BLUEMOVE

Using Bluetooth® to Control a YaMoR Modular Robot

Semester Project
Winter 2004-2005

Cyril Jaquier - Kévin Drapel

Supervisors: Professor Auke Ijspeert, Alessandro Crespi, Andres Upegui

February 11, 2005

Abstract

The Biologically Inspired Robotics Group (BIRG) of the Swiss Federal Institute of Technology in Lausanne (EPFL) is working on modular robotics. Computer simulations are mainly used but prototypes also exist. Most of them are controlled by a wired connection. This document explains how the wires can be replaced by a Bluetooth connection on the YaMoR robot units. Wireless communication allows better freedom of movements and ease of use. Although complete autonomous unit is the final aim, a centralised Java application is used to control every module. The problems and solutions met during the development of this software are discussed in the following pages.

Contents

List of Figures

Chapter 1

Project description

1.1 Introduction

The aim of the YaMoR (Yet another Modular Robot) project at BIRG is to construct a wireless autonomous modular robot made of several units. These units were designed by Elmar Dittrich who was in charge of the mechanical aspects (motor, case, materials). Rico Moeckel then worked on the electronics (power supply, Xilinx FPGA and Bluetooth boards), the control software in VHDL for pulse width modulation and the Bluetooth protocol. Despite the fact that a Bluetooth support was available on each module, a real interface between a computer and the modules was still missing. The videos produced by Elmar show several modules without electronics and connected with wires to an intermediate device. This device was acting like a multiplexing bridge between the computer and the modules. Positions were generated by the PC and sent via the serial interface (RS232) to the Bluetooth device on the computer. The Bluetooth signal was then received by the intermediate device which would in turn generate the PWM commands for the servos.

With the new wireless features added by Rico Moeckel, it was obviously necessary to get rid of this middle component and provide a direct wireless communication between the PC and the robot. With a low budget in mind, the solution based on Bluetooth had to be simple and easy to install. This solution is described in Chapter 4. Another point which had to be reevaluated was the software used to communicate with the modules. Jean-Philippe Egger had written a Java application called "Java-Motion" [15] where the user could draw trajectories. Elmar Dittrich slighty modified this application to fit with the modules. We wrote a new application from scratch which is called Bluemove. We used some concepts already present in Java-Motion. We added new features such as a project manager or smooth interpolation. Bluemove will be explained in Chapter 5.

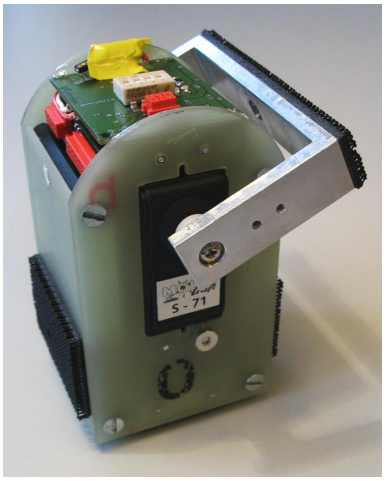

Figure 1.1: A complete module

As Rico left the BIRG after only six months, he did not have time to assemble enough modules. We were in charge of ordering and assembling the rest of the modules. The goal was to mount seven modules to form a robot in a piconet network before the end of the semester. We directly ordered additional components to reach a total of 17 units (plus some "rescue" components from the 18th unit). The delays between ordering and shipping are quite long and at the time of writing, we can not construct new modules. In Chapter 7, we will discuss the problems and solutions related to hardware. We also had to experiment and take videos of the wireless YaMoR with different configurations.

Chapter 2

A literature and projects review

2.1 Bluetooth

As Bluetooth is a complex protocol, we first had to read some documentation. This helped to have a better idea of the advantages and limitations of wireless communications based on this protocol. Knowing the Bluetooth terminology was useful to find the necessary software components and understand the steps that Rico Moeckel went through when he designed the electronics of the modules.

The Chapter 3 is a reminder of the Bluetooth protocol and addresses its issues. We also explain how the specifications relate to the YaMoR module.

There are more than a dozen books about Bluetooth. Most of them share chapters about the protocol. Some books favour low-level and electronic/physical concepts while others are meant to be read by people interested in the communication itself.

We first read some chapters of an in-depth guide to the version 1.1 of Bluetooth protocol without many links to a programming language nor electronics.

The book "Bluetooth: Connect without Cables" [10] covers the whole v1.1 specification and is hence quite complicated. Nonetheless, it is an excellent source for people who are making their way through the Bluetooth protocol.

We skipped some chapters about the radio, power consumption and the low-level aspects which were rather part of Rico Moeckel project and focused on the higher levels of Bluetooth protocol (serial communication, devices discovery, master/slave configuration). "Bluetooth: Connect Without Cables" also features advanced topics like encryption and security which our project was not concerned with.

This book as well as the official specifications

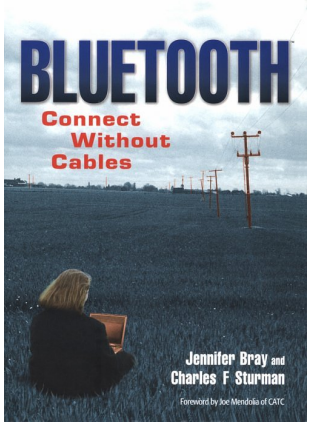

Figure 2.1: "Bluetooth: Connect without Cables"

of the Bluetooth protocol furnished a clarification for the following points:

- rate of Bluetooth communications
- delays related to communication slots and frequency hopping
- limitations in the slave/master architecture
- amount of devices that could be controlled
- range and maximum distance between devices
- the Bluetooth services useful for the modular robot

Some of these points are restricted by the software and the type of hardware which are used (the communication range and the maximum amount of devices in a network may vary).

2.2 Java

After we decided to use Java as the main programming language for the project, we read a book called "Bluetooth for Java" [18]. This document explains the JSR-82 norm, namely the javax.bluetooth namespace which provides a standardized interface to communicate with Bluetooth devices from Java. In Section 5.2.1, we will discuss the different languages and solutions we examined and why we chose Java.

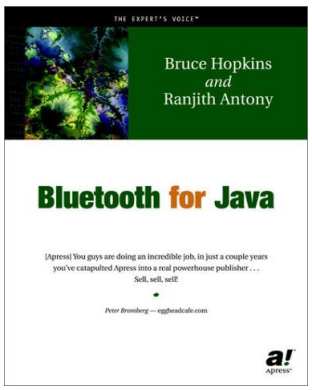

Figure 2.2: "Bluetooth for Java" - APress

2.3 Bluetooth® in robotics

We also looked if other research groups had been working on Bluetooth based robots. Some interesting projects have been developed, mainly in Japan but also here at EPFL. Most of them are humanoid robots. We did not find another project combining modular robotics and Bluetooth. The modular robots either work with wires or radio/IR interfaces. Nonetheless, the few robots below show that Bluetooth is a good protocol for short range communication. It also ensures a compatibility with cellphones.

2.4 Dengoro

This humanoid robot can be controlled using a Bluetooth cellphone. Toshiba, KDDI and I Bee worked on the Dengoro (namecoded "Pirkus R.Type 01") which is made of aluminium and will be sold as a self-assembly-kit for about $1800 (February/March 2005) [12]. Toshiba was in charge of the cellphone part and the operating system, KDDI together with I Bee designed the mechanical and electronic parts of the robot. The robot can walk, jump and wave his hands.

Figure 2.3: "Dengoro" controlled by a Toshiba phone

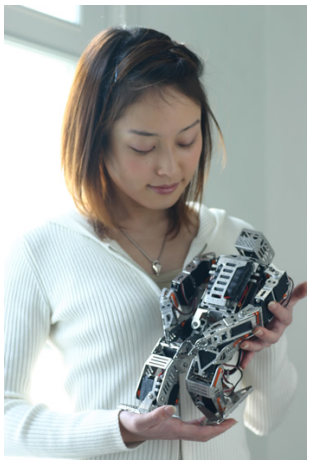

Figure 2.4: "Dengoro"

2.5 Aibo ERS-31xb series

A variant of the famous dog-like robot by Sony supports Bluetooth communication via a remote control shipped by Sony. The other Aibo versions work with IEEE802.11b. The product of the ERS-311b (white dog on Figure 2.5) has been discontinued and replaced by a similar model, ERS-312b (black model on Figure 2.5) [4]. It also uses Bluetooth. As far as we know, these Aibos are only available in Japan and not sold in Europe or the USA.

Figure 2.5: The Sony AIBO ERS-31x family

2.6 Autonomous Systems Lab flying robots (EPFL)

The Autonomous Systems Lab at EPFL works on flying robots. Obviously, wires between the robot and a computer or other devices are not convenient and must be replaced by a wireless solution. They chose to use a serial communication based on Bluetooth.

2.6.1 Blimp

The lightweight flying robot developed at ASL communicates in a wireless manner with a PC [39]. The channel is bidirectional and ensures a short-range link between the microcontroller and the PC. Weight is the main concern and special care was taken to decrease the size and the mass of the components. The robot ceaselessly evolves in a room with black and white stripes on the wall.

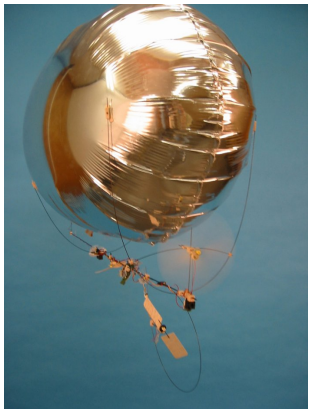

Figure 2.6: The "Blimp" airship

2.6.2 Indoor plane - F2

This Bluetooth plane is extremely lightweight with only 30g. With its autonomy of about 20 minutes, F2 can fly using 2 or 3 linear cameras and additional sensors (gyroscope, accelerometers) [6]. Compared to the Blimp, the plane has more degrees of freedom but can not be evolved in a room. A simulator has been specially developed for this purpose. The communication is similar to the one in the Blimp with a bidirectional channel between the robot and a PC [29].

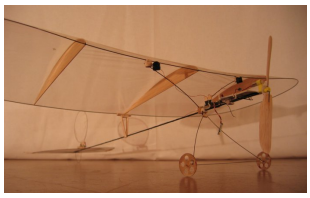

Figure 2.7: The "F2" plane

2.7 μFR-II

A similar project, μFR-II, concerning a micro-flying helicopter (8.6 grammes) with Bluetooth transmission of data (camera snapshots) and commands has been presented by Seiko Epson Corporation [32]. An interesting fact about this robot is that the flight instructions are precalculated and sent via the Bluetooth channel. Moreover realtime control is supported and signals can also be transmitted using two LEDs. Power consumption is, therefore, lowered if only basic signals must be sent at a short range via LEDs. The Bluetooth chip can be switched off after having received the trajectories.

Figure 2.8: The Seiko Epson micro-flying helicopter

2.8 Monsieur II-P

A very small robot with a volume of only 7.8 cm^3 and a weight of 12.5g developed by Seiko Epson Corporation and presented in 2002 [24]. It is the successor of the "Monsieur" robot launched in 1993. Moving at about 7 cm per second when Bluetooth is activated (15 cm per second without wireless), it features a promising ultrasonic motor designed for tiny devices. With their built-in Bluetooth support, these robots can be used simultaneously to create swarms. The Bluetooth and CPU are powered at 2V and the running time is approximately 5 hours.

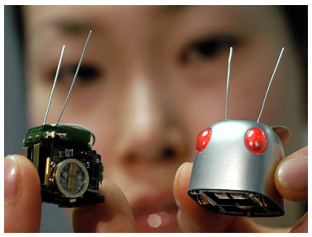

Figure 2.9: The "Monsieur-II" robot

2.9 Morph (Atom Project)

Another humanoid robot is part of an ambitious project supported by the Japanese government at the rate of 50 billion Yen per year over three decades [25]. However, according to some articles and the economical situation, this will be difficult to sustain and the development of the robot could be delayed.

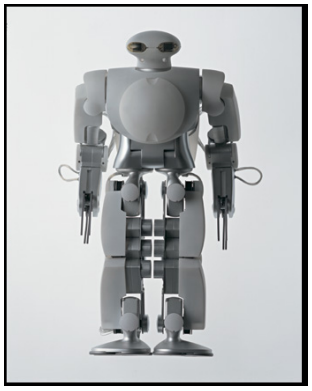

Figure 2.10: Morph 3

The goal of the Atom Project maintained by the ERATO (Exploratory Research for Advanced Technology) is to create an humanoid with the neural and physical capacity of a 5-year-old child [26]. Morph would be used to rescue people in natural disasters. It could also be sent in dangerous or contaminated environments.

2.10 Remarks

There are many other humanoid Bluetooth robots like "Nuvo". Unfortunately, they are either undocumented or the research pages are in Japanese. Moreover, companies are competing each other on the toy market and do not give much relevant information about the technical details implemented in their robots.

Figure 2.11: The "Nuvo" robot

What we can notice is that all these robots except the "Monsieur II-P" are supposed to be used as independent devices. There are no interactions between units to create new behaviours as in the modular robotics paradigm. The Seiko Epson Corporation has demonstrated that it was possible to create Bluetooth robots whose size does not exceed a few centimetres. The "Dengoro" robot is a good example of a robot controlled by a small remote device. Clearly, a modular robot could receive its commands from a cellphone or a PDA instead of a desktop computer.

