Application of Artificial Neural Networks for Human Muscle Signal Analysis and Mechanical Equipment Control. 2. Hardware, software, interfaces

Vadim Gerasimov [1 a] , Gintaras Jonaitis [2], Vytautas Jonkus [1]
[1] Machine-to-Machine laboratory, Department of Radiophysics,
Faculty of Physics, University of Vilnius, Saulėtekio al. 9-III, LT2054 Vilnius, Lithuania
[2] Laboratory for Medical Rehabilitative and Assistive Technologies, Department of Biomechanics,
Faculty of Mechanics, Vilnius Gediminas Technical University, J. Basanavičiaus str. 28, Vilnius, Lithuania

Received 12 June 2015, accepted 5 August 2015

Abstract. Design of electronic system devoted for processing and recognition of myoelectric signals were observed. Description of used embedded systems were presented and analyzed.

Citations: Vadim Gerasimov, Gintaras Jonaitis, Vytautas Jonkus. Application of Artificial Neural Networks for Human Muscle Signal Analysis and Mechanical Equipment Control. 2. Hardware, software, interfaces – *Innovative Infotechnologies for Science, Business and Education,* ISSN 2029-1035 – **1(18)** 2015 – Pp. 13-17.

Keywords: Artificial Neural Networks; ANN; recognition of myoelectric signals; electromyogram; linear regression method; supervised learning; logistic regression; artificial neuron.
Short title: Artificial Neural Networks. 2. Hardware, software

Introduction

Previous publication [1] represents literature overview - how to create the electronic system devoted for processing and recognition of myoelectric signals. Artificial Neural Networks (ANN) algorithm was described as an electronic model based on the neural structure of the human brain.

The hardware needed for scanning and amplification of myolectic signal was provided by Laboratory for Medical Rehabilitative and Assistive Technologies at Vilnius Gediminas Technical University. The hardware construction, assembly and testing was carried out at Machine to Machine Lab at Vilnius University in 2014÷2015. Such signal decoding and interpretation could be used in prostheses technology and other human-machine interface.

Work objectives could be formulated as follows:
a) to design an interface, suited for myoelectric signal recognition - from human body mounted sensors;
b) to construct a robot-manipulator controllable by afore mentioned interface.

1. Embedded systems

The *embedded system* represents a computer system, the purpose of which is precisely determined in a mechanical or an electronic system. Often it is equipped with an operating system (OS), which can be embedded or real-time.

The difference between them in execution manner is that the former, embedded system buffers its tasks [2], usually it has its own command interpreter and additional applications such as the Internet browser can be installed. The latter is real-time system; the performance of its tasks is carried out as soon as possible and the tasks are started not from command line or user interface, but programmed before booting the system. An embedded system may even be somewhat similar to that of a normal PC system [2]. Some embedded systems are able to compile the executable code inside without the use of external equipment.

Physically an embedded system can be used for different purposes - from digital clocks and music players, to industrial objects - traffic lights and production lines. Its user interface can be graphical (GUI) with various controls, console prompt, or without any control. A system with GUI may be connected to an external monitor or an embedded miniature screen. Most frequently used processors are RISC processors (Reduced Instruction Set Computing). These processors can be with an integrated memory or with external circuits, suited for the operational memory access. The number of cores can be from one to several, e.g. Raspberry Pi B and Pi 2 differ in core number. The second model may utilize its four cores for parallel computing software.

An embedded system may also be designed with a periph-

[a]Corresponding author, email: vadim.gerasimov@ateities.lt

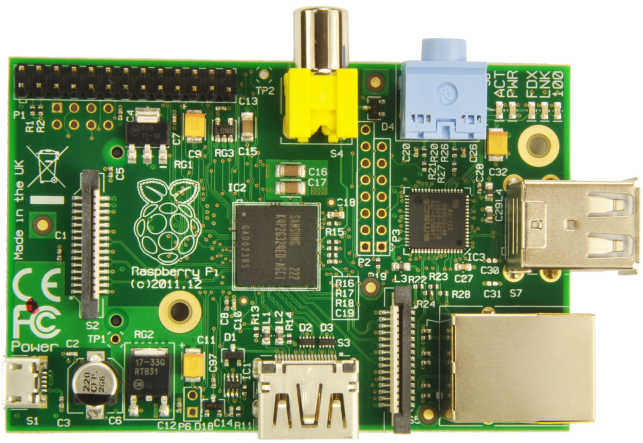

Fig. 1. Raspberry Pi B board. According to Ref. [4].

ery: asynchronous transmission line (RS-232, RS-422), synchronous transmission line (I2C, SPI), USB, card reader, network interface, timers, discreet/digital signal pins, analogue/digital converters. [3]

Embedded system software. The embedded system control can be implemented in several ways [3].

