics in the headset?
08.04.11 9:17 PM

Hi Alon, in case you don't have access to the Research forum, I
collected a couple of references and repeat them below. Otherwise,
search for filtering, IIR, CMS, impedance, common mode etc to get all
forum posts on this topic.

The referencing system defines the electrical ground point for the
measurement of all the other sensors - we effectively measure the
voltage difference between the left-hand reference point and every
other sensor. The right-hand reference is a standard feed-forward
reference compensation electrode which allows the headset electronics
to ride on top of changes in body potential - for example electrical
pickup from lights, power circuits, transformers and so on, which
change drastically as you move around the room. The left-hand
reference is called CMS and the right-hand reference is called DRL
(driven right leg smile:)) after the original use in high-resolution
ECG systems where it was traditionally attached to the right leg of
the patient. Two-point referencing is very common in expensive medical
grade EEG and ECG systems. We use it on the EPOC becuase it seriously
cuts out the noise (by about 55dB at mains frequencies for the
tech-heads out there). We don't actually measure those points
individually and send the data to the PC. CMS would be pretty boring -
the difference between CMS and itself is a pretty dull looking signal.
We also added a little extra (patented) trick to the conventional DRL
circuit - we use it to help measure the contact quality at every other
sensor by adding a smal additional modulation to the feedforward
signal, and reading the magnitude of that signal back from each other
channel. This is converted into the nice green/yellow/orange/red/black
contact quality map. If nobody is receiving, the references are shown
```

as black. If any other channel is receiving, they show as green.

We have applied some filtering in the hardware and firmware to remove mains frequency interference. The signals are collected through a C-R high-pass hardawre filter (0.16Hz cutoff), preamplified and low-pass filtered at 83Hz cutoff. Data is also processed in the headset as follows: raw ADC collection rate is 2048 /sec / channel. This data is filtered using a 5th-order sinc filter to notch out 50Hz and 60Hz, low-pass filtered and down-sampled to 128/sec/channel to eliminate mains harmonics. No further processsing is done - the effective bandwidth is 0.16-43Hz

about the algorithm of Emotiv
10.07.11 8:23 PM
Hi Zhang,

I can't send you any more details than I can publish on the user forum, which is as follows: Each detection suite is slightly different.

Some facial expressions (blinks, winks, eye movements) depend on pattern matching in real time, for example blinks are characterised by coherent rising pulse shapes on several frontal sensors which correspond to a specific wave shape and risetime, followed by a fall. These signals are balanced against rear channels which must not show the same trace shapes. The blink is flagged after the signals have passed through a specific profile matching algorithm, approximately 100ms after commencement of the blink. The sensitivity slider adjusts a threshold for the fitting algorithm and may also scale the signals to better match the profile.

Other facial expressions depend on the distribution and relative strength of several frequency bands across many channels. These signals are processed to yield specific features by analysing a trailing sample of data (allowing frequency extraction) and are passed to a classifier every 0.25 seconds.

Affectiv detections also depend on the distribution and relative intensity of specific frequency bands, as well as some custom features based on fractal signal analysis. These are passed to a classifier system to detect specific deflections, low-pass filtered

```
and the outputs are self-scaled to adjust to each user's range of
emotion.

Cognitiv detections are trained in-situ. Neutral data and the actions
in question are trained (possibly repeatedly) and the training data
is segmented into short data epochs. We extract a large number of
features including spectral and other measurements and apply a
feature reduction method to provide a parsimonious model which is
used to clasify states in real time using the reduced feature
set. The features and sensors chosen for Cognitiv are unique to each
user signature. We also apply some corrections to the background
state to allow for different fittings, moods, contact quality and
detected noise patterns.
```

3.2 Software interface

Emotiv Epoc sends to the host a stream of encrypted data: the interface is not open, but some reverse engineering work has been accomplished to use it: information can be found at Emokit project.

By the way, in the same discussion there's a reference to the following paper: http://sensorlab.cs.dartmouth.edu/pubs/neurophone.pdf, saying: "The headset transmits encrypted data wirelessly to a Windows-based machine; the wireless chip is proprietary and operates in the same frequency range as 802.11 (2.4Ghz)". This means that the receiver is not a standard Bluetooth device (i.e. we cannot use directly a smartphone do receive data) but we need the provided Emotiv dongle.

The headset samples all channels at 128Hz, each sample being a 14 bit value corresponding to the voltage of a single electrode.

3.3 Emokit

A project started by Cody Brocious, http://daeken.com/emokit-hacking-the-emotiv-epoc-brain-computer-0, available at http://github.com/qdot/emokit. This project is now included in http://www.openyou.org/libs/.

Emokit has C and Python interfaces which allow to access raw EEG data from the Emotiv EPOC on Windows, Linux, and OS X.

Emotiv comunicates using a USB dongle: it's a USB device with VID=21A1, PID=0001 (note: from walking through the device enum code in the EDK, it seems that PID=0002 might be the development headset, but that's to-

tally unverified). It presents two HID interfaces, “EPOC BCI” and “Brain Waves”. Reading data off the “Brain Waves” interface gives you reports of 32 bytes at a time; “EPOC BCI” is not clear what is.

The data is crypted, with 128-bit AES in ECB mode, block size of 16 bytes: the found key was: 31003554381037423100354838003750.

The first byte of each report is a counter that goes from 0-127 then to 233, then cycles back to 0. The X coord from the gyro is byte 29 and the Y coord is byte 30. The EPOC has some sort of logic in it to reset the gyro baseline levels, the baseline generally (not perfect) is roughly 102 for X and 204 for Y.

If you assume that each sensor is represented by 2 bytes of data, that gives us 28 bytes for sensor data. 32 - 28 == 4, so what's the extra byte? Looking at byte 15, it's pretty clear that it's (almost) always zero – the only time it's non-zero is the very first report from the device.

According to some data on the research SDK, there're 2 bits per sensor that give you the signal quality (0=none, 1=very poor, 2=poor, 3=decent, 4=good, 5=very good).

Packet structure is:

```
struct epoc_contact_quality {
    char F3, FC6, P7, T8, F7, F8, T7, P8, AF4, F4, AF3, O2, O1, FC5;
};