Chapter 3

An overview of the Bluetooth® protocol

3.1 Introduction

Bluetooth was first developed by Ericsson and then Sony, IBM, Intel, Toshiba and Nokia joined the project. It was intended to provide a cheap wireless support for mobile devices communicating at close range. Bluetooth is said to be a cable replacement for personal area network while 802.11x is used for local area network. Hence, Bluetooth is the best solution when distances between mobile devices are relatively short, high transmission rates are not needed, and power consumption has to be low. With their 1 mW transmission power, most Bluetooth devices operate in a range of a few meters (class 3). There are two other power classes: 100 mW (class 1) and 2.5 mW (class 2). They obviously allow wider coverage, up to 100 meters with class 1.

The first specification was released in 1999 but it suffered from many issues related to interoperability. This was fixed in version 1.1. Support for non-encrypted channels was also part of this update. Bluetooth 1.2 is the version currently used by most devices available, though chips based on version 2.0 are available since October 2004. Bluetooth 1.2 contains major improvements like adaptive frequency hopping, synchronous connections and information about the signal strength and timing. The transmission speed in versions 1.x was set at 723 Kbit/s but in practise, this rate was only achieved with version 1.2. Frequency hopping is probably the most important feature in this revision as it improves resistance to interferences. Actually, the license-free 2.4 GHz frequency band used by Bluetooth is quite crowded. Popular wireless devices and systems (cordless phones, door openers, 802.11 networks) also work in this band and frequency hopping together with a low power signal was a solution to avoid collisions between packets.

Bluetooth devices pseudo randomly switch between 79 channels available in the 2.4 GHz to 2.4835 GHz range. Although in some countries like Japan, the bandwidth is more limited and only 23 channels are available. Hopping is performed every 625 microseconds (1600 switches per second) and is also supposed to improve security. This kind of protection scheme is quite insecure nowadays and has been replaced by robust cryptographic primitives in Bluetooth 2.0.

The second generation of Bluetooth is compatible with previous versions. Among the new features in Bluetooth 2.0 are support for broadcast/multicast, a faster mode which replaces the frequency hopping, data rate of 2.1 Mbit/s and improved response times with less handshaking. Another interesting feature for mobile devices is the power consumption which has been halved in Bluetooth 2.0, the chip itself needs more power than an equivalent Bluetooth 1.2 chip but as transmissions are three times faster with less exchanges, this results in a diminished consumption.

3.2 Piconets and scatternets

As Bluetooth is intended to connect many devices to create small networks, the protocol was designed with a master-slave architecture in mind. Each Bluetooth device can handle up to 7

connections, one per slave device. Only one connection is fully active (transmission), the others are in a special semi-active state. In short, they have an active-member address but are simply waiting their turn. The frequency hopping defines slots of 625 microseconds, the master transmits its packets in even time slots while slaves use odd time slots.

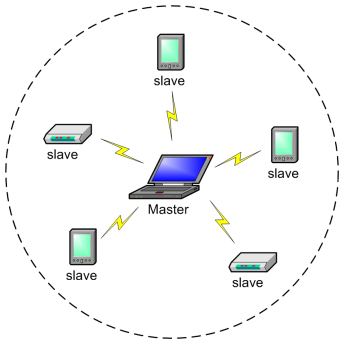

Figure 3.1: A piconet

The master can finally handle 255 standby (parked) slaves. These slaves are synchronised with the master but do not have a specific address. If the master wants to send data to one of these standby devices, he must first discard one of the active slaves and send a notification to the parked slave. It may take some time before the freshly woken up device gets his address and becomes active.

A device can either be a slave or a master and change its state when needed. A master with 7 slaves is called a piconet in the Bluetooth terminology. The master initiates the communication and defines a hopping sequence. It also maintains a clock used for synchronisation between devices. A device can be part of more than one piconet. This allows more complex networks called scatternets. A scatternet is basically a group of piconets. As some devices belong to several piconets, they act like bridges and help to increase the coverage of the network. A device can be the master of its piconet and on the other side, it can be a slave in another piconet. For example, a slave device can send a packet to its master, the host will then forward the packet to the master of an adjacent piconet.

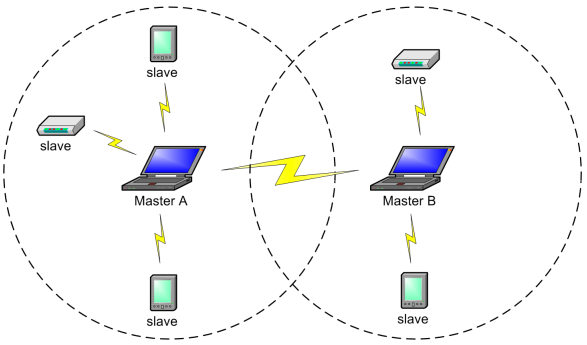

Figure 3.2: A scatternet

3.3 Bluetooth protocol layers and the Bluetooth stack

Bluetooth protocol is complex and for the modular robot project only a small subset of the protocol features is useful. Bluetooth is divided in many layers starting from the hardware/radio layer up to the top layer represented by the application.

There are mainly two types of data transfer available between devices. The first one called SCO (synchronous connection oriented) could be compared to TCP/IP (a connection is maintained between the devices) but is unreliable, it tries to maintain a constant bandwidth. The second is the ACL (asynchronous connectionless) . The ACL transmission rate is higher than SCO rate. The ACL is a reliable communication. Bluetooth also supports a third type of transfer with voice channels for phones.

The host controller interface is mainly related to hardware and responsible for radio, frequency hopping and other low level aspects. The L2CAP layer provides services for connectionless data transfer. Using the SDP (service discovery protocol), one can get information about a device and what kind of services are available on this device (file transfer, voice channels support, modem,...). When one works with a known kind of device, the service discovery, which is quite long, is not necessary. The RFCOMM layer is a serial port emulation. It provides a RS-232 protocol useful for upper level services like OBEX which is a session protocol for exchanging objects. The Bluetooth stack also contains modules for TCP/UDP and can be used as a bridge between Internet and the

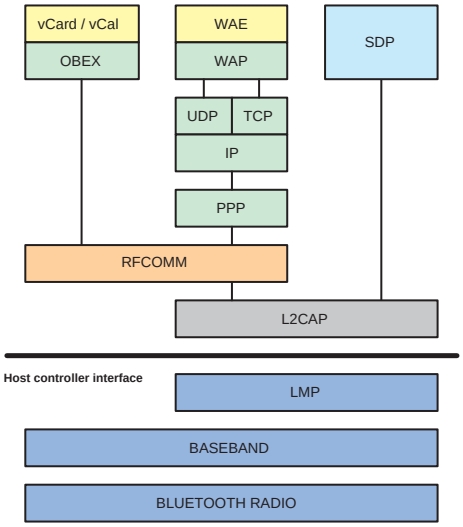

Figure 3.3: Bluetooth protocol - Main layers

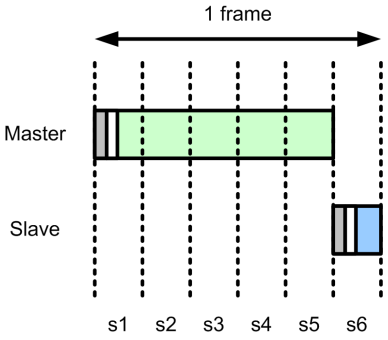

Figure 3.4: A Bluetooth frame composed of a 5-slots packet and 1-slot packet

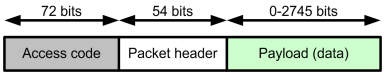

Figure 3.5: A Bluetooth packet

Bluetooth device (a phone cell used as a modem for WAP browsing).

3.4 The Bluetooth packets scheme

Each packet can be composed of multiple slots (one slot has a length of 625 microseconds). Thus, a Bluetooth packet extends itself over 1,3 or 5 slots. A Bluetooth frame consists of a master packet (transmit packet) followed by a slave packet (receive packet). The advantage of multi-slots packets is that they allow higher data rates, the packet header overhead is lower compared to a single-slot communication. With a single-slot communication, the maximum data rate is set to 172 Kbps. With 5 slots for the master and 1 slot packets for the slave, Bluetooth can achieve respectively 721 Kbps and 57.6 Kbps.

The packets have an internal format whic is fixed and consists of three parts. The first header called "access code" has a size of 72 bits. It contains the master identity which is unique as long as communication occurs on the same channel. The access code also contains the master clock. A 54-bits header follows the access code, it features error corrections (for the data and the header), controls and retransmission information. The last part is the payload containing data between 0 and 2745 bits. Depending on the number of bits, the packet will be spread on 1, 3 or 5 slots.

3.5 The Bluetooth implementation in the YaMoR modules

Some of the features of Bluetooth are not necessary for YaMoR. The service discovery is useless, the modules are all the same and their characteristics are known in advance. Encryption and security are aspects which have been left aside for the YaMoR project since they would only add superfluous complexity and decrease the transmission speed.

As the modules work with absolute positions and not relative parameters (deltas), some loss of information from time to time will not have much impact. The ACL communication is reliable but discards packets after a given timeout. If the robot can not be found (out of range), some positions will be flushed out. As soon as it gets closer to the transmitter, it will receive new positions and will synchronize with a correct motion. If the robot is close but a packet has not be received for any reason, the transmitter will resend the packet to the module.

The firmware mounted in the Bluetooth chip was designed by Zeevo and works with a piconet. It does not support scatternet yet, but Rico Moeckel is working on this feature and a custom protocol especially designed for the modular robot.

Chapter 4

Modules

This chapter describes the YaMoR unit. After a short summary of the previous work, we will see the module in more details. This chapter does not present the mechanical pieces of YaMoR but focuses on the electronic part.

4.1 Summary

The YaMoR unit was designed by Elmar Dittrich. One of the main goals was to develop the cheapest module unit possible. An exploded view is shown on Figure 4.1. We will not discuss the mechanical characteristics more in this chapter except on Section 7.1.1. However, we recommend the reading of Elmar's work which is available at [13] for further information.

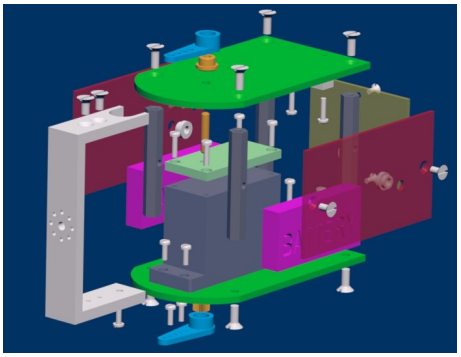

Figure 4.1: YaMoR exploded view

During his summer internship [27], Rico Moeckel developed all electronic boards needed to make YaMoR units autonomous. The modules are controlled using a wireless Bluetooth connection. As explained earlier in Section 2.3 on page 8, Bluetooth does not seem to be widely used in modular robotic. However, this type of communication is a good choice in terms of power consumption and costs which are often decisive in modular robotics. One of our project goals was to build more modules and find an easy way to control them. It allows the user to focus on the development of new kinds of movement rather than dealing with technical issues and an unfriendly user interface. We will now examine in more details each electronic board embedded in a YaMoR unit.

4.2 Power board

4.2.1 Description

The power board supplies the different voltages needed by the FPGA board (+3.3V, +1.2V), the Bluetooth board (+3.3V) and the servo (+6V). A battery taken from the Amphibot I [8] provides the energy to this board. An embedded charger is integrated. It can also be powered with an external power supply. A complete power board is shown on Figure 4.2.

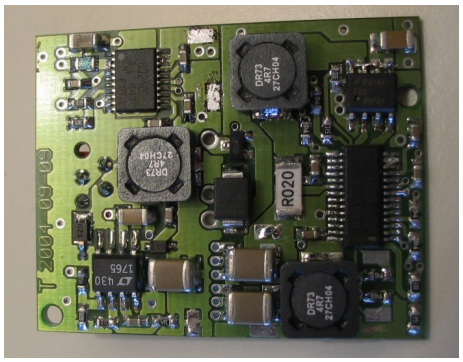

Figure 4.2: Power board

This board takes place at the top of the module. The dimension is 4cm x 3.2cm.

4.2.2 What went right

Once all the problems listed in Section 4.2.3 were resolved, the power board was really easy to use. The dip switch allows the user to effectively control the various functions. The output voltage stability is good thanks to the MAX1774. Once the device was running, we did not encounter further problems because of this board.

4.2.3 What went wrong

MAX1774 problem

When Rico left the BIRG, he had only one power board working. The two others did not work and had the same problem: after connecting the battery, some components' temperature highly increased. Therefore, we started comparing the working board with the two others. After hours of binocular search, we were not able to find any differences. André Badertscher and Fabien Vannel helped us tracking this hardware bug. We exchanged some components, checked all the voltages and currents (see Figure 4.3). Finally, Fabien discovered the source of these problems: the BIN pin was connected to the ground as shown on Figure 4.4. Current flowed across a PMOS transistor inside the MAX1774 because CS- was connected to +3.3V and BIN to the ground. The problem was that the transistor was not designed to work in this direction (see Figure 4.5). The problem was solved by disconnecting the BIN pin.