1. *Program loop.* It calls subprograms and executes them sequentially.
2. *Interrupt-controlled tasks.* A timer or a com-line after receiving control bytes switches tasks from one to another.
3. *Cooperative parallel task distribution.* The method is similar to a loop mechanism, but here the tasks are given their own environment for execution. Thus, when a task execution is no longer needed, it calls a routine process "PAUSE", which halts the execution.
4. *A multithread operating system.* Such OS contains a kernel, which assists tasks in switching, using priorities or semaphores. This is called a real-time OS.
5. *Micro-kernels.* This is a logical step forwards in comparison to a real-time OS. The difference is that the kernel allocates the memory and switches the processor from one task to another.
6. *Monolithic kernels.* This is a higher level, when the kernels have complicated control systems. Such a system can be of a Linux or a Windows basis, therefore they need high performance hardware (higher than just any ordinary microcontroller). These embedded systems may have separate drivers or programme packages.

There are different choices of embedded Linux OS distributions, intended for different processor architectures, for example: ARM, AVR32, MIPS, Xtensa. The possibilities of an OS depend on the distribution. For example, "OpenWRT" is intended for network switches or routers, "Debian" and "Ubuntu" based Linux systems more versatile, "Arch" is suited for a certain, fully programmable task to be executed.

Embedded system board - Raspberry Pi B The reason why this variant of an embedded system was chosen is its growing popularity. There were different attempts to use the basis of this board in constructing radio transmitters, control cores, or other probe nexuses. The Raspberry Pi B illustration is shown in Fig. 1.

The board is equipped with integrated circuits, suited for HDMI, SD card, USB, Ethernet RJ45 or audio connections control. The board also has a small battery of 26 digital pins for discreet digital functions or transition lines functionality of which covers, but is not limited to digital signalling, for example UART, SPI, IIC. Different equipment can be connected to the board: video cameras, GSM modules, sensors. The board is designed to utilize the Broadcom manufactured ARM11 (ARMv6) processor - the BCM2835.

BCM2835 processor. The Broadcom manufactured processor - BCM2835 utilizes the ARM11 architecture for the main processing node (with the ARMv6 instruction set), and Broadcom VideoCore IV for the graphical processing node [5]. Its technical specification are presented below. Second level 128 KB Cache memory module; 700 MHz CPU clock speed; High Definition and multichannel video processing supporting the most popular codec's; Embedded HDMI, NTSC, PAL, VGA interfaces; 256 MB SDRAM memory; DDR NAND Flash memory USB 2.0; SD Card 3.0; Ethernet 10/100 Mbps controller; Audio interface; SPI, UART, I2C; Is able to support Linux OS'es: Raspbian, Pidora, RISC OS, Arch, FreeBSD, OpenWrt, SlackWare;

Embedded system board - Raspberry Pi 2. The newest embedded system board was used for comparison and the mechanism control implementation (see Fig. 2). The second Pi model board (like the Pi B model) also has similar integrated circuits. Among those things it is also equipped with a bigger - 40 pin battery and four core processor ARM-Cortex A7 (ARMv7) BCM2836.

BCM2836 processor. The Broadcom's BCM2836 processor uses an ARM A7 architecture for the CPU (with the ARMv7 instruction set) [7]. Its additional specifications are (save for the basic BCM2835 ones): 900 MHz clock frequency; Only HDMI interface; 1 GB LPDDR2 SDRAM memory; Able to additionally support OS'es like: Android, Ubuntu, Windows 10.

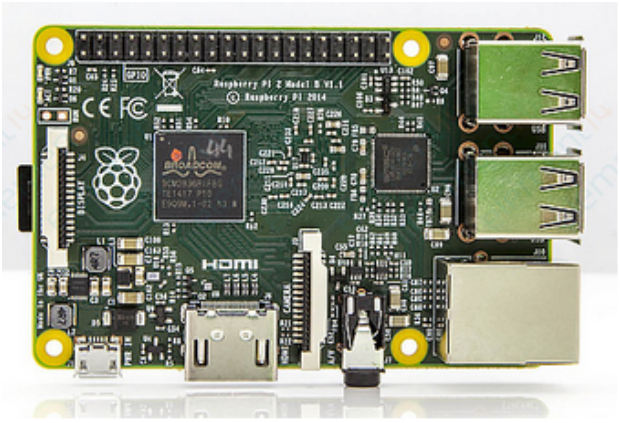

Fig. 2. Raspberry Pi 2 board. According to Ref. [6].