struct epoc_frame {
    int F3, FC6, P7, T8, F7, F8, T7, P8, AF4, F4, AF3, O2, O1, FC5;
    struct epoc_contact_quality cq;
    char gyroX, gyroY;
    char battery;
};
```

3.4 OpenVibe

From EEG raw data to meaningful information we need some elaboration:
OpenVibe http://openvibe.inria.fr/ is a general Brain Computer Interface.
It's not the only one; the paper which describes it http://www.hal.inserm.fr/docs/00/47/71/53/PD
reports some link about other interfaces too.

3.5 Applications

http://www.nanotechgalaxy.com/braintalk/ An application to simulate a keyboard.

3.6 Roadmap

Preliminary work:

- **acquire EE (Emotiv Epoc) data on PC via Emokit:** should be as easy as running the demo ... but sometimes there are troubles with the decryption key!
- **acquire EE (Emotiv Epoc) data on Android via Emokit:** this needs a porting to Android platform, in particular, verify the Android API for Bluetooth Input.
- **test EE and OpenVibe on Windows/Linux with official SDK:** this should be a plain task. Emotiv is officially supported on Windows, while on Linux there's a beta SDK (probably it can be asked by a SKD license holder)
- **write an acquisition server/driver for OpenVibe on Linux using Emokit:** the server building is documented. Interfacing Emokit to the server should be a work of "data structure adaptation"
- **write an acquisition server/driver for OpenVibe on Android using Emokit:** the porting of the previous work on Android.
- **run OpenVibe on Android (?):** OpenViBE is an analizying tool. Is it better to port OpenViBE on Android or take from OpenViBE the relavant algorithms and implement on Android? (Probably this last option is better!)
- **identify 2/3 signals which a user can control (x/y axis, click):** this can be done on OpenViBE on Linux/Windows. When we have meaningful data, we can try to duplicate them on Android
- **interface those signal to I/O methods:** to simulate mouse/joystick moving.
- **USE I/O FOR MIND CONTROLLED INPUT!:** and now ... the work can start! use the previous software to realize something!

Application work:

- **I/O test:** an application that moves a ball around (to test ... and to show as demo!)
- **Porting of Dasher:** http://www.inference.phy.cam.ac.uk/dasher/DasherSummary.html an efficient alternative input method, to Android
- **Interface Dasher to mind controlled I/O**

Specific for Android:

- Raw data acquisition
  - Bluetooth I/O API
  - USB I/O API (should not be needed - Emotiv send data via Bluetooth with a USB dongle: it should be possible to read directly the Bluetooth stream)
  - porting of Emokit to decode data
  - implement some API to access raw data
  - implement a data server (could be the OpenViBE acquisition server). This is an alternative to the previous point: to verify which is the best solution
  - while doing this, do think about having the same tools (API or server or both) to acquire data from different devices (Emotiv Epoc, PlxWave, Dummy1)
  - implement a "store raw data" and a "read stored data" to be used thru' a DummyDevice (for test of upper layer)
- Raw data elaboration. OpenViBE is a EEG data elaboration tools, but quite a big framework to implement on Android. The idea could be to test the data elaboration on PC and to implement in Android only the relevant parts. OpenViBE core is a data analysis tool, based on IT++: it could be interesting implement IT++ (or part of it) on Android. IT++ has other dependencies such as FFTW, BLAS, LAPACK.

  The package to port to Android are the following: these steps are to verify (i.e. verify if they are really necessary)

  - FFTW
  - BLAS

  - LAPACK

- Output of relevant parameter: the final result should be some (2/3/4/5) parameters. The ideal would be to have

  - 2 analogic signal for x/y axis
  - 1 click signal to select

- interface this parameter to Android I/O structure to simulate a joystick

Chapter 4

Hands on tests!

4.1 Hardware connection

The initial test on the Emotiv Epoc are quite interesting:

- the comunication channel is NOT Bluetooth: it uses a http://www.nordicsemi.com/ chip - a ultra low power device implementing a wireless communication on 2.4GHz. From the practical point of view, the Emotiv Epoc cannot be accessed directly by a smartphone via Bluetooth
- the comunication Epoc/smartphone could be done:
  - using a dedicated device from Emotiv: they seem interested in selling such a device, but at the moment nothing is available.
  - using a wireless hub: http://http://www.iogear.com/product/GUWH104KIT/. This could be used in the configuration headset - dedicated usb dongle - wireless hub - smarthphone. The problem is the driver, not available in smartphone.
  - using a PC as bridge (in the PC there would be the EPOC driver)
  - using an embedded solution: a small factor/mini/micro PC with a USB host, a Bluetooth dongle and a stripped down Linux. There's the Epoc driver problem.

4.2 Primary data reading

Data sent by the Epoc is crypted, but it is present a software to read it: http://github.com/qdot/emokit

It is available a Python software to read the primary data, using 3 possible decrypting keys: the correct one for the available consumer key under test is another one, extra1_key

```
consumer_key = '\x31\x00\x35\x54\x38\x10\x37\x42\x31\x00\x35\x48\x38\x00\x37\x50'
research_key = '\x31\x00\x39\x54\x38\x10\x37\x42\x31\x00\x39\x48\x38\x00\x37\x50'
special_key = '\x31\x00\x35\x48\x31\x00\x35\x54\x38\x10\x37\x42\x38\x00\x37\x50'
extra1_key = '\x4D\x00\x47\x54\x38\x10\x31\x42\x4D\x00\x47\x48\x38\x00\x31\x50'
```

found at http://github.com/qdot/emokit/issues/4.

The procedure used to get that key was the following (as reported in the forum):

```
Here is detailed howto for figuring my consumer key:

- Download OllyDbg http://www.ollydbg.de/
- Download SnD Crypto Scanner
  (http://www.woodmann.com/collaborative/tools/
    index.php/SnD_Crypto_Scanner_%28Olly/Immunity_Plugin%29)
  and copy OllyCScan to Olly directory (version 0.4b)
- Run OllyDbg
- Open F3 EmotivControlPanel.exe (do not run debugging yet)
- Run OllyDbg plugin -> SnD Crypto Scanner
- Perform Scan
- MD5, Rijndael (Sig 1) and Rijndael (Sig 2) should be found
- Keep SnD Crypto Scanner Window open
- Run debugging of EmotivControlPanel.exe (F9)
- EmotivControlPanel should completely start, display window
  with brain and be receiving data
- no need to have headset properly positioned
- Switch to SnDCrypto Scanner window
- Click on Rijndael (Sig 2) then on line in lower window
  (Signature e1f898...), then click Set Hardware Breakpoint
- Debugger should almost immediately break at some instruction
  (e.g., SHL EDX,8)
- Scroll up until sequence of INT3 opcodes is found
- Put breakpoint (F2) to first instruction below INT3 sequence -
  this is the start of function manipulating key
  we now have begin of the function manipulating with epoc key
  you may remove hardware breakpoints now (will break on normal
  breakpoint at the beginning of this function)
- Resume debugging (F9)
  should break almost immediately
```

```
- step over (F8) unless first CALL instruction is found
- step into (F7)
  this function is strange, doing memset on NULL param and free
  (NOT what we are searching for)
- continue (F8) to second CALL instruction
- step into (F7), you should see memcpy and memset calls
  disassembled in OllyDbg window
- step to part where memcpy function arguments are prepared
  (PUSH instruction)
- n should be equal to 0x10 (16B) - number of bytes in eopc key
- src should point to buffer with epoc key
- read buffer address (OllyDbg will show that or see value of
  corresponding PUSH parameter)
- switch to Memory dump window, RClick->Goto->Expression
- type buffer address (or parameter of PUSH)
- read epoc key (16 bytes) from Memory dump window
  {0x4D,0x00,0x47,0x54,0x38,0x10,0x31,0x42,0x4D,0x00,
   0x47,0x48,0x38,0x00,0x31,0x50};
- Insert your headset key into Emokit software
  (https://github.com/qdot/emokit)
- Change last parameter of epoc_open to 1 instead of 0
  (epoc_open(d, EPOC_VID, EPOC_PID, 1)) in main()
```

The used program is almost the standard Emokit: just added the new decryption key on `emotiv.py`, while the test program has been rewritten as `rawread.py`:

```
#!/usr/bin/python

import sys, time, logging
from emotiv import Emotiv

emotiv = None

def main(debug=False):
print 'Starting MAIN'
while True:
for packet in emotiv.dequeue():
print '{0:4d},{2:4d},{3:4d},{4:4d},{5:4d},\
{6:4d},{7:4d},{8:4d},{9:4d},{10:4d},{11:4d},{12:4d},{13:4d},{14:4d},\
{15:4d},{16:4d},{17:4d};'.format(packet.counter, packet.sync,
 packet.gyroX, packet.gyroY,
 packet.F3[0], packet.FC6[0],
```

```
 packet.P7[0], packet.T8[0],
 packet.F7[0], packet.F8[0],
 packet.T7[0], packet.P8[0],
 packet.AF4[0], packet.F4[0],
 packet.AF3[0], packet.O2[0],
 packet.O1[0], packet.FC5[0])
#sys.stdout.flush()
#time.sleep(1.0/128.0)

try:
logger = logging.getLogger('emotiv')
logger.setLevel(logging.INFO)
log_handler = logging.StreamHandler()
logger.addHandler(log_handler)
emotiv = Emotiv()
main(*sys.argv[1:])
finally:
if emotiv:
emotiv.close()
```

To use it on MacOSX:, first of all the Linux `lsusb` command is the following:

```
~ $ system_profiler SPUSBDataType
...
        EPOC BCI:

          Product ID: 0x0001
          Vendor ID: 0x21a1
          Version:  0.03
          Serial Number: SN201105230918GM
          Speed: Up to 12 Mb/sec
          Manufacturer: Emotiv Systems Inc.
          Location ID: 0x04100000 / 4
          Current Available (mA): 500
          Current Required (mA): 100
```

then the python program expects to find a `/dev/hidraw?` device: it is a HID raw driver (http://http://lxr.free-electrons.com/source/drivers/hid/hidraw.c) which:

```
/*
```

```
 * HID raw devices, giving access to raw HID events.
 *
 * In comparison to hiddev, this device does not process the
 * hid events at all (no parsing, no lookups). This lets applications
 * to work on raw hid events as they want to, and avoids a need to
 * use a transport-specific userspace libhid/libusb libraries.
 *
 *  Copyright (c) 2007 Jiri Kosina
 */
```

4.3 An open server

Emotiv Epoc data could be used from various software, such as OpenVIBE: it could be interesting to prepare a daemon sending data on a socket.

At the moment, for testing purposes, we can use the python program, sending the data in plain ASCII (values separated by ',', line terminated by ';') to standard output, combined with the nc tool (netcat) to pipe the data to a socket.

This app can be used to read data from a socket: http://giammy.com/eegview

4.4 OpenVIBE test

First considerations:

- OpenVIBE must be compiled from sources for Linux
- Epoc driver is available only for Windows, needs the Emotiv SDK and must be compiled
- OpenVIBE can read data from a socket :-)

Compilation of OpenVIBE on Ubuntu 11.04:

- linux-install_dependencies
- linux-init_env_command
- linux-build

4.5 Test: Evoked potential

The test (Audio evoked potential) has been accomplished on 2011.10.14, with the Emotiv Epoc headset. The setup includes:

- patient with Emotiv Epoc
- acquisition on a Linux box via a Python program (variation of the one included in the EmoKIT)

The acquisition runs continuously while the software emits 2 100ms beeps (200Hz and 4000Hz frequency), every 1-3 seconds (the variation of the interval length is random).

Whenever a sound must be emitted, the low frequency one is chosen with probability 0.2.

The subject was sitting on his chair, closed eyes, mentally counting the low frequency beeps.

The program is the following:

```
#!/usr/bin/python

import sys, time, logging, random
from emotiv import Emotiv

try:
    import winsound
except ImportError:
    import os
    def playSound(frequency,duration):
        #apt-get install beep
        os.system('beep -f %s -l %s &' % (frequency,duration))
else:
    def playSound(frequency,duration):
        winsound.Beep(frequency,duration)

#Freq = 2500 # Set Frequency To 2500 Hertz
#Dur = 1000 # Set Duration To 1000 ms == 1 second
#playSound(Freq,Dur)

emotiv = None

#
# Configure test
```

```
#

hz = 128    # device acquisition frequency as integer
hzf = 128.0 # ... and the same as float

# frequency in Hz
lowFreq = 200      # frequency of low frequency sound
highFreq = 4000    # frequency of high frequency sound

# duration in ms of the two sounds
lowFreqSoundDurationMs = 100
highFreqSoundDurationMs = 100

# interval between 2 sound (a random value included in the range)
timeBetweenSoundMinMs = 1000
timeBetweenSoundMaxMs = 3000

# percent: 0.2 -> 20% low frequency sounds, 80% high frequency sounds
lowFreqSoundPercent = 0.2

#
# End of configure
#

timeBetweenSoundDeltaMs = timeBetweenSoundMaxMs - timeBetweenSoundMinMs

#
# Real end of Configure
#

showQuality = 0

currentFrequency = 0
currentTime = 0.0
currentSample = 0
nextSoundSample = 1*hz # first sound after 1s
stopSoundSample = 0

def isLowFreq():
    global lowFreqSoundPercent
    if random.random() < lowFreqSoundPercent:
        return True
    else:
```

```
        return False

def ms2samples(ms):
    global hzf
    return int(hzf * float(ms) / 1000.0)

def samples2ms(s):
    global hzf
    return int(1000.0 * float(s) / hzf)

def checkSound():
    global currentSample
    global nextSoundSample
    global currentFrequency
    global lowFreq
    global highFreq
    global lowFreqSoundDurationMs
    global highFreqSoundDurationMs
    global timeBetweenSoundMinMs
    global timeBetweenSoundDeltaMs
    global stopSoundSample

    if currentSample > nextSoundSample:
        if isLowFreq():
            currentFrequency = lowFreq
            soundDurationMs = lowFreqSoundDurationMs
        else:
            currentFrequency = highFreq
            soundDurationMs = highFreqSoundDurationMs
        playSound(currentFrequency, soundDurationMs)
        nextSoundSample = currentSample +
          ms2samples((timeBetweenSoundMinMs +
            random.randint(0, timeBetweenSoundDeltaMs)))
        stopSoundSample = currentSample + ms2samples(soundDurationMs)
    if currentSample > stopSoundSample:
        currentFrequency = 0

def main(debug=False):
    global currentSample
    global currentTime
    global currentFrequency
    global showQuality
    global hzf
```

```
    print 'Starting MAIN'
    if showQuality == 0:
        formatString = '{34:10d} {36:5d} {0:5d} {2:5d} {4:5d} \
{6:5d} {8:5d} {10:5d} {12:5d} \
{14:5d} {16:5d} {18:5d} {20:5d} \
{22:5d} {24:5d} {26:5d} {28:5d} \
{30:5d} {32:5d} ;'
        print '# f(Hz) C giroX giroY F3 FC6 P7 T8 F7 F8 T7 P8 AF4 F4 AF3 O2 O1 FC5'
    else:
        formatString = '{0:4d},{2:4d},{3:1d},{4:4d},{5:1d},\
{6:4d},{7:1d},{8:4d},{9:1d},{10:4d},{11:1d},{12:4d},{13:1d},\
{14:4d},{15:1d},{16:4d},{17:1d},{18:4d},{19:1d},{20:4d},{21:1d},\
{22:4d},{23:1d},{24:4d},{25:1d},{26:4d},{27:1d},{28:4d},{29:1d},\
{30:4d},{31:1d},{32:4d},{33:1d};'

    while True:
for packet in emotiv.dequeue():
            currentSample = currentSample + 1
            currentTime = (1000.0*float(currentSample))/hzf
            checkSound()
            print formatString.format(packet.counter, packet.sync,
 packet.gyroX[0], 4,
 packet.gyroY[0], 4,
 packet.F3[0],  packet.F3[1],
 packet.FC6[0], packet.FC6[1],
 packet.P7[0],  packet.P7[1],
                                  packet.T8[0],  packet.T8[1],
 packet.F7[0],  packet.F7[1],
 packet.F8[0],  packet.F8[1],
 packet.T7[0],  packet.T7[1],
 packet.P8[0],  packet.P8[1],
 packet.AF4[0], packet.AF4[1],
 packet.F4[0],  packet.F4[1],
 packet.AF3[0], packet.AF3[1],
 packet.O2[0],  packet.O2[1],
 packet.O1[0],  packet.O1[1],
 packet.FC5[0], packet.FC5[1],
                                  currentSample, currentTime, currentFrequency)
#sys.stdout.flush()
#time.sleep(1.0/128.0)