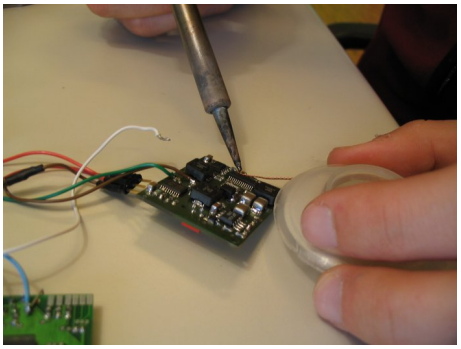

Figure 4.3: Electronic bug tracking

Charger problem

The power board has an embedded charger. It allows to charge the battery without having to dismount the module. The datasheet and Rico's schematic recommend an input voltage of +10V. We applied this voltage of +10V precisely measured with a calibrated multimeter. After a few minutes, the red led started to blink, indicating a charging error. After some investigation, we found out that applying a voltage just below +10V allowed the charger to work correctly. The voltage must be decreased to about +9.5V.

Polarity inversion

The board does not support polarity inversion. If the battery is connected the wrong way, it can result in destroyed components. In order to minimize the risks, André Badertscher soldered a diode between VBAT and GND on a few boards. This protection should be put on all power boards in order to enhance the reliability of the circuit.

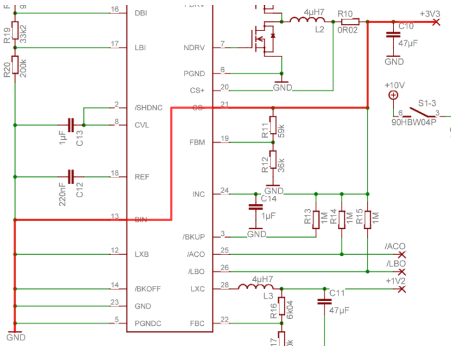

Figure 4.4: Power board schematic

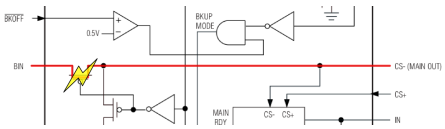

Figure 4.5: MAX1774 internals and the shortcut

4.3 FPGA

4.3.1 Description

The FPGA board contains a Xilinx Spartan 3 with 400'000 gates and 4 Mbit of high speed SRAM. FPGA programming is done using JTAG or slave serial. We currently use the FPGA as a PWM generator which drives the servo. The PWM duty is sent by the Bluetooth

board using a serial line. These duties or positions are sent to the Bluetooth board (see Section 4.4) by an external computer. The ultimate step would be that the FPGA calculates its servo position and transmits it to the others modules. This could be achieved with the implementation of one or more MicroBlaze into the FPGA. An external +2.5V reference voltage is needed in order to program the Spartan.

Figure 4.6: FPGA board

The board dimensions are 7.5cm x 4cm. It takes place vertically along the servo. There is a square hole of 2.4cm which allows the battery to be placed horizontally at the bottom of the module.

4.3.2 What went right

We never had problems with this card. The FPGA programming is easily completed with the Xilinx tools and the JTAG connector. One must not forget to connect the +2.5V voltage reference. The PWM design written in VHDL by Rico Moeckel is used without any modification.

4.3.3 What went wrong

During our tests a module sometimes stopped moving without any apparent reason. The Bluetooth connection was established but the servo did not move anymore. After a reset of the module and FPGA reprogrammation, the module started to work as expected. We were not able to track this problem down as it only occurs rarely and randomly.

4.4 Bluetooth

4.4.1 Description

The Bluetooth board on Figure 4.7 is equipped with a *System on Chip* from Zeevo [38]. This chip (ZV4002) is a combination of a Bluetooth radio frequency system and an ARM core [5]. The board contains 8 Mbit of SRAM and up to 32 Mbit of flash memory. The ARM programming is done by JTAG or via RS232. With its development kit, Zeevo provides a basic operating system called BlueOS with a basic multitasking. The communication between processes are done using messages[1]. Zeevo also provides a test application called "Zerial" which is based on their BlueOS operating system. It allows the replacement of serial cable with a Bluetooth connection.

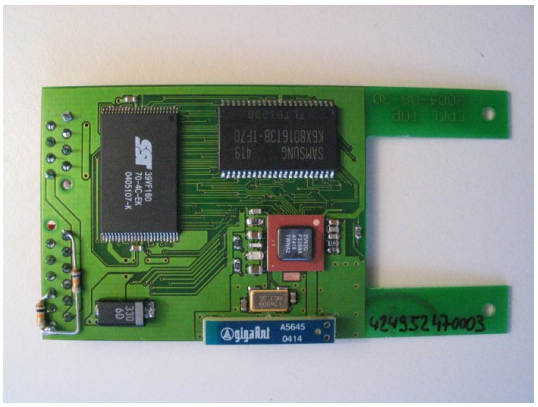

Figure 4.7: The Bluetooth board

4.4.2 Control via an USB dongle

At its early stage, the modular robot was controlled via a serial interface (RS232) with signals generated by the desktop computer and forwarded to its Bluetooth device. The Bluetooth signal was then received by an intermediate device connected to the modules with wires. This middle device generated the appropriate PWM commands for the servos.

We had to remove this intermediate device and provide a simple solution to directly communicate with the modular robot from the computer. A few solutions appeared:

- A Bluetooth stick connected to the USB port of the PC
- Many Bluetooth devices connected to a FPGA which routes data coming from a serial port

[1] also called "signals" in BlueOS terminology.

- A PC with many serial ports, each serial port is connected to a Bluetooth transmitter

We quickly decided to use Bluetooth sticks. It was the cheapest way to get a handy Bluetooth solution. The USB sticks are available in all computer shops for a low price. One must verify that they are compatible with Linux. We bought two different models (class I with 100m range) that perfectly work in a Linux environnement: MSI (MS-6967 on Figure 4.8) and ACER (AG.BTCSR.UD3).

Figure 4.8: the MSI 6967 Bluetooth dongle

A custom Bluetooth device would have been expensive to build and difficult to transport or deploy. It did not provide any significant advantages compared to the Bluetooth stick. The Bluetooth sticks are versatile devices that support all features needed for the modular robot. They are particularly useful for laptops as they are lightweight and small. Compared to the custom device which would have been a big black box with maybe an external power supply, they are the best solution for a seamless integration into a personal computer.

4.4.3 Connection

The connection establishment between two "Zerial" interfaces is achieved using special AT commands. Here is an example:

```
AT+ZV Discovery
```

This command launches a discovery scan and returns all the available Bluetooth devices found in the area. Another example.

```
AT+ZV SPPCONNECT 00043e000000
```

This one connects the current "Zerial" interface to another one whose address is "00043e000000". After the connection setup, the interface switches to "bridge" mode and just forwards the received data.

This firmware is distributed in source code and binary file. This serial line replacement can be used to communicate with the YaMoR units. As we do not want a "Zerial" interface on the computer side, we tried to find a more flexible solution. We discovered that it was possible to make a direct serial connection between a "Zerial" interface and a standard Bluetooth dongle. This is the adopted solution in regard to its flexibility and low cost. We will discuss this in more details in Section 5.2.3. The data flow is represented on Figure 4.9.

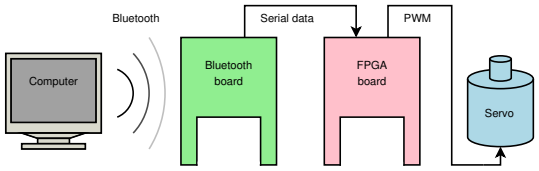

Figure 4.9: Communication flow

The data are transmitted from the computer to the Bluetooth board using the Bluetooth protocol. The packets are then forwarded to the FPGA board which expresses them into PWM values. These pulses finally arrive at the servo which converts them into a mechanical position.

Each Bluetooth device needs to have an unique address in order to be identified without any confusion. The address of the ZV4002 can be statically set with the firmware loader application. It is a one time operation. However it is possible to program the address dynamically afterwards. A Bluetooth address is a 48 bits value. We decided to use the following convention for YaMoR units.

- The address begins with `42495247`.
- The four last positions are the module number.
- `0000` is reserved and must not be used.

For example, module 1 has the Bluetooth address `424952470001`. This is not a random number but the hexadecimal ASCII representation of "BIRG" (`42` stands for `A`, ...). These addresses are statically stored in the ZV4002.

The dimension of this board is 7.5cm x 4cm like the FPGA board. It takes place vertically

along the servo on the opposite side. There is a square hole of 2.4cm which allows the battery to be placed horizontally at the bottom of the module.

4.4.4 What went right

The ZV4002 programming is done using JTAG and the Zeevo flash tool. This operation is easy to accomplish and needs to be done only once. Afterwards it is only required to turn the power on in order to have an operational board.

A green led shows the Bluetooth connection state. It flashes when no connection is established and remains on when connected to our master Bluetooth device. This was really helpful for debugging.

4.4.5 What went wrong

CRC errors

We quickly got three built modules. One was already flashed by Rico with the "Zerial" firmware for his project presentation. The second unit was flashed by us without any troubles. But the third one always reported CRC errors when launching the flash tool. We exchanged the ZV4002 and the flash memory. Unfortunately, the CRC failure still occurred. At this time, we have not yet discovered the reasons of these errors but we highly suspect the flash memory component to be faulty. We have not yet received these memories, further investigations are therefore impossible at the moment.

Chapter 5

Bluemove

This chapter focuses on the software application developed during this project. After a short summary, we will see in more details the different parts of Bluemove, the technologies we used and the mathematical routines implemented in this application.

5.1 Summary

When we started this project, the YaMoR units were quite functional. But it lacked a good software. We thus wrote a Java application that allowed the user to control a robot without having to learn a programming or script language.

Even an unskilled person had to be able to create and play with YaMoR modules. We also wanted to make an application generic enough to be reused in other projects with minimal changes. We will show how we accomplished these aims.

5.2 Java 1.5

5.2.1 Why Java and not another language ?

While we were looking for a software solution, we quickly noticed that Java was the language that provided the most convenient interface. Despite its standard status, Bluetooth has been efficiently implemented only in a small set of languages including Java, C/C++ and C#. C# suffers of portability problems as it is closely related to the Win32 API. We checked if the open source .NET implementations (Mono, DotGNU) could provide the needed services. Their network code being at an alpha stage[1], Mono or DotGNU are still missing vital features.

A solution based on C or C++ was not good either for the portability. We did not find a library that could transparently work on Linux or Windows. The remaining language that we considered was Java 1.5 (J2SE 5.0). It quickly turned out to be the best candidate for the Bluemove project. It is indeed exceptionally well-suited for applications that must be deployed on several operating systems. Moreover, Java is the only language with a standard and consistent interface for Bluetooth (JSR-82). Java via Swing gives a powerful access to graphical functions which were necessary to design the interface. Thanks to the J2ME (Micro Edition of the Java 2 Platform) and its wireless toolkit, it would be possible to create Bluetooth applications for the modular robot on mobile phones, PDA or other small systems that support Java and Bluetooth.

5.2.2 The Bluetooth stack

The Java API for Bluetooth (JSR-82) standardizes an interface for the following protocols:

- RFCOMM
- OBEX
- Service Discovery

An expert group composed of engineers of big companies (IBM, Ericsson, Nokia and Motorola) has leaded the JSR-82 to a robust and consistent specification (released in March 2002). Additional protocols could be added in future versions. However for our project, the RFCOMM (serial ports emulation) protocol is enough.

[1] about 13 classes are missing in DotGNU, Mono seems to be a little more advanced

JSR-82 only defines a set of functions but does not provide any implementation. The latest which is called "Bluetooth stack" is left to the chips manufacturers and other commercial third-parties. Some open source projects have started but their stacks are rather limited, they do not work with the Bluetooth stick connected to the USB port of a PC. Many companies sell more advanced stacks for both Windows and Linux. One of them, Rococo offers a special academic edition of its Bluetooth stack. Moreover, it is free for universities and works under Linux. Their development kit called "Impronto" just needs a text file license which is stored in the same directory as the application.

The specifications and the good documentation ensured that it was the perfect stack to start our project. If another stack is released by another company or targeting another operating system, we could easily replace the Rococo stack. It basically behaves like a plugin. The Rococo stack contains some helpers for advanced Bluetooth functions but we did not use them because they would break the JSR-82 compatibility. Some of these extra functions furthermore adds a layer of complexity. They are unnecessary for a serial communication to devices whose specifications are known in advance. For instance, the services discovery which is essential for mobile phones could be left apart in Bluemove.

5.2.3 Transmission between Bluemove and the module

Communicating with a module is a matter of sending data to a specific URL starting with "btspp://" and followed by the address of the device in hexadecimal. The btspp prefix stands for "Bluetooth serial port profile". The firmware which is mounted on the Zeevo chip accepts a serial communication. After the Bluetooth headers, a data containing two bytes is appended. The first byte indicates a module ID, the second byte is the servo position. The system was designed with a broadcasting support in mind. This turns out to be useful when all modules are reset to the same value. The module ID is set to 255 for a broadcast (all modules receive the same packet). Another value indicates a packet forwarded to one and only one unit according to its ID.