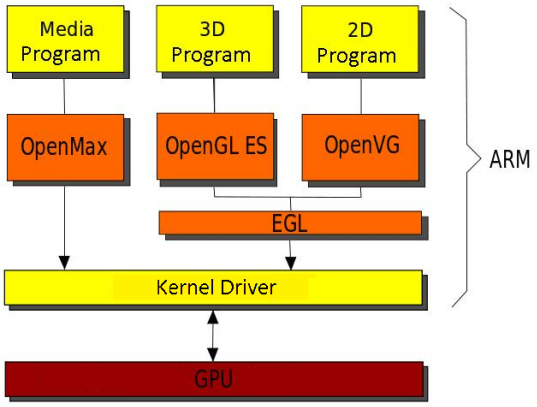

Fig. 3. Raspberry Pi software hierarchy diagram. Adapted according to Ref. [8].

2. Linux embedded operating system

Many operating systems were released for the Raspberry PI to fulfil different tasks; the most popular were listed above. Linux Raspbian OS was used in this work. This system is created in compliance with the GNU GPL licence. Its distribution stems from its progenitor - Linux Debian [9]. Work in the operating system may be carried out via the internet or/and the external monitor.

Writing and compiling of a code is possible on the board itself, which is very convenient - there is no need for external computer resources that should assist in constructing the code before porting it to the board. The compilation is performed like on a simple personal computer, entering a command in the prompt terminal: gcc (for C language), g++ (for C++ language) with the source code file names, object file names and an object connection key "-o". The Raspbian system even supports other *Integrated Development Environments* (IDE), offering a better comfort for programming, one of the IDE examples could be "CodeBlocks".

However, additional libraries, like the <math.h> and other periphery connecting libraries, like the <wiringPi.h> are reached using special terminal keys -lm and -lwiringPi. The peripheral library must be installed separately, and may also differ for different operating systems. The principle diagram of the system, its drivers and application software are illustrated in different layers in Fig. 3.

3. Parallel programming

To control several nodes at the same time, it is necessary to write a parallel programme. Parallel task execution on a system usually commences independently and asynchronously (unless it is a distributed calculation) - unlike in a sequential programme. The operating system, which is installed in an embedded board, takes care of the parallel programme execution to not hinder each other. Each separate programme is a different independent process, which is executed in allocated time on one of the processor's cores. However, one process can be decomposed into separate parallel nodes. Such coordinated operation (avoiding conflicts in data exchange) can be implemented in several ways [10].

1. Interaction via shared memory. Each processor core commences thread execution, which in turn belongs to a single process. Threads exchange their data via shared memory, allocated and common for the given process. This is implemented by using either the features of the programming language itself (like Java or C#), or using the help of libraries (such as used in this work - Posix PThreads for C language), or declaratively (using OpenMP library), either using the embedded compiler tools (like the High Performance Fortran). Such thread implementation demands additional control in thread interaction - Mutex'es, semaphores or monitors.

2. Interaction between thread using message passing. Each processor launches one thread, which exchanges messages with other threads that are executing on other processors. Implementation of such interaction is possible using MPI libraries or language features (like High performance Fortran, Erlang, Occam). Messages can be passed assynchronically or using a *rendezvous* method, when the sender is blocked until its messages are received by another thread.

In this work a library called Posix PThread was utilized for several reasons. Firstly, it is native in the Linux system. Secondly, its thread control is moderately explicit, comparing with an OpenMP, although not as much as in MPI, where one has to take care of each thread's message passing [14]. It is worth mentioning, that launching a few threads, which share variables, it is necessary to utilize mutexes, semaphores and monitors.

1. Mutex. An object/structure, which can be locked in one of the threads, before executing operations with shared variables, and unlocking them afterwards. In another thread, where the same mutex is attempted to be locked, the process halts (optionally) until the wanted mutex is unlocked by the different occupied thread.

2. Semaphore. A mechanism, working just like a mutex, which also has a quota for the variables - one can lock it or unlock it several times. In such case a variable can be reached by multiple threads, as many as is permitted by the semaphore. The set quota for one thread is, in fact, a semaphore's substitution for a mutex.

3. Monitor. An observer which lets the threads not only to wait for their turn to execute, but supplies additional conditions, as well as informs other threads of the changed statuses.