try:
    logger = logging.getLogger('emotiv')
```

```
    logger.setLevel(logging.INFO)
    log_handler = logging.StreamHandler()
    logger.addHandler(log_handler)
    emotiv = Emotiv()
    main(*sys.argv[1:])
finally:
    if emotiv:
        emotiv.close()
```

4.6 Emotiv to OpenVibe and EEGLab

Here we present some ideas and tools to connect the Emotiv EPOC to some elaboration frameworks: the headset comes in a consumer/developer/research edition, but we are interested in Open Source elaboration tools, so we suppose to have just the raw data in a ASCII file.

4.6.1 Getting the Emotiv EPOC raw data

This is the initial step which must be accomplished to start the work. There are 2 possibilities:

- use the Emotiv EPOC Research Edition and the given framework
- use the Emokit tools to acquire the data

When raw data is available, we can store it in an ASCII file, one acquisition for line, each line being the sequence of acquired samples. We do not use a more definite format, as these notes **are not an out-of-the-box solution**: we just give the basic ideas to test the various solutions.

4.6.2 From raw data to OpenVibe

OpenVibe is a very flexible tool to analyze EEG data, in particular it supports various input mode. The one we are exploiting is the **Generic Raw Telnet Reader** with the following configuration:

- Generic Raw Telnet Reader
- Number of channels: 16
- Sampling Frequency: 128
- port 1337

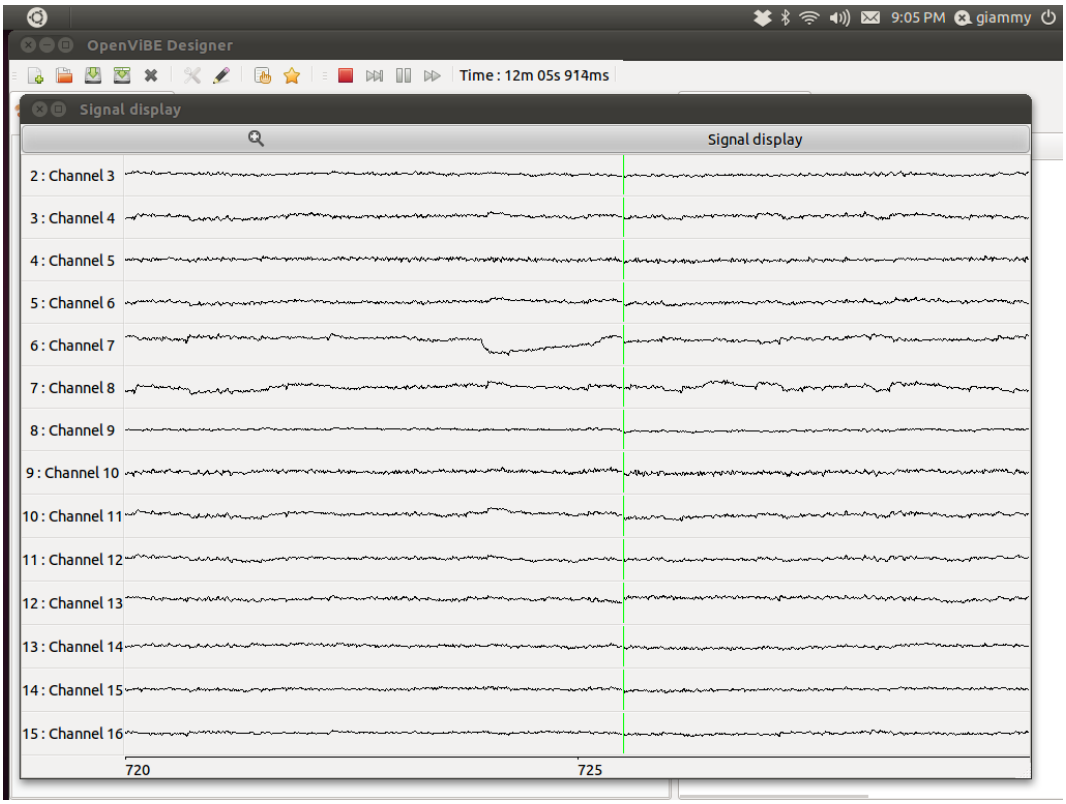

Figure 4.1: An EEG from Emotiv EPOC acquired on OpenVibe

- Limit speed - checked
- Little endian
- 16 bits unsigned integer
- Header size: 0
- Inter frame size: 0

We need to start the acquisition server (one of the modules of OpenVibe: `dist/ov-acquisition-server.sh`), select the **Generic Raw Telnet Reader** and set the given options.

After this we run a scenario in OpenVibe Designer `dist/ov-designer.sh`: the scenario used for this test includes just the acquisition box and a graphical widget to see the data in realtime.

Having OpenVibe up and running, the Emotiv EPOC ASCII raw data is piped to the `a2b` tool which just takes the input stream as ASCII numbers

and transform it in a binary output stream (16 bit unsigned integers, little endian - as defined in the acquisition server) The source is the following:

```
#include <stdio.h>

// ./a2b | nc -l 1337
//
// Run acquisition driver for Emotiv EPOC:
//   ./produceASCIIoutput.py | ./a2b | nc -l 1337
//
// Run OpenVibe acquisition server
//   t60/openvibe-0.11.0-svn3052-src/dist$ ./ov-acquisition-server.sh
// Configuration:
//   Generic Raw Telnet Reader
//   Number of channels: 16
//   Sampling Frequency: 128
//   port 1337
//   Limit speed - checke (cos'e'?)
//   Little endian
//   16 bits unsigned integer
//   Header size 0
//   Inter frame size 0
//
// Run OpenVibe designer
//   ./ov-designer-gmy.sh
// with scenario: t60/gmy-test1.xml
//

#define LINE_SIZE 256
#define NCH 16

#define LO(x)   ((unsigned char)((x)&0xff))
#define HI(x)   ((unsigned char)(((x)>>8)&0xff))

int main(int argc, char *argv[])
{
  int i;
  int ch[NCH];
  char theLine[LINE_SIZE];
  char *ret;

  for (i=0; i<NCH; i++) {
    ch[i] = 0;
  }
```

```
  while ((ret = fgets(theLine, LINE_SIZE, stdin)) != 0) {
    //printf("Read %d '%s'\n", (int)ret, theLine);
    sscanf(theLine, "%d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d",
   &ch[0], &ch[1], &ch[2], &ch[3], &ch[4], &ch[5], &ch[6], &ch[7],
   &ch[8], &ch[9], &ch[10], &ch[11], &ch[12],
   &ch[13],&ch[14], &ch[15]);
    for (i=0;i<NCH;i++) {
      putchar(LO(ch[i]));
      putchar(HI(ch[i]));
    }
  }
}
```

The tool sends data to the standard output: the network management is done using the standard Linux tool `nc` (see a Linux manual for the details: the command is named `nc` or `netcat`) which take the binary stream and sends it to the socket 1337 (the one defined in the acquisition server).

At this point we use the Linux flexibility: as we use pipes to manage data, the incoming stream can come from a file or from a running program doing a realtime acquisition and sending data in realtime to standard output. In this configuration we get a tool to acquire and visualize EEG data in realtime. The next step is to prepare an elaboration scenario.

Working with the research edition you need to write a server in a Windows machine, from there the connection to OpenVibe is the same (or use the included OpenVibe driver for Emotiv Epoc).

4.6.3 From raw data to EEGLab

EEGLab is an Open source suite which runs in Matlab, for non realtime EEG data elaboration: we proced to save on file an acquisition session, then we convert it for EEGLab using EDF format.

In this case we are interested in recording some events (the instant of the PC beep and it's frequency, as we will need to identify evoked potentials.

Data is converted in EDF format with the following tool, which is just a test program, not a program to use in the field as the header is manually written in the code!