In the current version of Bluemove, the ID is always set to 255 but a packet is transmitted to its final destination using the btspp URL; no broadcasting is used.

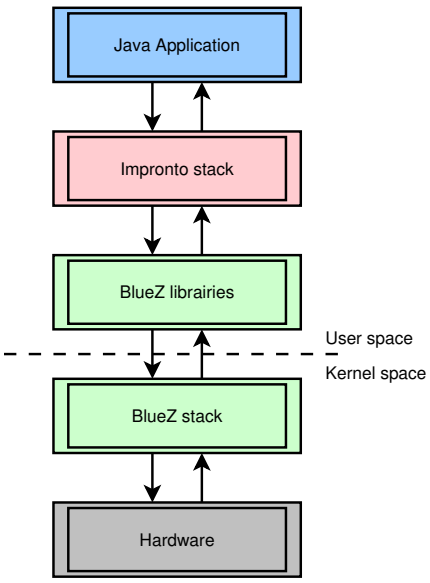

Figure 5.1: From Java application to the Bluetooth device

Figure 5.1 represents the communication flow from the Java application to the Bluetooth device. The Impronto stack does not directly interact with the hardware device. Instead, it uses the BlueZ [1] libraries which then interact with the kernel Bluetooth drivers. With this scheme, Rococo Software do not need to rewrite code already present in BlueZ and hardware dependent.

5.2.4 What went right

The Rococo stack

We did not have much problem with the Rococo stack except some unsuccessful scans that randomly happen (devices are not detected at all). It is hard to tell whether they are caused by the Rococo stack or the underlying Linux system. In such cases, it is recommended to reset the Bluetooth device in Linux using the `hciconfig` command or to call `hcidump` to check the traffic on the Bluetooth device. If the problem persists, the best is to reset the kernel modules as follows.

```
# rmmod hci_usb bluetooth
# modprobe bluetooth hci_usb
```

The Java namespace which is in charge of the Bluetooth implementation is extremely well-designed and was straightforward to use. The Rococo stack is transparently integrated in the application and sending data is a matter of a few functions calls.

New features in Java 1.5

We used the latest version of the JDK, J2SE 5.0 which is in fact Java 1.5. Useful features that were previously missing in the Java language have been added in the new version [23]. Among all these major updates, we used the following features which had a significant impact on the ease of development:

- generic types
- autoboxing
- enhanced "for" loop
- enums

The generic types are mostly present in code dealing with Collections (lists, hashmaps, etc.). In previous versions of Java, the Collections only contained objects of type `Object`. It was up to the programmer to insert casts when he retrieved objects from a collection. This kind of syntax was error-prone and could lead to arduous debug sessions. With generic types, an additional constraint is appended to a collection, it must contain objects of a given type and casts are not needed anymore.

The autoboxing avoids clumsy conversions between primitive types (int, float, double, etc.) and their equivalent Object counterparts (Integer, Float, Double, etc.). Previously to Java 1.5, one could not store primitive types in collections. The programmer first had to convert the primitive to its Object with code similar to `list.add(new Integer(1234))`. With Java 1.5, this extra code is not needed anymore: `list.add(1234)`. The opposite is also true. In Java 1.5, a primitive retrieved from a list does not have to be explicitly casted.

The enhanced "for" loop was already present in most high level languages such as C#, Python or PHP, though often named differently ("foreach" keyword). It is a powerful syntax that avoids the creation of "iterators" when one just wants to read all elements from a collection. The code looks like: `for (Integer i : list) {...}` and is probably the most useful feature among the changes from Java 1.4 to Java 1.5.

The enums that all C or C++ programmers have at least used once, are often used to define a set of choices. Prior to Java 1.5, the enums had to be simulated through a sequence of `static final int`. It was tedious to write and a value had to be given to each integer. In Java 1.5, an `enum` keyword has been added. Such a definition looks like this: `public enum Choice { ok, wrong, cancel };`. Enums can be used in switch/case blocks and are considered as objects.

5.2.5 What went wrong

The Java 1.5 compiler

We faced serious problems with the Java 1.5 compiler[2]. Our mathematical code raised strange errors with signs in floating operations. For example, a float or double variable was supposed to be negative but the executed code discarded the sign. This ended up with totally wrong results in the interpolations and timelines. It was actually hard to track this problem as it seemed to depend on external conditions like the size of the code, previous recompilations or changes in other classes.

After patiently investigating the different steps that the compiler went through, we concluded that the issue was not caused by the virtual machine or the operating system as it also occurred on a Windows XP platform.

We disassembled the generated bytecode and noticed that the compiler simply removed the signs of some constants. Code such as `float f = -2.0f + a;` could be interpreted like `float f = 2.0f + a;` once compiled. One solution was to move the constant at the end and add some parenthesis: `float f = a - (2.0f);`. Another possibility was to use the object counterparts of the concerned primitive. The code would then use declarations like `new Float(-2.0f)` but this led to complex code in

[2] all 1.5.x were plagued by this bug, for Bluemove, we used build 1.5.0_01-b08

mathematical routines. We tried to avoid the latest solution as much as possible.

We filled a complete bug report and sent it to Sun. Unfortunately, we did not get a response. We hope they will fix this problem as soon as possible as it may lead to serious bugs and unstable applications.

JBuilder designer

JBuilder comes with an integrated GUI designer (see Figure 5.2). This tool allows the user to draw the application interface on screen instead of coding it. A lot of time is thus saved. Moreover, coding a GUI interface can quickly become unmanageable when the complexity of the interface grows. However, we faced some problems with the JBuilder designer.

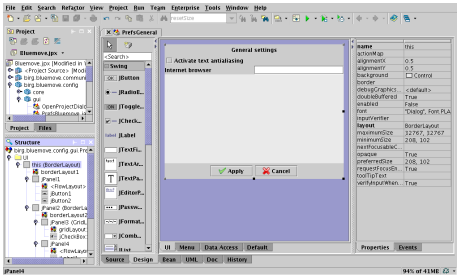

Figure 5.2: JBuilder designer

When starting the designer on a newly created class, it worked as expected and was really helpful. But as soon as we started to add our own code, especially constructor parameters or class variables with direct instantiation, JBuilder crashed. The returned error was the following.

```
An unexpected exception has been
detected in native code outside
the VM.
```

This happened under Windows and Linux and was a really annoying problem because unsaved file modifications were lost. We were forced to modify GUI code directly by hand.

We tried other GUI designer such as JFormDesigner [33] or IntelliJ IDEA [20] but each one has its own code generation style which did not integrated well in JBuilder. So we continued to use JBuilder designer and took the practice to save our files regularly. For big GUI modifications, we copied the JBuilder generated code to a new class file, then worked on this version. We finally copied the newly generated code to the original class. Note that the "JBuilder 2005 Update 1" did not solve the problem.

5.3 Interface

5.3.1 Description

"Java-Motion" by Jean-Philippe Egger was the software modified and used by Elmar Dittrich and Rico Moeckel to generate trajectories on the robots shown in their videos. Unfortunately, Java-Motion lacks many features and somehow looked too limited for the modular robot project. One of the main drawbacks in the version used by Elmar was the fact that only six modules could be managed. Moreover, Java-Motion did not provide a tool to draw smooth curves, the trajectories were linearly interpolated.

We preferred to focus our efforts on a new software that used some ideas present in Java-Motion but with a drastic change in the interface and flexibility. Bluemove was written from scratch because our new features would have been incompatible with the code present in Java-Motion. It was more efficient to rewrite things instead of fixing them. We also wanted to create an application that could be used with other robots, not only the Bluetooth modular robot. If we did not go as far as a plugin system, we still have a flexible system split in several independent parts. A given part could be replaced or slightly modified to fit with constraints and specifications of another robot, protocol, etc.

As we had a good experience with Java, Swing and its graphical functions, we aimed at a nice and convivial GUI with custom controls when necessary. This was particularly true for the timelines which were simply too difficult to create with the standard Java controls. We also tried to find the best solution for each problem related to human interaction, we wanted Bluemove to be simple and that the user could play with the main functions without even reading the documentation.

As said above, we took our inspiration from Java-Motion and added ideas that we had seen in other softwares. Systems based on timelines are present in many applications. Most of them deal with video or audio editing.

Here is a non-exhaustive list of such softwares:

- Demopaja by Moppi Productions (video)

- 3DSMAX by Discreet (3d)
- After Effects by Adobe (video)
- Sonar by Cakewalk (audio)
- FruityLoop by FL Studio (audio)

Among the different applications cited in the above list, one of them had a strong impact on Bluemove: Demopaja [30]. It is a free realtime-animation authoring tool similar to Macromedia Flash and Adobe After Effects [3]. The interface is based on a tree. Its nodes can be expanded to display different types of timelines. These timelines contain trajectories with linear, smooth and hold keys, segments of curves can be displaced or deleted.

Demopaja provides curves for different objects: colours, scalars, ranges, etc. Another concept that we did not have time to implement in Bluemove but which is present in Demopaja is what they call "time segments". It is, therefore, possible to enable or disable some parts of the sequence.

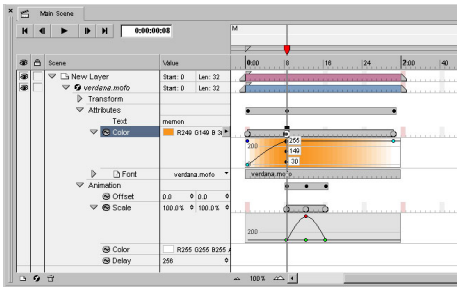

Figure 5.3: Demopaja by Moppi Productions

As stated by their authors, Demopaja is strongly based on Adobe After Effects. Looking at a screenshot, we see many similarities in both the interface and the ideas. There are clear advantages of such interfaces. They are straightforward to use and if they may look eye-candy at first, they are still the easiest way to define and work with curves.

5.3.2 The different tabs

The current version of Bluemove is split into three tabs:

- the timelines manager
- the modules manager
- the console

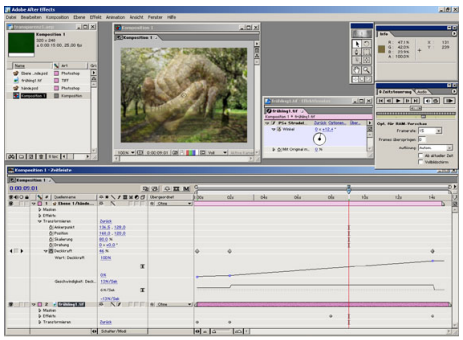

Figure 5.4: Adobe After Effects

The timelines manager

The timelines manager is in charge of the trajectories for all modules that were loaded in the modules manager (from a previous project stored on disk or a new project). It is composed of three parts which are custom Swing controls (modules tree, timelines tree and timescale). The left part is a summary of the project and its modules (name, range, address). This panel is in fact a tree whose nodes can be expanded or collapsed. The user can also disable a module or a subcomponent such as an actuator or a sensor[3]. When a component is disabled, it turns red. When activated, the panel is green. When expanded, the components available in a module can be resized. The user just has to drag the cursor over the grey bar at the bottom of the component zone.

Changes in the left panel are propagated to the right panel that contains the curves. The trajectories can be drawn using the mouse. Clicking on the left button will add a new key. The key can be moved with a drag and drop (pressing the left button and keeping it down while moving the key). There are currently two types of keys available in Bluemove:

- Linear key
- Hermite spline key

Both can be mixed within the same curve, the interpolator will properly compute the trajectory. By default, the user works in a "spline mode". To add linear keys, he must select the "linear keys tool" icon in the icons bar. He can switch back to splines keys with the "spline keys tool" icon.

[3]sensors are not available in the current version of Bluemove

A right click will bring a popup menu. If the user right-clicks on a key, a key menu will appear. It features actions like deleting a key or converting the key to the opposite type (to a spline if it's linear and vice-versa). A right-click at another place (not on a key) will bring a global popup menu. The user can delete all keys or converts them to the same type (linear or spline). Bluemove does not provide more advanced features yet such as selecting some keys, moving or deleting a whole set of keys at once. These features are planned for the next version.

At the bottom of the timelines manager, a timescale is available. It gives information about the frames and the current frame position on the curve. This position is represented by a small triangle moving on the timescale. Dragging the mouse cursor over the timescale will shift the curves in time. The user can also scale the time range using the zoom-in and zoom-out icons.

On the lower left part of the timelines tab, the player zone and its "play/pause", "stop", "repeat" buttons allow the user to send the data to the robot, interrupt the sequence or enable/disable looping of the whole sequence. The end of the sequence is represented by the very last key in all active trajectories. This is something to consider when designing the trajectories.

The modules manager

Bluemove has been built with the idea of storing everything into a compressed file that we call "project". The project contains the information about a robot, its modules and the trajectories. In the modules manager, one can modify the different parts available in a project (see Figure 5.5). The modules manager acts like a serialisation and management system; it drives all routines related to loading and storing data. The user can add, modify or remove modules. He can also write a description about the robot and upload a picture. It is indeed quite useful to have a sketch or a real picture of the modular robot when mounted. The modules manager is composed of a tree on the left side and a panel for settings and information on the opposite side.

A module contains actuators or sensors; the interface provides a way to deal with all these parameters. The user can add or remove these components and change their settings (such as the range for an actuator).