4. Equipment construction

4.1. Tools for probing myoelectric signal

The probing experiment was carried out using electrodes as described in Ref. [11]. These cables are protected from mechanical stresses and breaking, they are made of an nonsol-

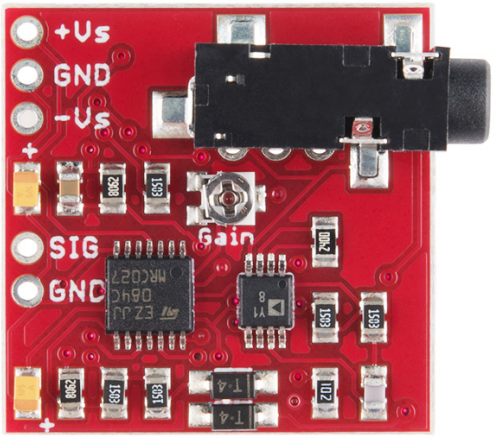

Fig. 4. Muscle Sensor v3 model muscle signal amplifier (manufacturer Advancer Technologies). Adapted according to Ref. [11].

derable metal and are relatively expensive. Disposable mounting stickers are prepared with a conductive gel. They are moulded on the sides of the measured muscle nodes, and the black ground wire on a bony area near the muscles. Muscle signal amplification takes place in a special precision amplifier circuit - see Fig. 3.

The principle of its operation is based on a differential amplifier AD8226 to strengthen microvolt level signals up to a couple of hundred millivolts or even 2÷3 V. The amplified signal is then inverted to the positive voltage, using the instrumental operational amplifier - TL084 model.

After the signal is being smoothed using another circuit node of TL084 model the signal received in such way is not just any set of modulated sinusoids, but rather a curve, which depends on the muscle tension intensity. If need be, the signal may be amplified or weakened at the second-stage amplifier before sending it to another node. A major drawback of this scheme is the implementation of the power supply - it must be powered by a three-pole power supply or a battery (with the plus, minus and ground terminals).

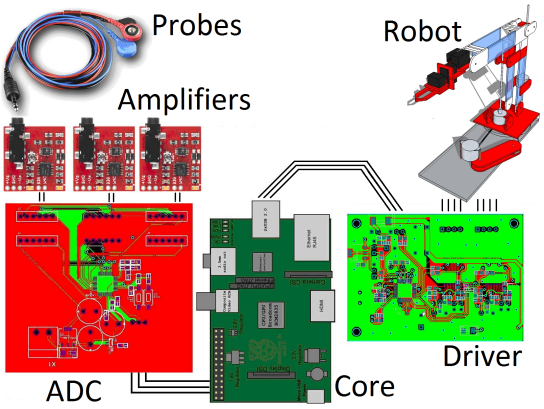

Fig. 5. The basic diagram of the system.

4.2. Main board construction

In order to scan a multi-channel signal, it was decided to construct a main board with a 32-bit ARM Cortex M3 processor (LPC1316). It was chosen because it has a number of pins connected to a 12 bit analogue-to-digital converter, as well as UART (Universal Asynchronous Receiver Transmitter) interface. The final layout is able to support up to six amplifier modules - see Fig. 5, left bottom part. It works as follows: every fifty milliseconds the signal is registered from the amplifier, converted into digital information and forwarded through the UART interface to other nodes. Transmission is constant, regardless of receiving node. Receiving node is described in the theory section - embedded systems Raspberry Pi B and Raspberry Pi 2.

4.3 Assistive mechanical robot

For the research of mechanism control a robot-manipulator E.A.R.L (Ergonomic Assistive Robotic Limb) was created - see Fig. 6. Its parts were manufactured using a laboratory milling machine. The vertical axis rotates due to belt transmission. Shoulder and elbow joints are driven using a worm-gear. These gear systems are powered using stepper motors, while the wrist actuation is handled by servo motors.

To control and drive the stepper and servo motors printed circuit boards were constructed. The operating mode of the above mentioned boards is selected using the UART interface. The operation commands are sent from the embedded system board to the stepper driver boards.

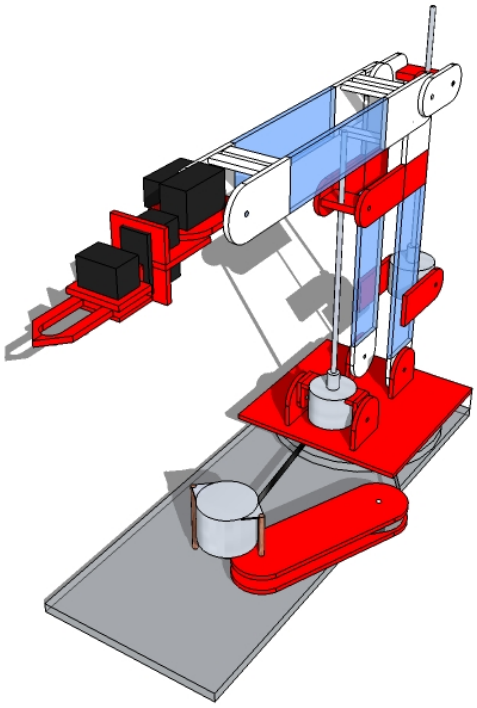

Fig. 6. Prototype of robot manipulator E.A.R.L. which was used to test the neural network control performance.