```
/*

Copyright 2011 Gianluca Moro - giangiammy@gmai.com - www.giammy.com
```

```
This file is a2edf utility.

a2edf is free software: you can redistribute it and/or modify it under
the terms of the GNU General Public License as published by the Free
Software Foundation, either version 3 of the License, or (at your
option) any later version.

a2edf is distributed in the hope that it will be useful, but WITHOUT
ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
for more details.

You should have received a copy of the GNU General Public License
along with Foobar. If not, see http://www.gnu.org/licenses/.

*/

/*
 An ASCII to EDF format converter

 ./a2edf <in >out.edf

 THIS IS AN IMPLEMENTATION TO TEST THE FORMAT CONVERSION
 AND THE EDF SPECIFICATION.
 TO BE USED IN REAL APPLICATION THE SOURCE MUST BE MODIFIED!

 The input file is an ASCII file, each line is a sequence of
 17 integers, space separated representing the following type
 of data:
 Freq giroX giroY F3 FC6 P7 T8 F7 F8 T7 P8 AF4 F4 AF3 O2 O1 FC5
 where Freq is the audio frequency emitted in that instant,
 giroX and giroY are a giroscope input and the others are EEG sensors.

 The header is built ad hoc, all the fields should be setted runtime,
 in particular the
 printf("%-8s", "324");         // TODO DATA number of data record
 (see the note below)

 References:

 @article{
 Kemp_Värri_Rosa_Nielsen_Gade_1992,
 title={A simple format for exchange of digitized polygraphic recordings.},
```

```
 volume={82},
 url={http://linkinghub.elsevier.com/retrieve/pii/0013469492900097},
 number={5},
 journal={Electroencephalography and Clinical Neurophysiology},
 author={Kemp, B and Värri, A and Rosa, A C and Nielsen, K D and Gade, J},
 year={1992},
 pages={391--393}}


 *** Note ***
 The implementation has been checked with EDFbrowser
 http://www.teuniz.net/edfbrowser/ but the field
 "number of data record" accordingly to the paper
 could be -1 (in my interpretation: read as many data
 record as you find in the file), while EDFbrowser want
 a real number of data record.

 For the date/time format, the paper writes:
 "startdate of recordlng (dd mn yy)" i.e. dd,mm and yy are
 space separated, while EDFbrowser wants them point separated.

*/

#include <stdio.h>
#include <stdlib.h>

#define LINE_SIZE 256

#define NCH_IN 17
#define NCH 18            // event: 0=no sound, 1=low freq, 2=high freq
#define NSECPERRECORD 1
#define FREQ 128

short dataRecord[NCH][NSECPERRECORD*FREQ];
int currentDataRecordSamples = 0;
int eventCounter = 0;

#define LO(x)   ((unsigned char)((x)&0xff))
#define HI(x)   ((unsigned char)(((x)>>8)&0xff))

void printEDFHeader() {
  int i;
  // main header
```

```
  printf("%-8s", "0");           // version
  printf("%-80s", "Giammy");     // TODO TEST local patient id
  printf("%-80s", "Casa mia");   // TODO TEST local recordin id
  printf("%-8s", "14.10.11");    // TODO TEST start recording date
  printf("%-8s", "14.15.16");    // TODO TEST start recording time
  printf("%-8d", 256+256*NCH);   // header size: 256+ns*256
  printf("%-44s", "");           // reserved
  printf("%-8s", "324");         // TODO DATA number of data record
  printf("%-8d", NSECPERRECORD); // duration of data record in seconds
  printf("%-4d", NCH);           // number of signals in data record

#define FOR_ALL for (i=0; i<NCH; i++)

  // data header
  printf("%-16s", "Freq");      // label
  printf("%-16s", "giroX");     // label
  printf("%-16s", "giroY");     // label
  printf("%-16s", "F3");        // label
  printf("%-16s", "FC6");       // label
  printf("%-16s", "P7");        // label
  printf("%-16s", "T8");        // label
  printf("%-16s", "F7");        // label
  printf("%-16s", "F8");        // label
  printf("%-16s", "T7");        // label
  printf("%-16s", "P8");        // label
  printf("%-16s", "AF4");       // label
  printf("%-16s", "F4");        // label
  printf("%-16s", "AF3");       // label
  printf("%-16s", "O2");        // label
  printf("%-16s", "O1");        // label
  printf("%-16s", "FC5");       // label
  printf("%-16s", "FreqEvent"); // label

  FOR_ALL {
    printf("%-80s", "Transducer type");// transducer type
  }
  FOR_ALL {
    if (i==0)
      printf("%-8s", "Hz");            // physical dimens.
    else
      printf("%-8s", "raw");            // physical dimens.
  }
  FOR_ALL {
```

```
  switch (i) {
  case 0:
    printf("%-8s", "0");          // physical min
    break;
  default:
    printf("%-8s", "-500");          // physical min
  }
}
FOR_ALL {
  switch (i) {
  case 0:
    printf("%-8s", "5000");          // physical max
    break;
  default:
    printf("%-8s", "500");          // physical max
  }
}
FOR_ALL {
  switch (i) {
  case 1:
  case 2:
    printf("%-8s", "-128");         // digital min
    break;
  default:
    printf("%-8s", "0");          // digital min
  }
}
FOR_ALL {
  switch (i) {
  case 1:
  case 2:
    printf("%-8s", "128");         // digital max
    break;
  default:
    printf("%-8s", "16384");          // digital max
  }
}
FOR_ALL {
  printf("%-80s", "prefiltering"); // prefiltering
}
FOR_ALL {
  printf("%-8d", FREQ);            // number of samples per data record
}
```

```
  FOR_ALL {
    printf("%-32s", "");             // reserved
  }
}

int main(int argc, char *argv[])
{
  int i,j;
  int ch[NCH];
  char theLine[LINE_SIZE];
  char *ret;
  int prevFreq = 0;

  for (i=0; i<NCH; i++) {
    ch[i] = 0;
  }

  printEDFHeader();

  while ((ret = fgets(theLine, LINE_SIZE, stdin)) != 0) {
    //printf("Read %d '%s'\n", (int)ret, theLine);
    sscanf(theLine, "%d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d",
   &ch[0], &ch[1], &ch[2], &ch[3], &ch[4], &ch[5], &ch[6], &ch[7],
   &ch[8], &ch[9], &ch[10], &ch[11], &ch[12],
   &ch[13],&ch[14], &ch[15], &ch[16]);

    // build ch[17], the event, from ch[0], the frequency
    ch[17] = 0;
    if (ch[0] != 0 && prevFreq == 0) {
      ch[17] = ch[0];
      ++eventCounter;
    }
    prevFreq = ch[0];

    for (i=0; i<NCH; i++)
      dataRecord[i][currentDataRecordSamples] = ch[i];
    ++currentDataRecordSamples;