By pressing on the "Apply" button, the user saves the changes to the local version of project. The project can later be saved on disk using the menu bar or the save icon in the icons bar. It is obviously possible to load a previous project from the menu or the open icon. All parameters will be restored as well as the trajectories. A new project can be created using the new command in the menu or the new icon.

The modules manager is synchronised with the timelines manager. If a module is removed from the modules manager, the timelines will disappear in the timelines manager. The same applies for the actuators and sensors. Changes in the modules manager (settings or new components) will be notified to the timelines manager.

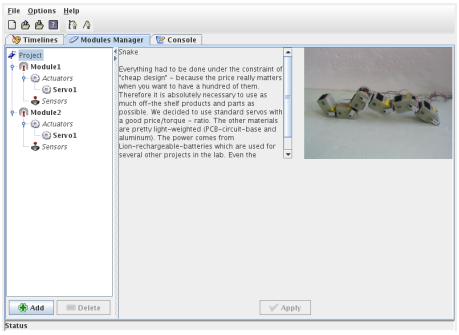

Figure 5.5: Modules Manager

The console

The console tab (see Figure 5.6) displays the log messages generated by Bluemove. There are five levels of messages:

- DEBUG: messages useful in order to debug the application
- INFO: information messages
- WARN: a problem leading to an error occurred
- ERROR: something that should not have happened
- FATAL: unrecoverable error

Each level is displayed in a distinct colour with its specific icon. This allows the user to quickly diagnose what appended in the application.

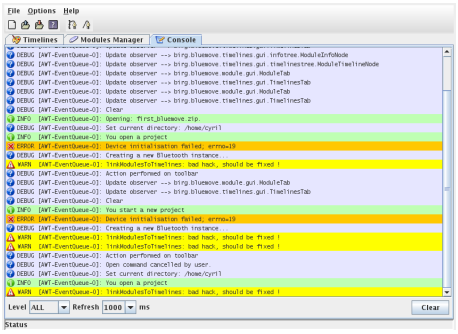

Figure 5.6: Console

5.3.3 What went right

The small learning curve for the end-user

The timelines manager is a nice set of custom controls. It is easy to use and quite powerful. The learning curve is small and the user can quickly add or delete keys, move them around. The integration with the modules manager is also quite robust thanks to a last-minute "intensive software engineering session". The user will have a summary of the whole robot with a quick glance at the left part of the timelines manager. As everything is based on a tree, the interface space is optimized and not too crowded.

The console

We did not encounter many problems with this part. We only noticed that it could slow down the application and the communications with the robot if a large amount of messages was printed. The filter option is hence quite useful in such cases. Printing a message takes about 15 ns, this delay is insignificant compared to the rest of the routines executed during runtime.

5.3.4 What went wrong

Writing bugs free and clean Swing controls are tedious

Writing complex Swing controls is quite cryptic and error-prone. There are many concepts to deal with, including timing, refreshing, performance, interaction with the user (mouse events). It is quite time consuming because one has to write everything from scratch. Bluemove still has some minor issues related to the timelines but it is sometimes hard to say whether they are caused by Java or a bug in the application. For example, when the user wants to resize a module component (left part of the timelines manager), the cursor is supposed to change when the mouse is over the grey rectangle. When the application is in fullscreen, this does not work anymore. Coming back to a windowed mode fixes the problem. Such a minor bug could also be caused by the Linux windows managers (metacity, fluxbox, enlightenment, etc.).

When we started writing the code for the timelines, we quickly noticed that it would be difficult to come up with a clean solution. A first version of the code was not satisfying and we decided to rewrite it with a slightly better architecture. Despite the attempts to design a good structure using our knowledge in software engineering, we are pretty confident that it could be improved and simplified if we had more experience with Swing. The most complex part was to write the code related to the tree system (expanding, collapsing components), there are still a few bugs that may appear from time to time but none of them is critical.

Attempts to tweak standard Java controls failed

In order to improve the interface, we tried to use the JTree control (tree control of Swing) for the settings. We wanted to achieve something similar to what is proposed by .NET (Figure 5.7) or the L2FProd components [11]. These controls provide rows with various settings like color pickers, text, number, sliders, etc. Unfortunately, the JTree and JList controls of Swing are rather limited. It is impossible to resize a single row without a large amount of obscure code. After extensively searching on Google, we came to the conclusion that everybody had this problem but only a few bad solutions had been proposed. It was also difficult to add controls inside the row of a JTree/JList, though not impossible. Most of the suggested hacks have a significant impact on performance and we preferred to keep a fast and clean code. The L2FProd could have been a solution but it lacked a real documentation. As the settings window was not an important part of Bluemove, we did not waste too much of our time on this issue.

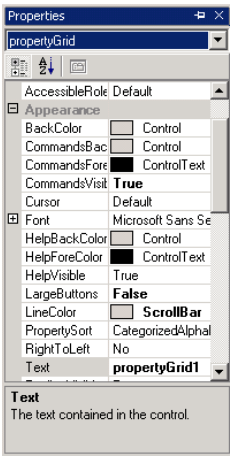

Figure 5.7: .NET properties

5.4 The inners of Bluemove

In order to make a versatile application, we designed Bluemove in such a way that the different parts were as independent as possible from each others. The modules manager could be modified to support for example an I^2C communication instead of Bluetooth without big changes in the timelines manager. However, there are still some links between all these parts, we did not try to make a system based on plugins. It would have been more complex to handle but we think that features like scripting could replace parts of the advantages provided by plugins (more on this in Section 7.2).

5.4.1 An overview of Bluemove architecture

On the code side, Bluemove is composed of two central parts: the timelines manager and the modules/project manager. They are themselves split into two conceptual groups: the core and the GUI. The core is in charge of the logical and system operations; it is used by the GUI. The GUI acts like a bridge between the user and the core.

In Bluemove, the real "brain" is the modules manager core. Its purpose is to link the timelines, the modules, the communication part and the rest of the settings. The modules manager is in charge of serialisation and persistence (loading/storing in files).

The timelines core maintains the curves, this includes algorithms dealing with keys, interpolation and other mathematical routines. The modules manager GUI just needs to call the methods provided by the various objects attached to the modules manager core. On the other side, the timelines manager gets its information from the modules manager. When the timelines manager asks for the curve X of the module Y, the modules manager will query the timelines core. The latest will provide the right curve that will be transferred to the timelines GUI via the modules manager.

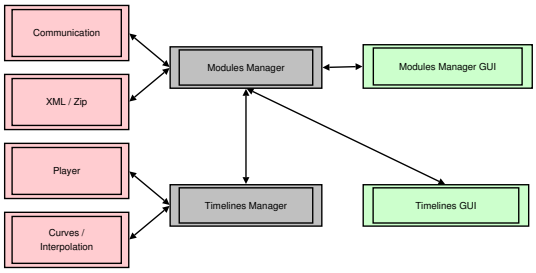

Figure 5.8: A schematic view of Bluemove architecture

The Bluetooth communication is initialised by the modules manager after a request by the timelines GUI. The modules manager delegates this threaded task to a subgroup of objects specialized into communication and Bluetooth.

To sum up, there is a clear cut between the GUI and the underlying core. We could easily swap two different user interfaces while keeping the same core.

We invite the reader to look over the Bluemove programmers documentation for UML schema and additional information about the methods, objects, and namespaces.

5.4.2 Linear and spline interpolation

Bluemove can compute linear and smooth trajectories. Each trajectory is made of several keys. A key is itself composed of two scalars: a frame [4] and a value. Two keys can not share the same frame as this would lead to an inconsistent temporal state. Moreover, a stronger constraint has to be respected when interpolating a trajectory: it must be a function in the mathematical sense. For the same frame, we can not have more than one value. This is a problem when working with splines that are usually evaluated

[4] in this section, time or frame are equivalent, they are proportional to each other

in a two-dimensional plane. They are not functions but parametric curves that can overlap. For the modular robot, we needed interpolation methods, not approximations. Indeed to ensure correct motions, we had to be sure that the servo is at the correct position at a given time. This is not guaranteed by an approximated trajectory. We will now discuss how we solved these various mathematical issues.

Linear interpolation

Linear interpolation is straightforward to implement. Assuming that at least two keys A and B are available (with $A_t < B_t$), we can compute any intermediate trajectory position y at time t ($t \in [A_t, B_t]$) using the following formula:

$$y(t) = A_y + \frac{t - A_t}{B_t - A_t}(B_y - A_y)$$

The main drawback of a linear interpolation is that the first derivative is not continuous. The consequence is a complete absence of smoothness in the generated trajectory.

The smooth interpolation problem

For the modular robot, it was necessary to have at least one type of interpolation that provided smooth curve. By smooth, we mean trajectories with a continuous first derivative (speed). This avoids the sudden changes of speed that appear with a simple linear interpolation and produce jerky motions.

The problem of smooth interpolation has been widely covered in the numerical analysis field. Among the many methods developed throughout the last centuries, Lagrange's interpolation is a famous way to pass a polynomial of degree $N-1$ through N points. However, this method is not convenient for most purposes because it introduces large oscillations. Moving or adding a single point on the curve will radically change the shape of the whole curve. It is actually quite slow to compute and is consequently not a good interpolation scheme for a servo trajectory. Another method, the Chebyshev approximation minimizes the oscillations but as stated by its name, it's an approximation not an interpolation and the errors introduced by such a scheme are not acceptable.

What we can conclude after playing with high-order polynomials is that they are very sensitive to small perturbations in the keys positions and lead to unwanted oscillations (Runge's oscillations).

Piecewise interpolation schemes

Another way to solve the interpolation problem is to split the final trajectory into many pieces. Such methods are called "piecewise interpolations" and most of them are based on "splines"[5]. A piecewise interpolation is a good way to control local conditions and only modify a part of the curve.

At this point, we can either use Bezier curves or Hermite splines. There are other types of splines such as B-Splines but their names are confusing because they are approximation schemes.

It would have been possible to use Bezier curves but one must be careful with the positions of the control points. In the case of a cubic Bezier curve, four control points are needed. The first and last points are placed at the keys locations. The second and third points act like tangents. The curve is always enclosed by the convex hull formed by these four points. The curve reduces to a function under a strict condition on the X axis: the second and third control points must not be moved before or after the first and fourth points.

To keep a continuous first derivative between two Bezier curves, it is necessary to align the third point of the first curve with the second point of the second curve. This adds some complexity when the user moves a point since the tangents symbolized by the two middle control points in both curves must satisfy the constraints described above.

Hermite splines

Hermite splines do not exhibit this control problem though it is more difficult to ensure that the parametric curve will not overlap in a "S" shape and hence lose its function properties. On the other side, the Hermite splines are simple to implement and in our case, it was easier to get a continuous first derivative compared to Bezier curves. We thus used Hermite curves in Bluemove.

From the mathematical point of view, an Hermite curve (often called "cubic spline") is a parametric curve expressed by a parameter s with $s \in [0, 1]$. It is constructed using a set

[5] "smooth lines"

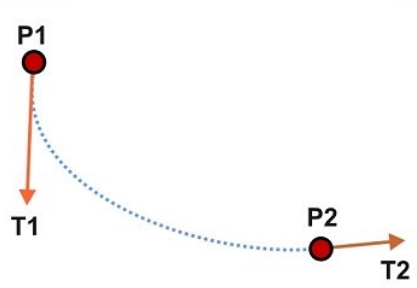

Figure 5.9: An example of an Hermite spline

of four third-order polynomials which form the Hermite basis functions as seen on Figure 5.10. An Hermite curve is composed of four vectors: a starting point and its tangent, the ending point and its tangent. The four polynomials are also called "blending functions". We will refer to them using the notation $H_1(s)$, $H_2(s)$, $H_3(s)$ and $H_4(s)$.

$$H_1(s) = 2s^3 - 3s^2 + 1$$

$$H_2(s) = -2s^3 + 3s^2$$

$$H_3(s) = s^3 - 2s^2 + s$$

$$H_4(s) = s^3 - s^2$$

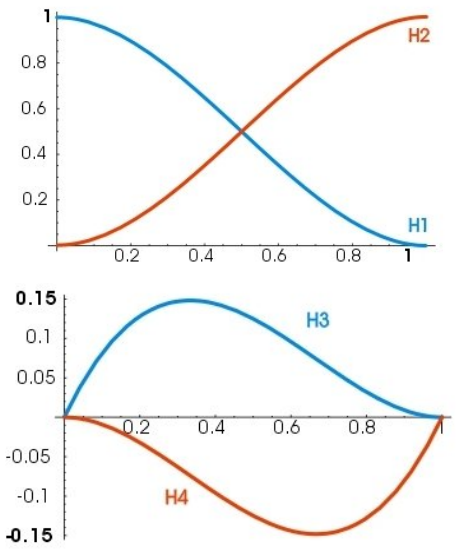

Figure 5.10: The Hermite basis

Each of the four vectors is multiplied by its respective Hermite function. The polynomials are first evaluated using the parameter s. Next the vectors are weighted and summed up to get the final position. The starting point is multiplied by H_1, the ending point is multiplied by H_2. The tangent for the first point is weighted using H_3 and the second tangent using H_4.