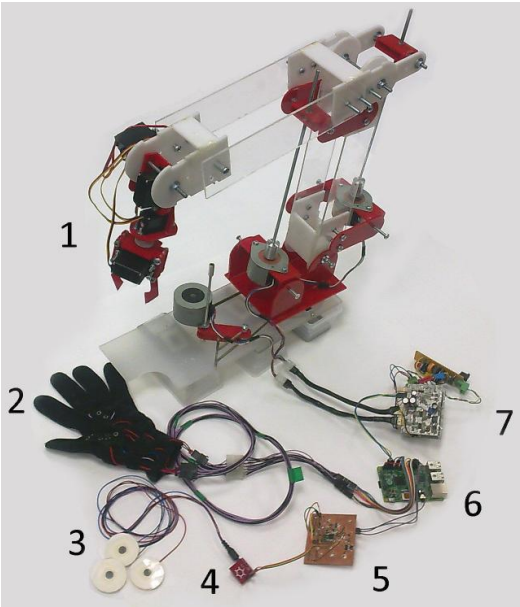

Fig. 7. Complete robotised system including: 1) robot manipulator; 2) remote control glove; 3) myoelectric probes; 4) amplifier; 5) signal converter; 6) Raspberry Pi 2 embedded system; 7) motor drive board.

Additional control is carried out with an ergonomic glove fitted with buttons and if pushed they signal the system.

Combining all the components into a single system, the functioning machine was developed as a hardware-software sollution (Fig. 5) E.A.R.L.

This system consists of muscle signal probes - stickers and the cables. They are connected to the muscle signal amplifier. The amplifier enhances about 50 thousand times, inverts and smothens the signal and feeds it into the distribution unit (main board). The distribution unit sends the data every 50 milliseconds using a command "ADC#XXXX", where # is the channel number and $XXXX$ represents the value from zero to 4095.

The embedded system, using the UART interface, receives and decodes incoming messages using one of the software threads, and depending on the selected programme mode, makes decisions. The remote control unit communicates directly with the embedded system. Its desicions are displayed either on the screen (in case of muscle signal decryption mode) or are sent to the motor control board (in case of robot control mode). The motor control board using a special processor and the UART interface accepts commands from the embedded system and using special driving circuits controls the voltage applied on the stepper motor coils, or the servo motor feedback system.

Fig. 7 represents complete robotised system including robot manipulator, remote control glove and the described electronic equipment.

Conclusions

1. The designed robotic mechanism E.A.R.L. (Ergonomic Assistive Robotic Limb) was succesfully created using several interfaces, suited for myoelectric signal recognition.
2. Package of monitoring program was created, adjusted and adapted for presented interfaces.

References

1. Vadim Gerasimov, Gintaras Jonaitis, Vytautas Jonkus. Application of Artificial Neural Networks for Human Muscle Signal Analysis and Mechanical Equipment Control. 1. Problem overview – *Innovative Infotechnologies for Science, Business and Education,* ISSN 2029-1035 – **1(18)** 2015 – Pp. 7-12.
2. Michael Barr, Anthony Massa. Programming Embedded Systems, 2nd Edition with C and GNU Development Tools. – O'Reilly Media, 2006 – P 336.
3. <http://users.ece.cmu.edu/k̃oopman/iccd96/iccd96.html>, accessed 2015.01.17.
4. <http://elinux.org/RPi_Hardware>, accessed 2015.01.17.
5. <https://www.raspberrypi.org/wp-content/uploads/2012/02/BCM2835-ARM-Peripherals.pdf>, accessed 2015.05.16.
6. <http://www.element14.com/community/community/raspberry-pi/raspberrypi2 ?ICID=rpimain-feature-products>, accessed 2015.05.16.
7. <http://www.adafruit.com/pdfs/raspberrypi2modelb.pdf>, accessed 2015.05.16.
8. <https://www.raspberrypi.org/wp-content/uploads/2012/01/RaspberryArch.jpg>, accessed 2015.05.16.
9. <https://www.raspbian.org>
10. Peter S. Pacheco. An Introduction to Parallel Programming. – Publ. Morgan Kaufmann, 2011 – p. 370.
11. <http://www.advancertechnologies.com/p/muscle-sensor-v3.html>, accessed 2015.01.17.