    if (currentDataRecordSamples >= NSECPERRECORD*FREQ) {
      for (i=0;i<NCH;i++) {
for (j=0;j<NSECPERRECORD*FREQ;j++) {
  putchar(LO(dataRecord[i][j]));
  putchar(HI(dataRecord[i][j]));
```

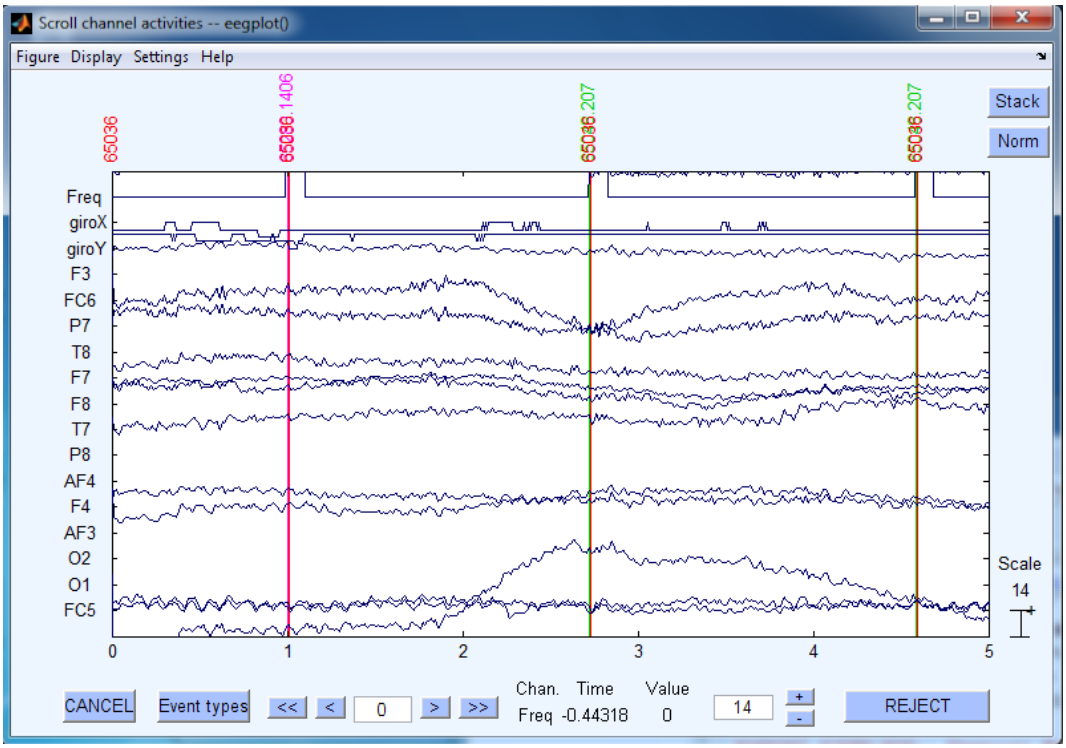

Figure 4.2: An EEG from Emotiv EPOC acquired on EEGLab, with recorded events

```
}
      }
      currentDataRecordSamples = 0;
    }
  }

  printf("EventCounter=%d\n", eventCounter);
}
```

The program takes as input a raw ASCII EEG and produces an EDF file: `./a2edf <in >out.edf`. In the output file a new signal is added, to code the beep event, which is automatically recognized by EEGLab.

4.6.4 Conclusion: OpenVibe and EEGLab

The two proposed tools give us the possibility to have EEG data available in OpenVibe or EEGLab. At the moment the scenario is:

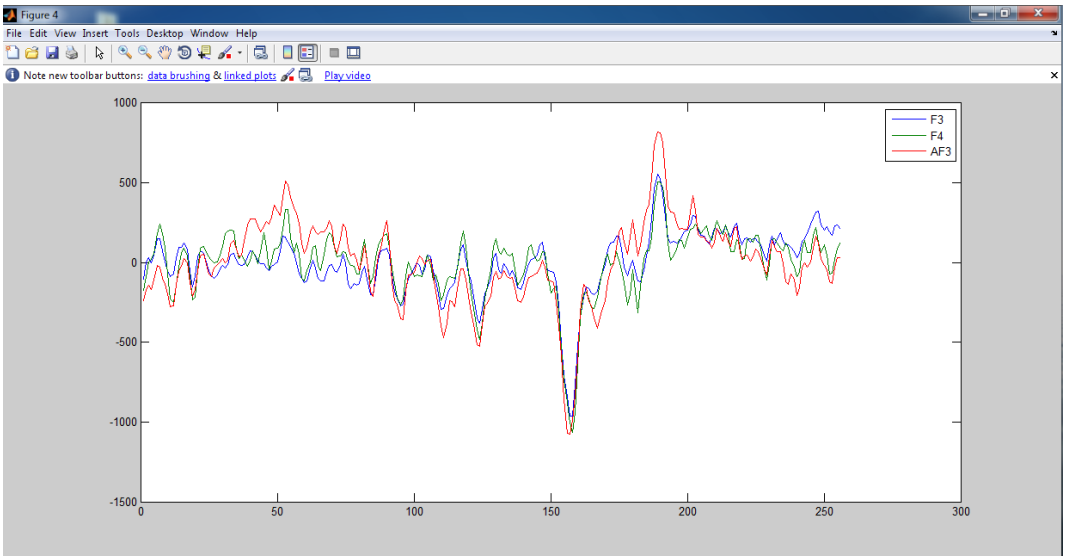

Figure 4.3: Graph of evoked potentials

1. getting the raw data from Emotiv EPOC via Research Edition SDK or Emokit
2. store the data on file for post-elaboration
3. build a server and stream the data via socket in realtime
4. or both ... obviously!
5. `for post-elaboration`: convert the data file to EDF format and import it in EEGLab (with informations about events - needed for evoked potential rilevation)
6. `in realtime`: use OpenVibe to get data via Raw Telnet

The idea of having 2 different elaboration tools is that we will use the one which will suit best to our needs. Other frameworks interfaces could be tested in the future.

4.7 A first manual P300 elaboration

Referring to the Audio Evoked Potential test described in section 4.5, **Test: Evoked potential**, we present a preliminary manual data analysis.

The test emitted high frequency sounds, with 33 low frequency sounds (random rare events).

The plotted signals are the resulting media of all the acquisitions corresponding to the 2s interval around the rare events.

We report just the signal corresponding to F3, F4, AF3 (the position of the sensors must be verified): in these signals the peak is more evident.

The rare event occurs at sample 128, while the peak occurs about 30 samples after (a quarter of second).

This seems to suggest that the Emotiv Epoc can record an evoked potential, but is preliminary data and preliminary report: more test - and in realtime configuration, must be done!

Chapter 5

Resources

- P300 speller Matlab code: http://asi.insa-rouen.fr/enseignants/ arako-tom/code/bciindex.html
- A tool to read data from a socket on Android: http://giammy.com/eegview
- Initial tests for Bluetooth interface on Android: http://github.com/giammy/bscan
- Discussion Mailinglist: http://groups.google.com/group/medical-bci (email address: medical-bci@googlegroups.com)
- Brain scanner on N900: http://milab.imm.dtu.dk/eeg
- Lego Mindstorm and Emotiv Epoc: http://www.youtube.com/watch?v=HLbly5rWPK0
- Mind control of a sculpture: http://brainstorms.puzzlebox.info/

5.0.1 Not strictly BCI

http://www.ofoa.net/applications/eagleeyes_apply.cfm

Chapter 6

Equipment

http://www.gtec.at/
http://www.micromed.eu/

Chapter 7

Tablets, Android, Kernel

7.1 ASUS TF101

```
PATH=$PATH:/home/giammy/Desktop/tf101/toolchain/prebuilt/linux-x86/toolchain/arm-

How to recompile Asus TF101 kernel

Main guide was taken from
http://forum.xda-developers.com/showthread.php?t=1197147
with some minor changes.

Now let's try to recompile the android kernel.  You will need some
packages not included in a default Ubuntu 11.04 installation:

sudo apt-get install git unrar libncurses5-dev qt3-dev-tools

Unrar and git are the only necessary packages.  Install libncurses if
you want to modify the kernel configuration with the command line
menu.  Otherwise, if you want a graphical interface, go with qt3.
Please refer to the documentation of your linux distro for help on how
to install additional packages.  For the purpose of this guide we are
going to rebuild the official asus kernel with no modifications.