Assuming we are working in the two-dimensional plane and the starting point is $P1$, the ending point is $P2$, the tangent for the starting point is $T1$ and the tangent for the ending point is $T2$, we obtain the following set of parametric equations:

$$x(s) = P1_x * H_1(s) + P2_x * H_2(s) + T1_x * H_3(s) + T2_x * H_4(s)$$
$$y(s) = P1_y * H_1(s) + P2_y * H_2(s) + T1_y * H_3(s) + T2_y * H_4(s)$$

With s at 0, the returned point will be $P1$. With s at 1, we will get $P2$.

Computing the tangents

There is a simple way to compute the tangent for a key $\vec{K}_n$. One just needs to retrieve the position of the previous and next keys: $\vec{K}_{n-1}$ and $\vec{K}_{n+1}$. The tangent for $\vec{K}_n$ is the vector $\vec{t} = (\vec{K}_{n+1} - \vec{K}_{n-1}) * \alpha$ where α is a smoothness factor usually between 0 and 1 (a higher value will give a smoother curve).

There are special cases at the start and end of the trajectory. When the next key is not available, we can compute the tangent using the previous and current key. The similar thing can be performed to find the tangent for the very first key. In that case, the tangent is the vector between the current and next key.

Converting an Hermite spline to a function

With the description in the previous section, we supposed that the curve was evaluated in a two-dimensional plane. It is not a function since the same value on the x axis could have more than one position on the y axis. For the trajectory, we had to find a way to convert the Hermite spline to a real function. Moreover, the parameter s defines the position along the curve. It is not related to the x axis which is the time axis in our case. To find a s parameter given a time value along the horizontal axis, we did not use an analytical solution but a simple numerical method: the bisection method.

To find a parameter s for a given X (time value), we first compute the position on the curve with $s = 0.5$. If the value $x(0.5)$ is above X, it means that the parameter s is too high. In such a case, we compute the middle value of the range $[0, 0.5]$, we obtain 0.25 and compute the curve with the parameter $s = 0.25$. Otherwise, we compute the middle value of $[0.5, 1.0]$ which is 0.75. The algorithm recursively evaluates the position on the curve and splits the range in two smaller parts. A stopping condition is met when N iterations have been done or the error is below a given threshold.

For Bluemove, 12 iterations are enough for a good accuracy. The resolution of the s parameter is fully determined by the minimum step size which is $\frac{1}{2^{12}} = 0.0002$.

5.4.3 Persistence

As almost all softwares, Bluemove needs to save and retrieve its data from disk. This is called persistence. It can be done using data bases, XML files, binary or text files. In Bluemove, we chose to use the XML standard for persistence. XML is widely used today and lot of libraries are available. One of the best of them is Xerces [17]. It provides a DOM [35] implementation which is exactly what we need. DOM is an API which allows the creation of XML tree with simple functions. We also used Xalan [16] which provides XPath [36] features. XPath is a language which allows retrieving nodes of a XML document without complicated XML parsing. Both of Xerces and Xalan are free softwares.

When saving a project, a DOM tree is created with all the modules parameters such as module name and address, components and sensors. The DOM tree is transformed afterwards into a XML file using Xerces serialization possibilities.

When opening a project, the module manager is regenerated by reading the XML files. This is done using XPath. A disadvantage of XPath is its slowness. However we do not handle large XML files so this is not noticeable.

The timelines are treated in the same way; they are just stored in a separate file. This allows a better reliability and separation of concepts.

As explained at the beginning of Section 5.3.2, a Bluemove project is a compressed archive. The modules and timelines XML files are thus packed in this archive. We used the ZIP format because it is widely present and Java has an API for handling such files. Below is a typical project archive content.

```
config/timelines.xml
config/modules.xml
Created on 19 01 2005 - 16 23 09
images/model.png
```

Note that there is a file called `Created on 19 01 2005 - 16 23 09`. This is an empty file created the first time that the project is saved. It is indeed impossible to create an empty ZIP file with the Java API.

Chapter 6

Experiments

6.1 Two working modules and an inactive skeleton

As we only had two fully mounted modules, we had to play with this limitation in mind and try to find interesting configurations. Looking at the videos produced by Elmar Dittrich helped us to find some robots based on two working modules and a few inactive units.

Even with the most basic configuration, we noticed that the smooth interpolation offered nice motions compared to the jerkiness of the linearly interpolated trajectories. Using Bluemove, we were able to quickly test various curves. Most of them are small oscillations with a shifted phase between the two working modules.

6.1.1 2-standup

This modular robot is inspired from *3-head2tail_dragon_carpet.avi* [13]. The basic idea is to start from a side position. The robot bends itself and quickly returns in a straight position. This fast movement makes it rotate and reach its moving position. The robot can then move forward using simple oscillations.

One important thing in this modular robot to take into consideration is the alignment of the modules. The centre of gravity may slightly move according to the main axis and it is therefore difficult to predict the motion. Figure 6.1 shows this modular robots.

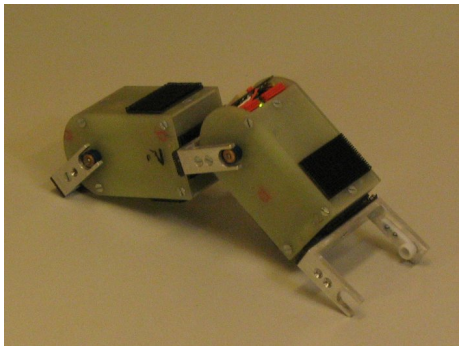

Figure 6.1: A 2 modules example

6.1.2 3-stickworm

This one was discovered almost by accident. As all modular robots we tried were composed of units connected in the same axis, we had the idea to put one module perpendicular to the others. This leaded to an interesting kind of worm which moved laterally.

Quick movements with low amplitude are required so as to get a good locomotion. This modular robot is represented on Figure 6.2.

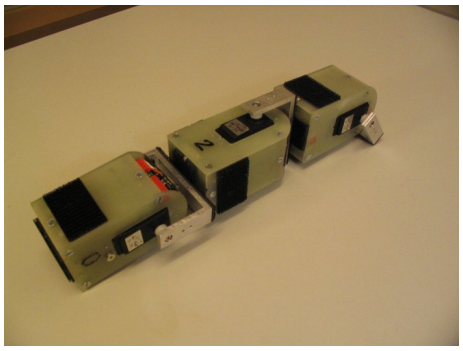

Figure 6.2: A 3 modules example

6.1.3 More examples

YaMoR units combinations are almost infinite. Even with only two or three modules, it is possible to imagine tons of different modular robots.

The number of trajectories for each unit is even greater.

A genetic algorithm would no doubt have helped to discover innovative configuration that nobody would have expected. We are looking forward playing with the results of Daniel Marbach and making Bluemove compatible with his application.

The two examples describe above and more example videos are available on the web site of the project [14].

Chapter 7

Further developments

7.1 Hardware

7.1.1 Mechanic

Servos

The servos used in YaMoR were selected for their low cost and high torque. They have a weak point as shown on Figure 7.1. One of the wheels inside the servo take place in the plastic hole. Because of high load on YaMoR arm, the wheel finally leaves its position and the servo remains blocked.

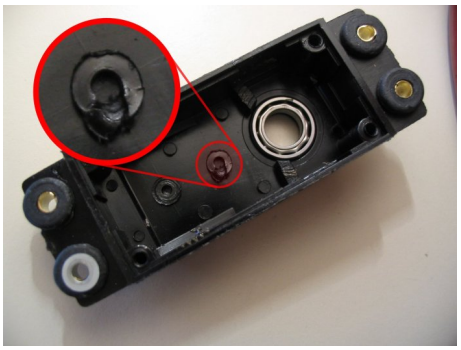

Figure 7.1: Weak point of the servo

Maybe a more expensive servo could be a good investment for an enhanced unit robustness. It would be interesting to test smaller servos. There is not much space left in the unit and adding sensors (see Section 7.1.2) would be problematic with the current design.

Batteries

When designing the YaMoR robot unit, Elmar planned to put the batteries along the servo. This can be seen on Figure 4.1. He defined the size of the module with this idea in mind. When Rico developed the electronic components, he found out that the best solution was to put the boards along the servo. He moved the batteries to the bottom of the module, in a horizontal position. The problem is that the module is 50mm large and the batteries is 48mm long. The batteries are too wide and we could not firmly fix the sides of the unit. We looked for smaller batteries but did not find a good replacement. André Badertscher had the great idea to mill the two panel at the location of the batteries (see Figure 7.2). This way, it was possible to correctly screw the side panel.

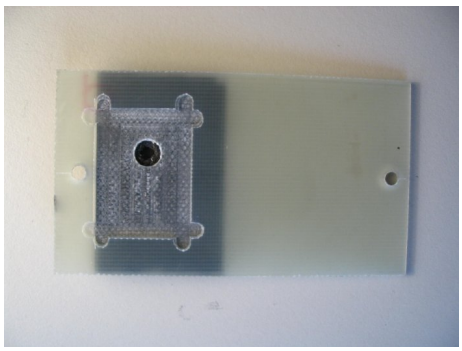

Figure 7.2: Milled side panel

Modules connection

The connection of a module to another is done with Velcro-like tape. There is a male and female version which should be used together. These two types are visible on the arm of the robot on Figure 7.3. This Velcro-like tape allows to quickly reconfigure a modular robot and try new configurations. Although the link between the two modules is strong, it is not stable enough to ensure reliable operations. It would be interesting to try other solutions such as magnetic or mechanical ones.

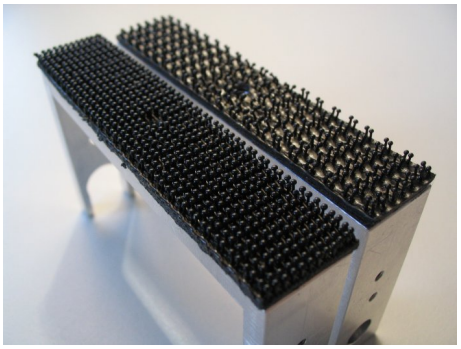

Figure 7.3: Male and female Velcro-like tape

Note that we finally use only the male version (in foreground on Figure 7.3). They connect without any problem but a male/female connection is definitely too weak.

7.1.2 Electronic boards

Sensors

YaMoR units are totally blind. There is no feedback for the module about its environment. Adding sensors like distance sensors (see Figure 7.4), cameras, shocks sensors or temperature sensors to the units could provide useful information about the environment. These data could be used in order to adapt the comportment of the modular robot. We have already taken sensors into account in Bluemove although they are not yet available.

Figure 7.4: Distance measuring sensor

Sensors also imply bidirectional communication. Bluetooth is naturally designed for this kind of communication. Moreover, the Zerial firmware is a serial cable replacement and thus allows transmission in both ways. Most of the work would be on the FPGA part which should send the sensors values to the Bluetooth board. Slight modifications should also be done in the communication package of Bluemove in order to receive the data.

FPGA board

Distributed control The Xilinx Spartan 3 of the FPGA board is clearly under used. It only features a PWM generator although it could support up to six MicroBlaze [37] soft processor cores. One big improvement would be to have completely autonomous modules with all the controls delegated to the FPGA and communicating with each other using the Bluetooth board. However, the actual Zerial firmware will not be adapted for this task because it only allows one connection at a time. A more sophisticated protocol should be implemented. More low-level investigations about Bluetooth protocol are certainly required.

FPGA configuration In the current YaMoR unit, FPGA configuration must be done with Xilinx tools and programming hardware interface. With a lot of modules, this operation is really demanding. A wireless configuration could be done by the Bluetoothboard. The ARM processor of the ZV4002 would be responsible for providing the needed signals. The Zerial firmware should be modified in order to recognise a given information telling it that configuration bitstreams will follow. This probably represents the main work. Some output pins of the GPIO should be dedicated and connected to the corresponding inputs on the FPGA board either on the JTAG or serial lines [1]. This feature will avoid a lot of waste of time.

Software related information are given on Section 7.2. We also recommend the reading of "An FPGA Dynamically Reconfigurable Framework for Modular Robotics" [34].

7.2 Software

FPGA configuration

As explained in Section 7.1.2, Bluemove should be adapted in order to send the bitstreams generated by Xilinx tools. A specific data sequence

[1] FPGA boards provide JTAG and Slave Serial programming mode

should also be defined so as to inform the Bluetooth board software that configuration data will follow. Moreover a "done" information should be sent back to Bluemove. Algorithm 1 represents a possible implementation.

Algorithm 1 Wireless FPGA configuration

```
1: CFG = {modules to be configured}
2: ERR = {}
3: for all m such that m ∈ CFG do
4:    create a connection to m
5:    if connection succeeded then
6:       send the configuration mode sequence
         to m
7:       send the bitstreams to m
8:       if not received done message then
9:          add m to ERR
10:         error message for m
11:      else
12:         remove m from CFG
13:         configuration succeeded for m
14:      end if
15:   end if
16: end for
17: if CFG = {} then
18:   all configurations succeeded
19: else
20:   check modules which are in ERR
21: end if
```

Maximum of allowed robots

The Bluetooth protocol allows a maximum of 7 devices in a piconet and a maximum of 10 fully loaded piconets in a scatternet. This gives a total of 80 devices. However, for physical[2] and technical reason, it will be almost impossible to reach this limit. A 80 units modular robot would be too heavy and the communication latency probably too high. However, we do not have enough working modules to experiment. We estimate that a modular robot using 12 units would be probably the physical limit. We can hardly predict the Bluetooth communication capacity but we estimate that it could support the physical limit. The Bluemove capacity is, however, not limited and depends on the computer capacity.