Download the latest kernel source from the ASUS site (atm 2.6.36.3)

wget http://dlcdnet.asus.com/pub/ASUS/EeePAD/TF101/Eee_PAD_TF101_Kernel_Code_8_4_
```

```
Now unpack it:

unrar e Eee_PAD_TF101_Kernel_Code_8_4_4_11.rar
tar xvfz kernel-3eb19ec-fix-lic.tar.gz

We need a config file for the compile process.  The config file
essentially sets features ON or OFF from the final kernel (yes, I know
that is much more than this...).  If your tablet is upgraded to the
latest build (8.4.4.11) you can extract the config file from your
running device:

adb pull /proc/config.gz ./

Now uncompress and move it into the kernel dir (renaming it in
.config).

gunzip config.gz
cp config kernel-3eb19ec/.config

We will need a compiler for the ARM platform.  The android source
comes with the arm compiler. You can download it following these
instructions http://source.android.com/source/downloading.html .
Or we download only the prebuilt toolchain:

git clone git://android.git.kernel.org/platform/prebuilt.git

Mirrors are available at https://github.com/android
for example:

git clone https://android.googlesource.com/platform/prebuilt.git

After git completes the download, you will have a new directory called
"toolchain".  We have to set some environment variables (assuming you
have downloaded the toolchain in your home directory):


export ARCH=arm
export CROSS_COMPILE=arm-eabi-
export PATH=$PATH:~/toolchain/prebuilt/linux-x86/toolchain/arm-eabi-4.4.3/bin/

or (for MacOSX):
```

```
export ARCH=arm
export CROSS_COMPILE=arm-eabi-
export PATH=$PATH:~/toolchain/prebuilt/darwin-x86/toolchain/arm-eabi-4.4.3/bin/


The compiler is installed and configured.  It's time to come back
to the kernel source.  If you want to modify some settings in your
config file (!!AT YOUR OWN RISK!!) you can edit the .config file or
use a graphical interface.

cd kernel-3eb19ec
make xconfig (make menuconfig if you prefer a command-line menu)

Now we can compile the android kernel:

make -j4

Our new kernel is stored in : arch/arm/boot/zImage

We can use it for make a boot image and flash in our TF101:
use this guide
http://forum.xda-developers.com/showthread.php?t=1193737
```

7.2 VIA8560

Here we have some preparatory tests to connect the Emotiv to an Android tablet. The tablet is the MID 7 - Via8650.

The device is:

```
# cat cpuinfo
# Processor    : ARM926EJ-S rev 5 (v5l)
BogoMIPS       : 797.97
Features       : swp half thumb fastmult edsp java
CPU implementer  : 0x41
CPU architecture: 5TEJ
CPU variant   : 0x0
CPU part   : 0x926
CPU revision   : 5
```

```
Hardware   : WMT
Revision   : 0000
Serial     : 0000000000000000

# meminfo
MemTotal:         220764 kB
```

To recompile a kernel, you need the ARM toolchain and the kernel sources.

7.3 ARM toolchain

We will need a compiler for the arm platform. The android source comes with the arm compiler. You can download it following this instructions http://source.android.com/source/downloading.html . Or we download only the prebuilt toolchain:

git clone git://android.git.kernel.org/platform/prebuilt.git

Mirrors are available at https://github.com/android for example:

git clone https://android.googlesource.com/platform/prebuilt.git

After git finishes his download, you will have a new directory called "toolchain". We have to set some environment variables (i assume you have downloaded the toolchain in your home directory):

An example of environment variables setting for Linux is:

```
export ARCH=arm
export CROSS_COMPILE=arm-eabi-
export PATH=$PATH:~/toolchain/prebuilt/linux-x86/toolchain/arm-eabi-4.4.3/bin/
```

7.4 Kernel sources

Finding tablet's kernel sources is usually a mess, as not all the producer have them available in their site.

As examples, the ASUS Transformer TF101 can be downloaded from the ufficial site:

```
http://dlcdnet.asus.com/pub/ASUS/EeePAD/TF101/Eee_PAD_TF101_Kernel_Code_8_4_4_11.
```

while for the Via8650 we have the vendor official archive:

```
ftp://ftp.gpl-devices.org/pub/vendors/Wondermedia/WM8650/
```

the file is: `KERNEL-DS_ANDROID_2.6.32_WM8650.111209.1514.tgz`

The configuration used for the compilation can (sometimes) be downloaded from the tablet:

```
adb pull /proc/config.gz ./
```

The compiler is installed and configured. It's time to modify some settings in your config file (!!AT YOUR OWN RISK!!) editing the .config file or use a graphical interface.

cd kernel make xconfig (make menuconfig if you prefer a command-line menu)

Now we can compile the android kernel with the command `make`: our new kernel is stored in

```
arch/arm/boot/zImage
```

while the command `make uImage` produces:

```
arch/arm/boot/uImage
```

7.4.1 Tablet ROM update

This tablet has a modded ROM available here:

http://www.slatedroid.com/topic/18025-rom-universal-hybrid-honeycombmod-uberoid-for-wm8650-devices-v130-w-video/

http://www.techknow.t0xic.nl/forum/index.php?action=forum

The package is `Universal_Uberoid_WM8650_v10.1_www.TechKnowForum.net` `http://www.mediafire.com/?z4nd3h85q0s65ok`

The update procedure is well done: just put the new image on a SD, and the tablet, when sees it, flash it! One way to proceed it to build an Uberoid standard SD update card and then put our modification in it.

For the Uberoid ROM, the correct type to select is 11 (Via8650, MID 7)

7.5 Debian on the Mid and Via8650 tablet

It seems it can be done:

http://www.slatedroid.com/topic/23028-running-debian-on-wm8650/

with the following image:

http://www.filefactory.com/file/ce12c6b/n/debian_wm8650_0.5.img.7z

You need a 4G (or more) SD card and just

```
dd if=theimage of=/dev/thecard
```

This is a kernel source for via8560:

`https://gitorious.org/linux-on-via-vt8500/vt8500-kernel`

The hardware support depends on the device: on the MID 7 the keyboard does not work.

Chapter 8

Acer Iconia A500 kernel recompile

Acer Iconia A500 is a tablet based on Android whose interesting feature is the standard USB port: this is valuable to connect an Emotiv EPOC device to the tablet as it uses a USB dongle: a standard HID usb device, which can be accessed via the standard linux `/dev/hidraw` driver.

The main problem with the tablet is that the HIDRAW support is not compiled in the kernel. Here we see how to compile and install a custom kernel on the Acer Iconia A500: the original kernel is: `Linux version 2.6.36.3 (simba@lion) (gcc version 4.4.3 (GCC) ) #1 SMP PREEMPT Wed Jun 29 14:15:34 CST 2011`

8.1 A500 and ClockworkMod ROM Manager

To work with the kernel is a good (almost mandatory, I would say) idea to have a recovery tool such as the http://www.clockworkmod.com/ ROM manager, which allow you to (from the ufficial description):

```
Must have app for any root user. Make backups, flash ROMs,
and own your device. ROM Manager is THE MUST HAVE APP for
any Android root user.
(Over 3 million downloads and counting!)
* Flash your recovery to the latest and greatest
  ClockworkMod recovery.
* Manage your ROMs via a handy UI.
* Organize and perform backups and restores from within
  Android!
* Install ROMs from your SD card.
```

```
* Install your favorite ROMs over the air!
```

The installation on the A500, with Android 3.1 installed (from the ufficial update) is not very plain as this device is not supported. Anyway, from a clean 3.1 update you need to use Acer Recovery Installer from Google market http://www.addictivetips.com/mobile/install-clockworkmod-recovery-on-acer-iconia-a500-honeycomb-tablet/ so:

- root the device
- install Acer Recovery installer from the market
- using the previous tool, you can install the ClockworkMod recovery

This is a good moment to STOP AND BACKUP ALL!
The ROM manager can be activated in Recovery mode: for the A500 you need to push volume down and then push the power on. There will be a menu with a lot of options, and the possibility to do a "Complete backup". Do it, try to recover from that backup, store it on a USB device, copy on your PC ... do not lose it!

8.2 Kernel compilation

The first step is to download the ufficial kernel sources from the Acer site http://http://us.acer.com/ac/en/US/content/drivers. At the time of writing (May, 2012) the kernel for version 3.1 had a missing Makefile, while the kernel for 3.2 was complete. The kernel is configured via a `.config` file which can be pulled from our device (the number is the A500 device UID):

```
~ $ adb devices
List of devices attached
280404443611197 device

~ $ adb pull /proc/config.gz config.gz
```

we will find a file `congif.gz` which can be expanded and moved as `.config` in the root of the kernel source tree. The only modification we need is to enable HIDRAW support: there should be some lines like:

```
CONFIG_HID_SUPPORT=y
CONFIG_HID=y
# CONFIG_HIDRAW is not set
```

the last line should be uncommented.

After this you need to recompile the kernel with the modified `.config` file: the quick overview of the procedure is:

- you need a Linux box with development tools installed
- install the ARM cross compiler
- expand the kernel source to `KERNEL_DIR`
- copy the modified `.config` to `KERNEL_DIR`
- compile with something like:

```
#!/bin/bash
export ARCH=arm
export CROSS_COMPILE=arm-eabi-
export PATH=$PATH:/home/giammy/Desktop/toolchain/prebuilt/
                    linux-x86/toolchain/arm-eabi-4.4.3/bin/
#Our new kernel is stored in : arch/arm/boot/zImage
```

- the kernel will be in `KERNEL_DIR/arch/arm/boot/zImage`

8.3 Android partitions' structure

The partition map of Android can vary from device to device: some infos are present in `http://elinux.org/Android_Fastboot`, but the memory for the A500 is mapped to `/dev/block/mmcblk0pn` with:

mmcblk0p1 recovery partition

mmcblk0p2 kernel, the place where the Linux kernel and RAM disk reside;

mmcblk0p3 base system;

mmcblk0p4 ...

mmcblk0p5 main file system;

mmcblk0p6 ...

mmcblk0p7 checksums of other partitions (follow explanations...)

Here we work only in `/dev/block/mmcblk0p2`, where the kernel and RAM disk are stored.

WARNING: if you use the wrong partition you can brick your device. What I did, worked on my device: I do not know what will happen to your device!

The kernel is located in the `mmcblk0p2` partition, packed with the RAM disk. To get the kernel you need to fetch the right partition: from the adb shell just do:

```
$ adb shell
$ su
# dd if=boot-mia.img of=/dev/block/mmcblk0p2
```

Note that you need a rooted device to access to the shell as root. Be careful that the shell application will ask you on the tablet display the authorization to become root, so, if you run the command without looking at the tablet display (unlocked) you miss the request and cannot become root.

The command is the standard GNU/Linux `dd` command - see a Linux manual for more info.

WARNING: if you give the wrong command you can brick your device: do this only if you undersand what you are doing!

Note that the backup procedure, essentially copy all the partition in, more or less this way.

8.4 The kernel and its partition

The kernel is stored in the partition 2, as we have seen, packed in a binary blob which we call `boot.img` with the RAM disk: to change the kernel we need to unpack the original kernel, compile our new kernel, repack it in a new `boot.img`. A description can be found here: http://android-dls.com/wiki/index.php?title=HOWTO:_Unpack%2C_Edit%2C_and_Re-Pack_Boot_Images.

The tools used to unpack the boot image are `split_bootimg.pl`: a perl script referred to in the previous quoted wiki page, whose output is something like:

```
# perl split_bootimg.pl boot.img
Page size: 2048 (0x00000800)
Kernel size: 2962516 (0x002d3454)
Ramdisk size: 147895 (0x000241b7)
Second size: 0 (0x00000000)
Board name:
```

```
Command line:
Writing boot-original-a500.img-kernel ... complete.
Writing boot-original-a500.img-ramdisk.gz ... complete.
```

and the two relevant files `boot.img-kernel` and `boot.img-ramdisk.gz`: the first is the kernel and the second ... guess what!

At this point you need to repack the image with your kernel (at the moment, we keep the same ramdisk): the tool for this operation is `mkbootimg` with a command line as:

```
../mkbootimg --cmdline "nvmem=128M@384M mem=1024M@0M
  vmalloc=256M video=tegrafb console=none
  usbcore.old_scheme_first=1 lp0_vec=8192@0x1840c000
  tegraboot=sdmmc gpt" --base 0x10000000
  --kernel zImage --ramdisk boot.img-ramdisk.cpio.gz -o
  output.img
```

be careful to the base address `--base 0x10000000`.

Now we need to reflash the new image to our device:

WARNING: IF YOU BRICK YOUR DEVICE DON'T BLAME ME!

The command line to reflash is:

```
# dd if=boot.img of=/dev/blobk/mmcblk0p2
```

At this point you could have bricked your device if the message you see at the following boot is:

```
secure boot: image LNX fail
```

The problem is that A500 keeps all the checksums of the partitions on the last partition, probably to stop people doing what we are doing, but this has been documented in http://forum.xda-developers.com/showthread.php?t=1121543; they say: "Checksums for all partitions are stored on mmcblk0p7. During boot, the actual checksums for boot.img and recovery.img are calculated and compared to the values stored in p7." You need to download the `itsmagic` tool and run it, or run this tool from the ClockworkMod recovery. What it does is to write a blob of 16 bytes somewhere on partition 7: if this is not clear to you, don't worry - you are not alone - you just understand the reason of its name: `it' s magic` :-)

At this point, we can reboot the tablet and see other problems: it shows the android logo and then it reboots and so on and so forth ... This behaviour was fixed with Clockwork recovery resetting everything to factory default (reset cache, dalvik cache, user data ...) As conclusion, the steps are:

- get kernel source
- compile them with your modifications
- get the original boot image
- unpack, change kernel, and repack
- do a dd of the new image
- run `itsmagic`
- factory reset of everything

8.5 Iconia A500 and Emotiv EPOC

At this point, the result is that, connecting to the USB port, the Emotiv Epoc Dongle, tha raw data is available on `/dev/hidraw1`. Now we need to use it.

TODO: Note that the hidraw device is now supported but there is a problem with the new kernel: at the moment the system tasks show the output as "mirrored left to right": we need to investigate this. Another problem is that enabling wifi, resets the tablet: we think that we got the procedure to ricompile the kenel, but we are missing some critical configuration!

Chapter 9

References

Devices

http://www.emotiv.com/ A low cost - not as low as PlxWave, but with more sensors, 5

http://www.gtec.at/ Guger technologies, 41

http://www.micromed.eu/ MicroMed, 41

http://www.neurosky.com/ a chip used by various producer, 5

http://www.plxwave.com/ A low cost device, 5

http://www.ted.com/talks/lang/eng/tan_le_a_headset_that_reads_your_brainwaves.h Emotiv demo, 9

discussion

http://www.emotiv.com/forum/messages/forum4/topic484/message2659/?phrase_id=15 Emotiv forum, 10

disease

http://en.wikipedia.org/wiki/Locked-in_syndrome Locked in syndrome, 6

EEG

http://www.bem.fi/book/13/13.htm EEG theory, 9

emokit

http://daeken.com/emokit-hacking-the-emotiv-epoc-brain-computer-0 Emokit, 12

http://github.com/qdot/emokit interface to Emotiv Epoc, 12, 17

Emotiv

http://emotiv.com Emotiv official site, 9

Input

http://www.inference.phy.cam.ac.uk/dasher/DasherSummary.html an interesting input method, 15