[2]The servos have limited torque.

Real-time control

The current version of Bluemove allows the user to change keys position while playing. This provides a kind of interactive play with the modular robot. An additional way to perform real-time control would be the use of box connected to other boxes. Figure 7.5 shows an example. The boxes on the left represents inputs (sensors, key pressed, mouse position, ...), boxes in the middle are transformations (addition, multiplication, if-then-else, ...) and finally the box on the right applies the previous transformations on the given parameter. This interactive control is more "robot centred" than changing the keys position ("module centred").

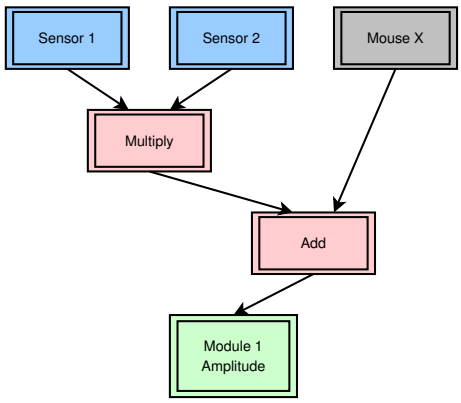

Figure 7.5: Real-time control

Scripts

An interesting feature could be the scripting of curves. Instead of drawing the curve by hand, a scripting language could be used. One of the possible solution is BeanShell [7]. It allows the user to easily write loops, branches and even complete programs. Below is an example of BeanShell usage.

```
% for(i=0;i<10;i++) {
    y = Math.sin(i);
    print(y);
}
0.0
0.8414709848078965
0.9092974268256817
0.1411200080598672
-0.7568024953079282
-0.9589242746631385
-0.27941549819892586
0.6569865987187891
```

```
0.9893582466233818
0.4121184852417566
```

This scripting language does not require variable declaration or strict typing. Thus, the user could draw curves by writing a script and storing the wanted position in a given variable. This would be more efficient to get curves from mathematical functions such as sinus. Moreover, BeanShell is specially designed for Java so it could be integrated without too many problems in Bluemove.

Module templates

When building a modular robot with Bluemove, it is necessary to describe the structure of the modules. At the moment of writing, Bluemove can only control YaMoR units. So instead of describing these units every time, it would be more efficient to have a template module which represents the YaMoR unit.

Import/Export

There is several projects related to YaMoR modular robot at the BIRG. All of them use computer simulations in order to evolve modular robot. The reader is invited to read works of Yvan Bourquin [9] and Daniel Marbach [21]. Here the idea is to retrieve data from simulations and import them into Bluemove in order to recreate curves. This could be really interesting for comparisons between simulation and real world. Another possible feature is to export data from Bluemove and import them into computer simulations. The data can thus be evolved and imported back into Bluemove.

XML would be a perfect data exchange format for this. A possible example would be the following.

```
<bluemove>
    <module name="Head">
        <key time="0.0" position="0.0">
        <key time="1.0" position="-1.0">
        <key time="2.0" position="0.5">
    </module>
    <module name="Tail">
        <key time="0.0" position="0.3">
        <key time="1.0" position="0.5">
        <key time="2.0" position="-0.8">
    </module>
</bluemove>
```

There would be certainly some work to do in the computer simulation softwares so as to make them speak the same language as Bluemove.

Blocks

An interesting feature would be the ability to define blocks in Bluemove. We can imagine repeat, ping-pong or skip blocks. It is often necessary to have a first step in which the modular robot sets its default position. After that, it moves with a cyclic movement and finally goes back to its initial state. This could be achieved by drawing the cyclic movement into a loop block surrounded by the one time initialisation and stopping steps. This is represented on Figure 7.6.

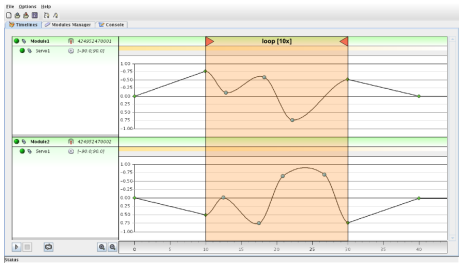

Figure 7.6: Blocks

Chapter 8

Conclusion

We have seen in this report that Bluetooth via an USB stick was the best way to control the YaMoR robot from a desktop computer. Despite the problems we had with Java 1.5, we are pretty confident when we look at the final result that it was the right choice to write a portable and robust Bluetooth application. Thanks to the Rococo stack and the Swing interface, our timelines based Bluemove tool provides an easy control over the robot for the end-user.

Unfortunately, we did not have time to construct more modules due to the long delays to get the ordered electronic components. Nevertheless, we enjoyed the soldering sessions and discussions about mechanical and electronic problems related to this project.

Acknowledgments We want to especially acknowledge Prof. Auke Ijspeert, Alessandro Crespi, Andres Upegui, André Badertscher and the rest of BIRG and LSL members. We are grateful to Rico Moeckel for his precious help with the electronic boards and Elmar Dittrich for his work on the module. We thank Ariane Knuesel for reading this paper and making useful suggestions to improve its readability.

Appendix A

Bluemove short manual

This chapter quickly describes the user interface of the Bluemove application. It is divided into three parts: the first one explains how to install the software and its requirements, the second one gives an overview of the user interface and, finally, the third part provides a step by step example which shows how to create a simple project with Bluemove.

A.1 Installation

In order to run properly, Bluemove requires some additional softwares and libraries. The following list summarizes the requirements for this application:

- A Linux system on a x86 machine
- J2SE 5.0 (Java JDK 5.0 or JRE 5.0)
- Java Bluetooth stack JSR-82 compliant

Other needed libraries (log4j, xerces, ...) are already included into the application. You do not need to care about them.

A.1.1 J2SE 5.0

Bluemove needs the J2SE 5.0 from Sun Microsystems. Bluemove takes advantage of the latest features of J2SE 5.0. In order to get the best performance, it is important to use the latest J2SE 5.0 version. You can choose to install the *J2SE Development Kit (JDK)* or the *J2SE Runtime Environment (JRE)* depending on whetever you are a developer or not. You can find the latest J2SE 5.0 here [22]. We will not present the installation of the Java Environment. Instructions are available in the downloaded archive.

In order to see if the J2SE 5.0 is correctly installed, open a new terminal and type the following command.

```
$ java -version
java version "1.5.0_01"
...
```

You should get a string with `"1.5"`. If not, please reinstall the J2SE environment and follow carefully the instructions.

A.1.2 Java Bluetooth stack

In order to communicate with the module, you need a Java Bluetooth stack . There is a lot of different stacks available for Linux, Windows and embedded systems. Rococo [31] offers a free implementation of the JSR-82 [19] called *Impronto* for University students. Other Java Bluetooth stacks can be found here [2]. By default, Bluemove is compiled to run with the Rococo Impronto stack.

Rococo Impronto

Rococo distributes its Impronto Developer Kit for students under a license which does not allow us to distribute it with our software. Instead you must fill in the form available on Rococo website[1] and wait for the confirmation e-mail. After you get the `impronto-1.3-1.i386.rpm`, you can simply install the Rococo Bluetooth stack with `rpm -ivh impronto-1.3-1.i386.rpm`. This should install the required BlueZ libraries [1]. If you encounter any problems, please make sure that BlueZ is correctly installed and configured on your system. Further information about the Impronto installation can be found in the file `impronto/user_guide.pdf` of the Bluemove archive.

Finally, you have to read and agree the `LinuxLicense.txt` file. If you agree with the

[1] http://www.rococosoft.com/registration_linux.html

license, you must copy it into the same directory as `Bluemove.jar`. The Impronto stack will not work without this license file.

A.1.3 Bluemove

Bluemove is distributed as a JAR archive. Everything you need to run the application is embedded into this file. Thus, the only thing you need to install Bluemove is to copy the JAR archive named `Bluemove.jar` in a directory. We recommend to unpack the Bluemove archive in your home directory and run the application directly from that location.

As seen above, do not forget to copy the Impronto license file in the same directory as Bluemove.

A.1.4 Launching the application

Once the J2SE 5.0, the Java Bluetooth stack and Bluemove are installed, you can finally launch the application. To start with, you need to initialise your Bluetooth device. Open a terminal and type the following command as "root".

```
# hciconfig hci0 up
# hciconfig hci0
hci0:   Type: USB
        BD Address: 00:0C:76:48:87:7A
        ...
```

Your "BD Address" should not be equal to `00:00:00:00:00:00`. If so, please check your kernel configuration and make sure that the needed modules are loaded or compiled into the kernel.

You can now go into the Bluemove directory and type the following command.

```
$ LD_LIBRARY_PATH=/usr/lib java \
-classpath .:Bluemove.jar \
birg.bluemove.main.core.Bluemove
```

The path `/usr/lib` represents the location of the file `libimpronto.so` which is normally installed in `/usr/lib`.

You should now see the Bluemove splash screen and then the main screen. If not, please check that all the previous steps were successfully completed.

You are ready to discover the power of Bluemove and the pleasure of modular robotic.

A.2 User interface

When you start Bluemove, you see the main view of the application which contains four important parts:

- the menu bar
- the tool bar
- the tabbed pane
- the status bar

The tabbed pane contains several tabs which are respectively used to draw the module curves, to manage the modules, and to view the log messages. In the next section, we will take a close look at the different items available in the menu bar.

A.2.1 The menu

The menu bar is located at the top of the window. It is composed of three menus, each one containing several menu items.

The File menu

This menu allows you to do the main operations on a project. You can create a new project, open or save a project. You can also quit the application. The last opened or saved project is remembered and can thus be quickly reopened.

The Options menu

This menu allows you to set the properties of the application. All the settings are locally stored on your computer.

The Help menu

This menu contains a menu entry called "Help". It opens your favourite browser and redirects it to the Bluemove help site. The menu item "About" displays information about your system.

A.2.2 The tool bar

The toolbar contains shortcuts for useful actions like creating a new project, saving or opening a project.

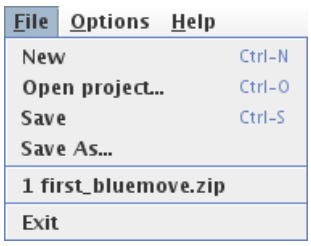

Figure A.1: Menu

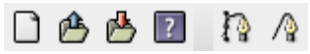

Figure A.2: Tool bar

A.2.3 Timelines tab

The timelines tab is the heart of Bluemove together with the modules manager. You can draw trajectories for modules in the timelines listed there.

Left part

Each module is composed of one or more actuators[2]. Sensors can also be part of a module. At the moment, sensors are displayed but no management is available. The name and the address of the module is displayed.

Right part

In this right part, you will be able to draw the trajectories of an actuator. There are two key types:

- Linear key
- Hermite spline key

You can mix these two kind of keys in the same actuator timeline without any restriction. The interpolation algorithm will take care of curves composed of these two types of keys.

A.2.4 Modules Manager tab

Within the modules manager you can declare the different parts of your modular robot. You can describe every module, set its name and address. Each module can have several actuators. These components have a name and a range which will be used to send the position to the module.

Please notice that the YaMoR servo name is "Servo1". You *must* use this name for the actuator when using a YaMoR unit.

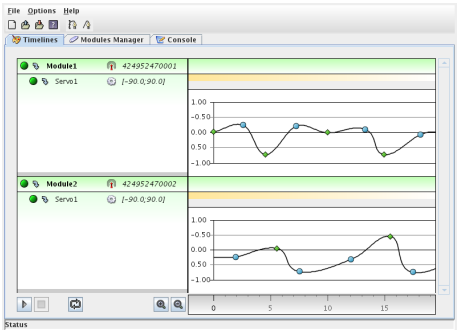

Figure A.3: Timelines

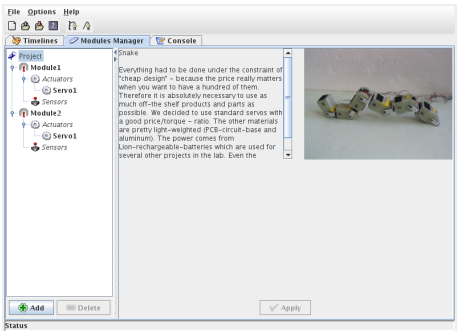

Figure A.4: Modules Manager

A.2.5 Console tab

The console tab displays the log messages generated by Bluemove. You can set the level of the output and the refresh rate. There are five levels of messages:

- DEBUG: messages useful in order to debug the application
- INFO: information messages
- WARN: a problem leading to an error occurred
- ERROR: something that should not have happened
- FATAL: unrecoverable error

Each level is displayed in a distinct colour with its specific icon. This allows you to quickly diagnose what appended in the application.

[2] A YaMoR module contains one actuator (a "S-71" servo)

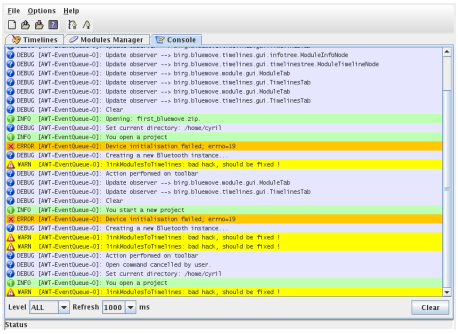

Figure A.5: Console

A.2.6 The status bar

The status bar displays information about the current action. It is also used to display important messages.

Figure A.6: Status bar

A.3 Tutorial: a small example

In this section, you will get to know how to create a simple Bluemove project. We consider that you have already installed the application and that you are able to start it. You will also need to have two functional YaMoR units.

A.3.1 Step one: add a module

First of all, start the Bluemove application. After the splash screen, you get a window similar to Figure A.7.

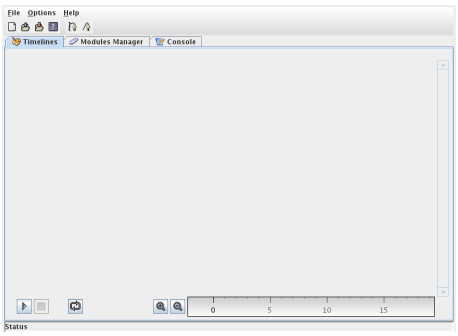

Figure A.7: Main screen

Now you have to describe the modular robot physically. In order to do this, switch to the modules manager tab. At the top of the left part, you should see a small icon representing a salamander followed by the text "Project". Click on this label. This should enable the "Add" button at the bottom of the left part. Click on "Add". A new module called "Module1" is inserted.

A.3.2 Step two: edit a module

Click on the label "Module1". You should get a screen similar to Figure A.8.

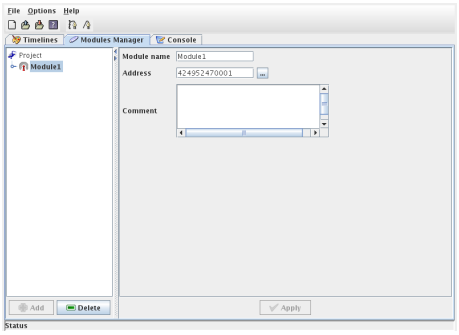

Figure A.8: Modules manager

Now, on the right part, edit the "Module name" field and type "Head". Enter the address corresponding to this module. Here we suppose that our module has the address "424952470001"[3]. Edit the "Comment" text area and type "This is the head of our modular robot.". You can now click on the "Apply" button. Your window should look like Figure A.9.

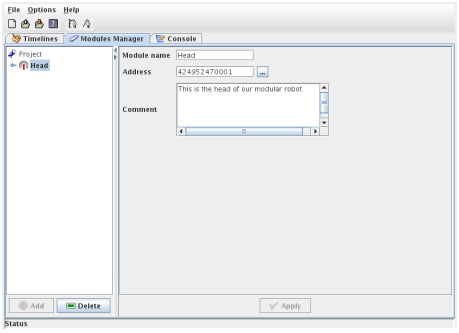

Figure A.9: Module edition

[3]You can also use the inquiry function to found the available modules.

A.3.3 Step three: add an actuator

A module without an actuator can not move. The YaMoR units have one actuator named "Servo1". Let us add it to our "Head" module. Expand the node "Head" on the left part. Two nodes should appear: "Actuators" and "Sensors". Click on "Actuators" and then click on the "Add" button. You get a new actuator called "Actuator1". Click on the label "Actuator1". You notice that the right part has changed. Edit the field "Actuator name" and type "Servo1". This is the servo name of the YaMoR units. Set a "Min value" of "-90" and a "Max value" of "90". Click on "Apply". Your window should be similar to Figure A.10.

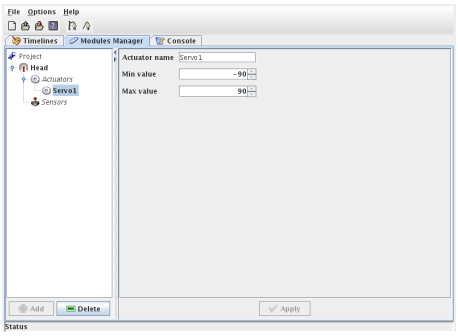

Figure A.10: Actuator edition

A.3.4 Step four: add another module

You should now be able to add another module without any problems. So restart at step one and add a new module called "Tail" with one actuator named "Servo1" which range is between "-90" and "90". We suppose that this module address is "424952470002". You should end with a screen similar to Figure A.11.

A.3.5 Step five: draw a module trajectory (spline)

We have finished the physical description of our modular robot. In order to see our animal moving, we have to draw the trajectory of its actuators. Here you will see the power of this software. Let us move to the "Timelines tab". Your screen should look like Figure A.12.

Expand the "Head" module by clicking on the small blue arrow located close to the label. Do the same with the actuator "Servo1". Now you

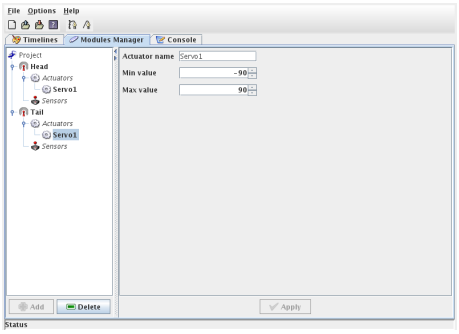

Figure A.11: Second module

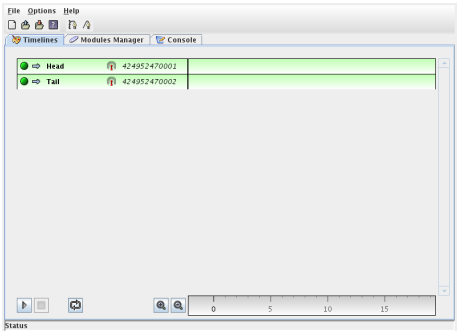

Figure A.12: Timelines tab

should see the timeline on the right. Enlarge the timeline a bit by dragging the bottom line. Select the "Spline tool" in the tool bar. Draw the same curve as the one on Figure A.13. You surely noticed that spline keys are represented by a blue round.

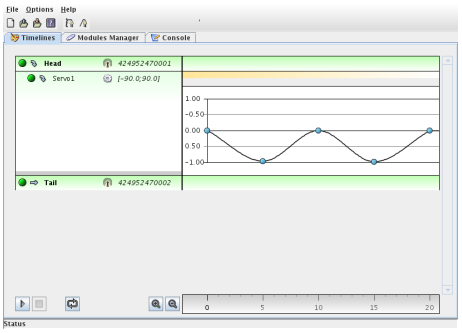

Figure A.13: The first curve with spline keys

A.3.6 Step six: draw a module trajectory (linear)

Repeat the above step for the "Tail" module. This time you will use only linear keys. Select the "Linear tool" in the tool bar. Linear keys

are represented by a green rhombus. You should end with a window similiar to Figure A.14.

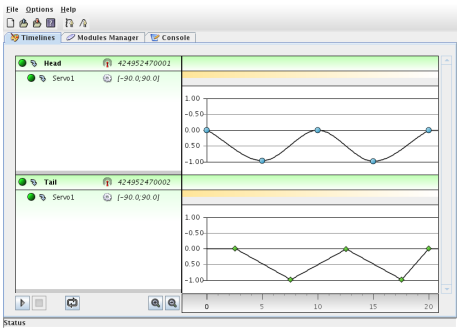

Figure A.14: The second curve with linear keys

A.3.7 Step seven: save the project

Before going further, it is high time to save your project. Click on the menu named "File" and select the item "Save As...". A file chooser dialog appears. Choose the directory you want and name your project "first_bluemove.zip" like on Figure A.15. You probably noticed that Bluemove projects are simply ZIP archives.

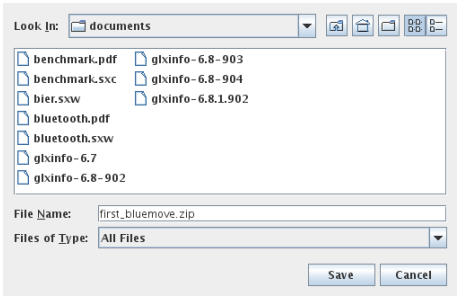

Figure A.15: Saving a project

A.3.8 Step eight: play

We arrive at the last step of this tutorial. Power on the modules and program the YaMoR FPGA. This operation is described at Section 7.3.3 of [28]. Build the modular robot as shown on Figure `\ref{fig:step-10}`. Click on the "Play" button located at the bottom of the timelines tab. After a few seconds, the modules should start to move. If so, congratulations, you have successfully completed this tutorial. If nothing is moving then maybe your Bluetooth device is not correctly initialised. You can switch to the "Console" tab and set the debug level to "ALL". Retry this step and look at the console.

A.3.9 Conclusion

We briefly looked at the basic operations of Bluemove. There are a lot of other features that were not described in this tutorial. However, you should be able to discover them on your own. Do not forget to periodically check the help website [14]. Lots of information on Bluemove are available there.

Bibliography

[1] Bluez. http://www.bluez.org.

[2] Java bluetooth. http://www.javabluetooth.com.

[3] Adobe. After effects. http://www.adobe.com/products/aftereffects/main.html.

[4] Aibo - Sony Japan http://www.jp.sonystyle.com/Product/Aibo/Ers-311b_312b/Store/.

[5] ARM. Processor core. http://www.arm.com/products/CPUs/index.html.

[6] ASL Flying robots http://asl.epfl.ch/research/projects/AdaptiveVisionbasedFlyingRobots/index.php.

[7] BeanShell. Lightweight scripting for java. http://www.beanshell.org.

[8] BIRG. Amphibot i: an amphibious snake robot. http://birg.epfl.ch/page53468.html, 2003.

[9] Yvan Bourquin. Self-organization of locomotion in modular robots. http://birg.epfl.ch/page53073.html.

[10] Jennifer Bray and Charles F. Sturman. *Bluetooth: Connect Without Cables*. Prentice Hall, 2000.

[11] L2FProd components. L2fprod components. http://www.l2fprod.com.

[12] Robot Labs http://www.robot-labs.jp/top.html.

[13] Elmar Dittrich. Modular robot unit - characterisation, design and realisation. http://birg.epfl.ch/page50163.html, 2004.

[14] Kévin Drapel and Cyril Jaquier. Using bluetooth to control a yamor modular robot. http://birg.epfl.ch/page56602.html.

[15] Jean-Philippe Egger. Roboka. http://birg.epfl.ch/page48822.html.

[16] The Apache Software Foundation. Xalan java. http://xml.apache.org/xalan-j.

[17] The Apache Software Foundation. Xerces2 java parser. http://xml.apache.org/xerces2-j.

[18] Bruce Hopkins and Ranjith Antony. *Bluetooth for Java*. APress, 2003.

[19] JCP. Jsr-82. http://www.jcp.org/aboutJava/communityprocess/review/jsr082/.

[20] JetBrains. Intellij idea, an intelligent java ide intensely focused on developer productivity. http://www.jetbrains.com/idea.

[21] Daniel Marbach. http://birg.epfl.ch/page32031.html.

[22] Sun Microsystems. J2se 5.0. http://java.sun.com/j2se/1.5.0/download.jsp.

[23] Sun Microsystems. J2se 5.0 in a nutshell. http://java.sun.com/developer/technicalArticles/releases/j2se15/.

[24] Seiko Epson Corporation - Monsieur 2 http://www.epson.co.jp/e/newsroom/news_2003_03_10.htm.

[25] Morph 3 (Atom Project) - http://www.japantimes.co.jp/cgi-bin/getarticle.pl5?nn20030820b8.htm.

[26] Murata about Morph3 - http://www.murata.com/articles/ta0255.pdf.

[27] Rico Möckel. Design and construction of an autonomous modular robot unit with bluetooth and fpga. http://birg.epfl.ch/page53075.html, 2004.

[28] Rico Möckel. Getting started user guide. http://birg.epfl.ch/page53075.html, 2004.

[29] J.D. Nicoud and J.C. Zufferey. Toward indoor flying robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 787–792, 2002.

[30] Moppi Productions. Demopaja. `http://moppi.inside.org/demopaja/`.

[31] Rococo. `http://www.rococosoft.com`.

[32] Seiko Epson Corporation - Lightweight Helicopter `http://www.epson.co.jp/e/newsroom/news_2004_08_18.htm`.

[33] Karl Tauber. Jformdesigner, wysiwyg gui designer for java swing user interfaces. `http://www.ktauber.com/formdesigner.html`.

[34] A. Upegui, R. Moeckel, E. Dittrich, A. Ijspeert, and E. Sanchez. An fpga dynamically reconfigurable framework for modular robotics. 2005.

[35] W3C. Document object model (dom). `http://www.w3.org/DOM`.

[36] W3C. Xml path language (xpath). `http://www.w3.org/TR/xpath`.

[37] Xilinx. Microblaze soft processor core. `http://www.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?key=micro_blaze`.

[38] Zeevo. System-on-chip solutions for bluetooth and other wireless communications applications. `http://www.zeevo.com`.

[39] J.C. Zufferey, D. Floreano, M. van Leeuwen, and T. Merenda. Evolving vision-based flying robots. *Proceedings of the 2nd International Workshop on Biologically Motivated Computer Vision (LNCS)*, 2002.

Index